HAL
open science

Musical Representation of Sound in Computer-Aided Composition: A Visual Programming Framework

Jean Bresson, Carlos Agon

To cite this version:

Jean Bresson, Carlos Agon. Musical Representation of Sound in Computer-Aided Composition: A Visual Programming Framework. Journal of New Music Research, 2007, 36 (4), pp.251-266. 10.1080/09298210801969145 . hal-00683475

HAL Id: hal-00683475

https://hal.science/hal-00683475v1

Submitted on 26 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Musical Representation of Sound in Computer-Aided Composition
A Visual Programming Framework

Jean BRESSON
IRCAM – CNRS UMR 9912
Music Representations Team
1, place I. Stravinsky, 75004 Paris, FRANCE.
Tel (+33) 1 44 78 16 57
e-mail: jean.bresson@ircam.fr

Carlos AGON
IRCAM – CNRS UMR 9912
Music Representations Team
1, place I. Stravinsky, 75004 Paris, FRANCE.
Tel (+33) 1 44 78 48 33
e-mail: carlos.agon@ircam.fr

This is a preprint of an article submitted for consideration in the *Journal of New Music Research*, 36(4), 2007, Copyright Taylor & Francis; *Journal of New Music Research* is available online at: http://www.informaworld.com/smpp/content~db=all~content=a793093105.

ABSTRACT: *This article addresses the problem of the representation and creation of sound by synthesis in the context of music composition, as seen from the computer-aided composition (CAC) perspective. An important theoretical basis of this work is the concept of computer modelling, discussed in relation to the notions of sound representation and music composition. Modelling sound as a signal is extended to the musical domain by considering as an alternative modelling composition as an activity that aims to produce sounds. The visual programming paradigm is adopted for the representation and conception of the composition models, and therefore for the musical representation of sounds. A composition framework dedicated to electro-acoustic music and sound synthesis integrated in the OpenMusic CAC environment is presented. Temporal issues are also discussed and are the object of specific developments.*

1. Introduction

The recording of sound signals has renewed the musical approach to sound material. Once stored on a physical support, sound could be observed, manipulated, and considered as a concrete object (Schaeffer, 1977). Meanwhile, transduction technologies allowed the creation of acoustic signals from electric signals, representing another important step: musical sounds could then be produced not only using acoustic instruments, but also using the logical modules of electronic devices. Since then, *composing sound* has become a new compositional challenge (Stockhausen, 1988a).

Computers and digital technologies represent another significant step in this direction. The computer helps to improve and develop new sound synthesis techniques, but also turns sound into a "composable" object. On a digital support, sound is represented by a sequence of symbols (binary digits encoding sampled and quantized values from the signal), which provide accurate visibility, edition, and manipulation possibilities. In theory, the range of sounds that can be created with a computer becomes almost unlimited: it boils down to the matter of computing the sample values. Sound is therefore likely to be composed like any other musical structure. Being the intentional and the final perceived form of music it might even be considered as the overall musical structure.

This theoretical standpoint seems promising, and has inspired most of the computer music research carried out over the past 40 years. Many technologies have been developed in the field of digital sound processing (DSP) and now provide wide-ranging means for synthesizing artificial sounds or manipulating recorded signals—see (Moorer, 1977; Roads, 1996) for a survey.

In the meantime, computer composition research focused on higher-level musical issues and resulted in composition systems and languages dedicated essentially to score creation, i.e. following implicitly the traditional schema of instrumental music. In this context, it is generally assumed that a musician will read this score and play an instrument in order to produce sound, or at least this is the presumed underlying process behind the representation: the score carries information about certain parameters of sound (event onsets, pitches, durations, intensity) and the main part of the signal information is concretely determined by the performance of the music. The role of a composition system therefore basically consists in the formalisation and representation of the processes leading to the creation of a score composed of specific musical objects. The same conceptual scheme applies when working with MIDI devices or software synthesizers, whose synthesis processes' inputs are basically the same (and sometimes even more restrictive) than those of an acoustic instrument. In this case, the notes connect the fields of composition and interpretation, and from indivisible symbolic primitives in the first one, they become complex sonic structures in the acoustic world.

With generalized sound synthesis, however, the compositional representation is converted into a real sound signal by a computer process. The multiplicity of existing sound synthesis techniques now makes it possible to extend the parameters of this process to any kind of data structures. In addition, as sounds must be entirely described (i.e. written) by the composer, the knowledge, experience, and sensibility of the human performer—as well as the mechanical properties and behaviours of the instruments—need to be balanced in order to reach equivalently rich sonic results, which leads to an increase in the complexity of the data and processes to be dealt with by the composer.

In the majority of early computer composition systems, this alteration of the musical scheme induced by DSP technologies were not truly and formally integrated. In consequence, a solution for actually composing sound, for *penetrating sounds with the composer's formal ideas* (Stockhausen, 1988b) has not actually been found.

Today, this issue raises an increasing interest from composers and computer music researchers. From the signal processing point of view, it consists of defining high-level representations of the sound signal according to a given underlying synthesis technique which corresponds to new parameters to be integrated in compositional processes. Hence, the tools for the control of sound synthesis generally provide (graphical) interfaces designed for an easier setting and organization (most often, in time) of these parameters. On the other side (i.e. computer composition systems), the focus is on the manipulation of abstract high-level parameters subsequently converted to low-level data and piped to specific sound synthesis software. In both cases, a clear boundary generally separates compositional and sound-related issues.

The approach we put forward here rises from computer-aided composition, therefore from one of the two previously mentioned domains. However, it adopts an original standpoint as it does not try to build on predefined high-level synthesis parameters, but rather builds abstract sound representations based on compositional concerns. The issue of the representation of sound is addressed simultaneously in relation to the notion of computer modelling, considered as a way of formalizing implement compositional processes, and to that of visual programming presented as a possible means for the creation and representation of the compositional models in a computer system. This will provide, as we hope to demonstrate, a generic framework for sound composition and may help composers to develop their musical conceptions freely both within the field of formal composition and of sound synthesis and processing, and eventually to put these fields in relation to each other following a global musical thought.

In section 2 we first discuss the general question of the representation of sound from a theoretical standpoint and particularly the possible connections between computer and musical representations. Section 3 provides a general discussion about computer-aided composition and the notions of computer and compositional models. OpenMusic, a visual programming language dedicated to music composition, will be presented in this section. In section 4 we will consider extending the CAC conceptual framework toward sound-related issues so as to consider sound as the object of compositional models. Previous related works will also be discussed. Section 5 will then introduce the

main development lines of *OM-Sounds*, a project developed in OpenMusic including new features for sound manipulation, representation, and synthesis. In this section the possible solutions for dealing with sound synthesis parameters starting from abstract compositional processes will also be considered. Finally, high-level representations and temporal issues are discussed in section 6, as well as the corresponding developments in the *OM-Sounds* framework.

2. The representation of sound

2.1. Structured representations

The representation of sound is a major underlying issue in the general problem of sound composition. Digital sounds are basically represented by a sequence of sound samples, which is generally not a suitable musical representation. This representation does not carry any valuable knowledge about the sound, considered as a musical structure to be defined and manipulated in a compositional process. Moreover, to create sound by chaining sample values is not a viable compositional task. More structured representations are therefore needed.

Different techniques for analysing or synthesizing sounds (generally called sound analysis/synthesis models) provide such structured representations (De Poli *et al.*, 1991). The classification of these models vary according to authors and intents, but it is possible to identify some general categories. The dominant category certainly corresponds to spectral models, which provide frequential representations of the signal: a sound is decomposed in frequency bins, and the energy carried in each of these frequency bins is considered. These models can be implemented by means of digital filters and oscillators, or by using the Fourier transform and its extension to non-stationary signals, the short-term Fourier transform – STFT, see (Allen and Rabiner, 1977). Subclasses of spectral models are, for instance, the additive models (McAulay and Quatieri, 1986) where the sound is represented with a set of partials (individual sinusoidal components), source/filter models where the sound is represented with a source signal associated to a filter bank structure, or granular models (Roads, 2002) where sound is considered as a cloud of primitive sound grains in the time/frequency plane. Another general type of sound synthesis model is called the abstract model. In abstract models, the sound signals are described using mathematical expressions. The more famous example is the frequency modulation (FM) synthesis (Chowning, 1973). Signal (or "time domain") models are based on pre-existent signals, generally represented in wave tables, which are processed in order to produce sounds (e.g. sampling, mixing, wave table synthesis, etc.) Finally, physical models are based on a mechanical description of virtual structures and actions, then converted into mathematical systems solved by the computer in order to create the corresponding sounds (Florens and Cadoz, 1991).

2.1. Symbolic representations

All the aforementioned models provide structured sound representations, which nevertheless do not insure any compositional significance. Composition requires the introduction of the notion of *symbolic* representations, that is, representations likely to be integrated in a general (musical) thought or a compositional approach. Generally speaking, a symbolic representation consists of the application of a set of signs used to substitute more or less complex realities in order to reduce, structure, and organize the information (Chazal, 1995). Symbols help memorize this information, but also to read, to understand or impart knowledge, to think about the represented object, or—and particularly in computer systems—to perform calculus on it.

Symbolic representations are therefore likely to be part of and to undergo mental processes as well as computer processes. A distinction must however be made here between what would be called symbolic at the computer level and at the composer's musical level. Bits are symbols in the computer: they correspond to a physical phenomenon interpreted as binary values. They can be combined in order to constitute bytes, characters, or numbers which are yet higher-level symbols, but neither a bit nor a number will usually be considered as musically relevant symbols likely to be interpreted and handled by a composer. However, their combination creating a note-like representation (pitch value, amplitude, etc.) might constitute a musical representation. This representation is compact and

structured enough to carry meaningful information and to be integrated mentally in compositional processes. It corresponds to a musical tradition and to a well-defined musical element.

Indeed, the notion of symbolic representation for music composition is often related to the traditional musical notation system and to the corresponding score representation. This representation corresponds to instrumental practice and to a formal conception of musical sound divided in timed events, essentially defined by a pitch (i.e. notes) organized (particularly in time) by compound structures (chords, voices, etc.) The musical objects and structures in the score are symbolic representations for they carry a musical sense and are likely to be handled (read, understood, written) in a given (musical) context (e.g. composition or music interpretation.)

On the contrary, the numbers that constitute a digital waveform in a straightforward representation of sound are not considered as symbolic elements. Individually, these elements do not hold any specific meaning for a composer and can hardly be put in relation with one another in the compositional process. We could relate these representations to the *subsymbolic* domain discussed in (Leman, 1993; Camurri, 1990) if (with some simplification to the original concept) we consider that this domain brings together processes and representations that carry relevant meanings for the bare computer processing of the data, while symbolic representations holds meaning for the system user (e.g. a composer).

Compared to a digital signal, the spectral representation is a yet more structured representation. The separation of the frequential and temporal information brings additional knowledge and some more musical sense is given to the overall information. It allows more powerful manipulations on sound, in the computer side as well as for composers' formal invention (Arfib, 1991). Further on, this information can be reduced: in the additive representations, the signal is represented by a set of partials, each representing a perceptually meaningful part of the sound. In certain circumstances, such compact and structured representations are thus likely to be considered as symbolic compositional elements. However, general sound synthesis parameters rarely reach this symbolic level: in a few seconds of sound, the parameters for any sound synthesis system can contain thousands of elements, essentially related to complex time-varying functions. A symbolic representation for composition will therefore involve personal and context-dependent decisions, and multi-layered abstractions will be required in order to ensure this symbolism at the various structural scales.

3. Computer-aided composition: modelling music composition

3.1. Computer-aided composition and music representations

Once a symbolic representation is defined, computer composition tools can help users to create music by providing alternative and improved ways to write the corresponding scores (using graphical user interfaces, editors, etc.) They would then follow the straightforward approach of the prevalent score editing tools and software, in which the generative part of the compositional process is carried out (mentally or using other supports) outside the computer system.

Computer-aided composition aims at integrating this generative part, insofar as it relates to calculus and thus is likely to be represented with programs. Musical formalisms, compositional concepts or processes are therefore considered in relation to computer tools and formalisms. As opposed to other domains, however, music composition has the particularity of implying creative processes that are not well understood and formalized. As a consequence, providing composers with computer tools and programs that would *make music* does not actually make sense (at least until these creative processes can be formally described). The problem is therefore considered from a slightly different angle: computer-aided composition tools aim at providing the composer with means to use programs for musical creation by implementing previously self-formalized ideas (Assayag, 1998). According to the terminology mentioned in (Girard *et al.*, 1989), we address here the problem of the sense and denotation of a program: the denotation being considered as a result, while sense deals with the way to obtain this result. Traditionally, the program semantics promoted the operating side, forgetting about syntactic aspects: the semantic of a program is determined by its result. In a creative process, however, the absence of a "correct" result (denotation) can bring the compositional process (sense) to the forefront. A composer might be more interested in exploring a processing space, which

takes place in the program writing, than in resolving a mere computing problem. The program writing can thus carry a semantic content: while the score denotes the piece, the program gives it sense.

An important idea of computer-aided composition is that as compositional processes can be (at least partially) carried out and represented by programs, then programming languages are the most relevant compositional supports for computer-musicians. Composers should then be considered as programmers rather than just as simple users of a given program. Indeed, most computer-aided composition environments have been designed in the form of specialized programming languages (Hiller, 1969; Rodet and Cointe, 1984; Taube, 1991; Dannenberg, 1989). In these environments, programming techniques and paradigms are applied to music creation, and provide broad expressive power. Functional languages have proved to be useful in this scope (Dannenberg *et al.*, 1997). The functional formalism derived from the lambda calculus (Barendregt, 1984) considers data and programs in a similar way; the abstraction mechanism makes data the source of programs, which can, in turn, generate new data. Considering a program as data thus allows to focus on the writing of programs. Other different programming paradigms were also considered for composition, such as object-oriented programming (Pope, 1991), or constraints programming (Laurson, 1996; Rueda *et al.,* 2001).

The development of human-computer interfaces, and particularly the emergence of visual programming languages such as PatchWork (Laurson and Duthen, 1989) and its successors OpenMusic (Assayag *et al.*, 1997a) or PWGL (Laurson and Kuuskankare, 2002) have considerably increased the creative potential of computer-aided composition environments. If we consider that the compositional process carries the sense of a musical piece, visual programming languages are then an appropriate means for expressing this sense. Programming is made accessible without requiring highly technical skills and the visualization and editing tools allow for a graphical representation of the processes and data structures involved (Assayag, 1995).

3.2. OpenMusic: a visual programming language for music composition

OpenMusic (OM) is a visual extension of the Common Lisp / CLOS (Common Lisp Object System) programming language (Steele, 1998). As it is used as a general support for the works we will present in this paper, we will give here a brief description of this environment.

In OpenMusic, programs are represented in *patch* editors, where the composer/programmer assembles and connects functional units using graphical boxes. These boxes represent calls to the functions (basic Common Lisp primitive functions or user-defined functions) defined in the environment, and the connections between the inputs and outputs of these boxes define a graph that represents the functional composition of the program. The prevailing paradigm in OM is thus that of functional programming, although as we shall see hereafter, it also encloses substantial "object-oriented" aspects. The program represented in a patch can be evaluated at any point; which initiates the recursive evaluation of the functional calls following the composition of the program, and corresponds to its execution.

In addition, this visual programming environment is provided with special functions and data structures (classes), allowing to head programming toward musical applications. Figure 1 shows an example of a patch created in OpenMusic.

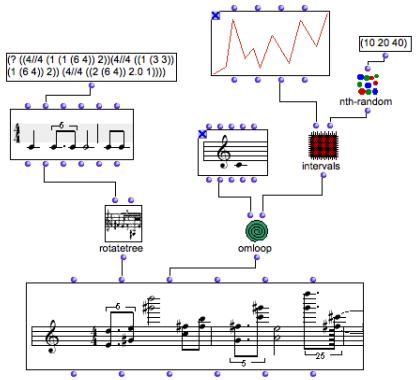

Figure 1. A patch (visual program) in OpenMusic: a score is generated from a rhythmic tree structure and a pitch generation algorithm. Each box represents a functional call. Factory boxes are used to represent the data structures (note, voices, break-point function.) The "interval" box encloses an abstracted processing unit (i.e. another patch.) and the "omloop" box contains the (graphical) description of an iterative processing of its input data.

The language also provides a set of graphical control structures, such as iterations and conditional controls, as well as the possibility to carry out further programming concepts like abstraction or recursion.

Functional abstractions let some elements of the programs become variables, which leads to the definition of functional objects. An abstraction is created by defining inputs and outputs connected to these variable elements and to the results of the patch. The resulting abstraction can then be embedded (applied) into other (higher-level) programs or abstractions and thereby be used in different contexts. This is a fundamental feature in a computer system for composition (and in a programming environment in general) since it makes it possible to organize and progress in the successive complexity and/or structural layers while carrying out compositional processes.

In addition, OM takes advantage of the potentialities of the object-oriented programming from CLOS by allowing the user to create classes, methods, relations of heritage (Agon and Assayag, 2003). Specialized libraries have also been created for constraints programming (Laurson, 1996; Rueda and Bonnet, 1998; Truchet *et al.*, 2003).

Various predefined classes (in the sense of object-oriented programming) of musical objects are available (e.g. notes, chords, chord sequences, break-points functions, etc.) and make it possible to manage musical material within the visual programs. They are represented through the concept of *factory* boxes, i.e. functional calls generating instances of these classes. These boxes allow the instantiation of objects by providing individual access to the values of their public slots, and to use these values in downstream computation processes. They also make it possible to visualize and edit the content of the most recently created instance via specialized graphical editors (e.g. score editors). The factories thus make possible to generate, store, and visualize the state of a given structured data set at a given moment of the calculus (i.e. in a given position of the calculus graph defined in a patch), and at the same time allow the manual editing of these data thanks to the associated editor, which constitutes a fundamental entry point for the intervention of the user (composer/programmer) in the calculus (Agon and Assayag, 2002). The dual representation and control over both the musical objects and data, and the processes in which they take part (or which they come from) constitutes another essential feature which distinguishes this environment from pure programming environments and emphasizes the particular approach of CAC that we will detail hereafter.

Interested readers can find more information about OpenMusic and the concepts mentioned in this overview in (Agon, 1998; Assayag *et al.*, 1999; Bresson *et al.*, 2005). For concrete musical

applications, they might have a look at (Agon *et al.*, 2006, Bresson *et al.*, 2008), which report a collection of articles written by composers and describing the various ways OpenMusic can be used for composition.

3.3. Computer modelling and compositional models

Although the first uses of the computer for music composition practically date back to the birth of the computer, the current position of computer-aided composition that we wish to emphasize here presents significant differences with the former conceptions. Indeed, most of the early compositional works carried out with computers related to a certain "automated" approach, in which programs were designed following various formalisms (functional, rule-based, stochastic, etc.) and run in order to produce music. Our present conception presents a slightly different point of view, and promotes the interaction between the composer and the program execution by the manual setting of the initial or intermediate material, the trial-and-error, and the progressive construct and complexification of the musical processes developed under the form of programs.[1] This conception is also closely related to the emergence of graphical tools and editors in the composition environments and gives rise to a new way of imagining the use of computing tools by the composers. As we shall demonstrate forthwith, it can be related to the notion of (computer/compositional) modelling. This idea is fundamental in our conception of CAC, and may help understand the general approach adopted for the problem of the musical representation of sound, or more generally, that of sound composition.

By modelling an object we mean considering it through a set of concrete or abstract elements referring to this object and interacting in a formal discourse (Berthier, 2002). This object is considered through certain components that interact in the model structure. The model therefore reflects a particular way of thinking about an object; the same object can be modelled in many different ways according to subjective approaches to this object. Hence, a model representation (implemented in a computer program, for example) puts forward some particular aspects of this object. This representation is not a neutral tool and suggests the possible operations to be carried out with the model by bringing out a particular operating viewpoint and a user interaction framework. This subjective character therefore supports the aesthetic interest of an open and unrestricted model representation (Risset, 1991; Eckel, 1992).

As mentioned above, the term of model is used in the DSP domain to classify the different types of existing techniques available for representing or synthesizing sounds. A certain way to consider sound corresponds to each of these techniques that explicit its structure: *a sound model allows the production of a sound as well as its formal representation* (Eckel, 1993). The synthesis models are implemented in the form of programs in computer systems, that is, processing tools which can produce sounds from a set of values which we call parameters (De Poli and Rochesso, 2002). From this point of view, the synthesis tools constitute experimental domains in which compositional processes must bring into being particular sounds (Eckel, 1993) by setting concrete values to each one of these parameters.

Computer-aided composition, on the other hand, places the compositional process in the centre of the modelling activity (Malt, 2003), leading to the introduction of the term "compositional models". In this new perspective, the modelling process is what actually abstracts music from its raw acoustic manifestation to the realm of composition, under the form of formal structures and processes.

In a computer-aided composition system, the compositional models are therefore represented with programs including the internal representations of their formal components (compositional data), but also with their structural relations which constitute the core of the compositional process. These programs are created, corrected, stimulated by the composer in order to obtain musical results. While being constructed the components, as well as the structure of the model, become concrete along with the definition of the object (Berthier, 2002). The creative musical sense of the composition model then arises from these different representations of the components and structural aspects of the model.

[1] Laske (1981) was one of the first articles that suggested the importance of this interaction between the composer and the computer.

4. Sound representation and modelling in computer-aided composition

4.1. General approach and summary of the conceptual framework

In order to address our problem of the musical representation of sound, we will now follow our approach of computer-aided composition (considered as a framework for the development of compositional models) and insert sound in this conceptual context.

First we must consider sound as the intentional object of a compositional process, and therefore as a structure likely to be represented by a program. Through the general structuring of the programs via the abstractions or hierarchical constructs, compositional models are created, made of organized component and structural aspects and corresponding to particular situations or compositional approaches. In this context, sound is therefore not considered only from the acoustic signal production point of view, but rather as the product of a compositional process that aims to create this signal. By determining particular components and structural contents of these models, certain aspects of the programs are put forward of the representation, thus establishing implicit symbolic abstraction levels within this representation.

This representation of sound, especially using visual programming interfaces, might therefore allow for the symbolic access and control of the corresponding compositional models, whether they concern symbolic or subsymbolic structures. The synthesis processes are then integrated within a network of structural relations and components, uniting the low-level sound synthesis aspects and the symbolic layers of composition, and providing *both a sound producing entity and a logical object musically meaningful* (Eckel and Gonzalez-Arroyo, 1994), that is, a representation allowing to formulate compositional intentions. The musical sense is made explicit by recovering (mentally and/or physically) the musical object of the model as a formal structure (for the composer-at-work) or eventually as a sound according to a sound production process.

4.2. Related works

To meet this objective CAC systems must be adapted with new representations for the compositional model components, and structurally extended down towards sound synthesis processes with an additional part, in charge of interpreting these components in order to recover sound signals.

Various works were carried out to further this purpose. Some sound synthesis languages provide high-level programming features which make it possible to embed sound synthesis processes in algorithmic composition processes. One of the most representative of this type of environment is SuperCollider (McCartney, 1996). Dannenberg (1997), Rodet and Cointe (1984), Eckel and Gonzalez-Arroyo (1994), Hanappe (1999) represent other attempts to work with sound synthesis using programming languages. As for visual programming environments, the practice up to now in PatchWork and OpenMusic was to format synthesis parameters computed by compositional programs and dedicated to specific sound synthesis softwares (most frequently, scores for the Csound synthesis language). PWGL proposes more advanced features since it is enriched with a real-time sound synthesis framework (PWGLSynth, see Laurson *et al.*, 2005). With PWGLSynth, direct interaction is possible between the score or other high-level musical structures calculated in visual programs, and sound synthesis processes designed in the same environment using a distinct but integrated engine. An implicit "score level" thus separates these two domains; however, the integration of real-time synthesis with ("non-real-time") compositional processes allows powerful possibilities.

As we will demonstrate, our approach in OpenMusic deliberately favours the compositional part, attempting to make it get as closer as possible to the final sound level, and delegate the sound rendering to external sound synthesis tools. The role of the CAC environment is therefore not only to link compositional parameters to sound synthesis processes but to make musical thought correspond with the sound domain in order to provide sound design with a musical dimension.

5. Visual programming framework

OM-Sounds is a visual programming project that completes the computer-aided composition environment OpenMusic inspired by the theoretical framework described above. It proposes new tools

and features for the creation of composition models using sound representations and synthesis processes (Bresson, 2006).

Figure 2 shows an example of a complete synthesis process developed with the *OM-Sounds* tools that will be presented in this section.

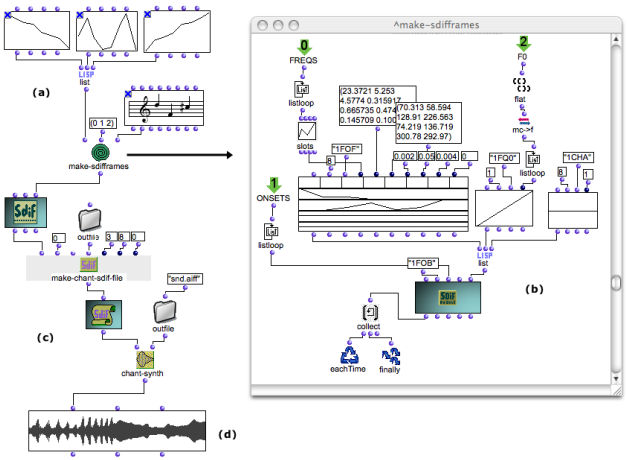

Figure 2. A sound synthesis process developed in OpenMusic using the OM-Sounds framework. Symbolic data structures (a) are converted into synthesis parameters under the form of SDIF structures (b) in a sub-process (abstraction) and then transferred to a synthesis system (c) in order to compute a sound (d).

5.1. Sound processing

Contrary to the major part of the aforementioned CAC systems oriented toward sound synthesis, sound synthesis is not performed within OpenMusic. A deliberate choice has been made to use external and pre-existing tools and standards so as to favour an openness of the system, interoperability, and to better concentrate on compositional issues. As we shall demonstrate in this section, these external tools, although they correspond to radically different paradigms and technologies, should be coherently integrated in the compositional framework. Sound synthesis processes can take on a large variety of forms, compositions, and complexity, and can inspire the corresponding possible (and subjective) representations. They are considered as an abstract variable element in our compositional models.

Different (possibly complementary) options are possible in order to compute the digital sounds from external sound processing tools. *OM-Sounds* can communicate with these tools by sending system commands and writing/reading external files (see Figure 2-*c*). Such interfaces were designed for synthesizers like Csound (Boulanger, 2000), SuperVP (Depalle and Poirot, 1991), or CHANT (Rodet *et al.*, 1984). Various predefined processes are available through specialized functions that convert their input data to adequately formatted parameter files. Computer languages for programming DSP processes—e.g. the Music *N* languages (Mathews, 1969) like Csound, but also functional languages (Orlarey *et al.*, 2004), or real time DSP visual programming systems (Puckette, 1991; Puckette, 1996)—allow the compositional activity to deal with sound synthesis programs design. As a complete programming language, Csound allows greater possibilities than precompiled synthesizers. An "orchestra" edition tool makes it possible to define the DSP processes in Csound within our visual programming environment. Through the possibility of creating and sending data in the form of events

in the OSC protocol (Wright and Freed, 1997) it is also possible to use, or define external sound synthesis processes in real-time environments such as Max/MSP or PureData and communicate the input data of these processes to them.

5.2. Sound descriptions

We use the generic term of sound description to refer to the data related to sound signals in the compositional processes. Sound descriptions generally correspond to the parameters of a targeted sound synthesis process, or higher-level structures built upon these parameters. They constitute a first abstraction for describing sounds following a particular angle defined by the model. The sound descriptions should be as general as possible so as not to constrain the development of compositional models with predefined assignations. We manage them with general-purpose objects like the envelope (evolution of an abstract parameter value) or the matrix (see Figure 2-*a* and *b*). The matrices represent the joint description of various parameters. Gathering them in a sole structure facilitates the setting of relationships between these parameters and their evolutions. It also clusters them in a single object, which then acquires a stronger symbolic status in compositional processing, since it represents a more consistent reality. Matrices were especially used in the implementation of the *OMChroma* project, now integrated in our environment, which has been described in previous articles (Stroppa *et al.*, 2002; Bresson *et al.*, 2007).

Digital sounds are the final intended representations of the object of our compositional models, and they might come into play in the processes as initial or intermediate material. In OpenMusic, the class *sound* represents an audio file, i.e. a sound under the basic digitalized waveform representation, which is also a particular form of sound description (see Figure 2-*d*). In a patch, this object is principally instantiated using a pathname assigning a file on the disk. The inner data of a sound are thus stored in an external file. They can be processed in the programs using a functional toolkit built on the underlying audio architecture:[2] cascading pointers facilitate combining and embedding various treatments and manipulations of abstract audio resources without handling the samples' data directly.

The cross-conversions between sounds and low or high-level representations of sound allow composers' personal processing and the possible subsequent definition of musical abstractions. The intermediate levels between symbolic entities and low-level data may indeed have a particular importance: in our comparison with instrumental sound creation, they may correspond to the steps that substitutes (and should balance) the contribution of the human performer. For this purpose, complementarily to the usual approach consisting in automatically converting data from composition to sound synthesis systems (on the reverse operation), we propose making it possible to apply the symbolic calculus directly on these data.

The conversion to standard data formats ensures the communication and transfers between composition and synthesis processes. While MIDI became to be the standard in instrumental-like controlled systems, the SDIF format (Sound Description Interchange Format) was created for generalized sound analysis/synthesis data, providing an open standard for the codification, storage and transfer of sound descriptions (Wright *et al.*, 1998). Basically, an SDIF stream is a sequence of timed frames, each containing one or various types of matrices of parameters. Considering this format as a generic support for sound descriptions allows us to provide generic tools for the low-level processing of sound description data (Bresson and Agon, 2004). The standard unifies all types of sound representations with the same format, and therefore allows for their manipulation by using the same programming toolkit. At the same time, it guarantees the compatibility of these data in possible communications and interchanges with external applications. Like *sound*, the *SDIFfile* class points to an external file containing sound description data. SDIF-formatted data can be created and processed thanks to the internal SDIF structures (frames, matrices, type declarations, etc.) available among the visual programming tools, and read/written from/to SDIF files (see Figure 2-*b*). Finally a 3D editor (Bresson and Agon, 2004) is used to visualize these files, and control the data transmitted across the system.

[2] LibAudioStream audio library, by GRAME (`http://libaudiostream.sourceforge.net`)

5.3. Compositional issues

As previously explained, the sound descriptions and sound synthesis processes are generally close to the subsymbolic domain, and must generally represent complex time-varying values precisely. In consequence, they can hardly be taken up entirely and extensively in a compositional thought. The tools of the visual programming framework, principally the abstraction and iteration features, might make it possible to manage this complexity by structuring the processing aspects of the models. Additional features also allow the handling of the complexity of the data using computational or behavioural features, or by using data derived from existent sounds.

When high precision is required in the sound description data, the values can be specified either by extension in a straightforward (but generally tedious) way, or by intension using either algorithmic or mathematical means. Programming and calculus allow for the generation complex data, or for the transformation and computation of these data from initial minimal ones (for example using sampling, linear, or polynomial interpolations—see Figure 3-*a*). An envelope, for example, can often be represented with few points (a break-point function) at the compositional level, since the user does not generally need to control the full values of a parameter evolution, but essentially its profile (e.g. for a dynamic envelope). Such functions can be interpreted and converted later in computing processes in order to recover more precise descriptions. The *matrix* class is also provided with particular features, and can be instantiated either by extension or by symbolic means, interpreted in the matrix processing. The number of elements is fixed, and the different fields are filled in appropriately according to the type of input data: constant values, lists repeated until completion of the matrix, envelopes sampled accordingly to the size of the matrix, or even user-defined functions evaluated, also accordingly to the elements of the matrix.

User-defined procedures can also be assigned to the matrix in order to specify how the elements should be processed in order to produce the final data (Agon *et al.*, 2000). Subclasses of *matrix* might even enclose the underlying synthesis process specification. A behavioural aspect can thus be associated to the sound descriptions, or is sometimes implicitly contained in the classes themselves. The symbolic means provided by the visual programming tools therefore allow the formulation of generative rules in order to create complex descriptions. This particular aspect was also discussed in (Bresson *et al.*, 2007).

Another widespread method to obtain complex parameters values for sound synthesis processes is to use signals from the real, physical world. These signals ensure the richness and diversity of the corresponding material, subsequently handled by programs in compositional processes. This is one of the main advantages of real-time systems, where gestural control, sliders, or other motion capture devices enable easy experimentation and direct integration of natural data within the synthesis processes. In contrast, the computer-aided composition approach favours an overall view over the musical structures and considers data from sound signal analysis as achieved entry-level material.

OM-Sounds provides a specialized toolkit for extracting data from sounds by analysis processes. These processes are carried out by external tools for which specialized interfacing functions were developed. The SuperVP program is used here again to process STFT analysis on sound files (and return the spectral analysis data), fundamental frequency estimations (Doval and Rodet, 1991), or transient detection analysis (Röbel, 2003). Pm2 is another additive tool from IRCAM used to process partial tracking or "chord sequence" analysis. The results of these analysis processes are stored in SDIF files. The data in these files can be accessed in the visual programs via a set of functions for extracting localized or selected data, likely to be later processed in downstream visual programs or converted into higher-level data structures (see Figure 3-*c*).

Figure 3 illustrates the process previously shown in Figure 2 with supplementary components for sound description data generation corresponding to the methods mentioned in this section.

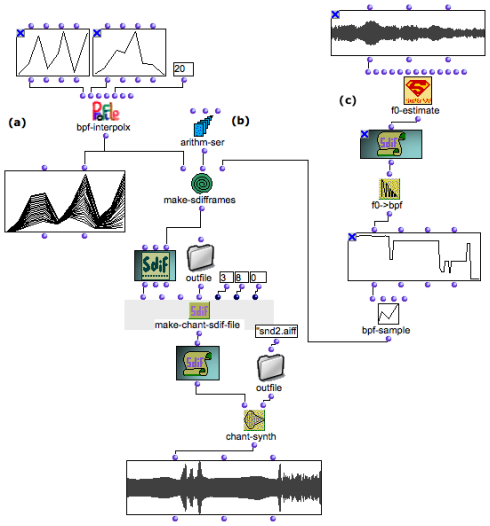

Figure 3. Generation of complex data for sound synthesis processes. In this patch, derived from the patch in Figure 2, the input data of the parameter setting function ("make-sdifframes") are generated by an interpolation process producing multiple envelopes (a), by a simple arithmetic series producing regular time frames (b), and by the fundamental frequency analysis of an audio file and symbolic processing of the analysis data (c).

The specificities of the environment described up to this point can thus be summed up with the following few points. First, it is possible to manage sound synthesis processes and sound representation using structured and visual processes. The specificities of CAC systems (namely the possibility of formatting a model following abstract compositional ideas, of experimenting with this model and refining it until the targeted results are reached) are therefore applied to the design and creation of sounds. Graphical editors and representations allow a trace of the processes to be kept and to efficiently apprehend, store, and experiment with the sound processing tools and operations. The programming features then make it possible to design complex algorithms in order to define the parameters of these processes as well as the possibility to carry on iterative processes in order to manipulate large data bases or sound material banks. As this toolkit is integrated in a general-purpose CAC environment, it is also possible to link the sound synthesis process to traditional musical structures and tools (such as score editors or symbolic data manipulation features). Finally, all these possibilities are combined in one functional execution and under a common calculation and representation paradigm.

6. Time and structure

6.1. Visual programs and temporal structure

By using the *OM-Sounds* framework along with the general visual programming tools of the OpenMusic environment, composers are thus provided with means to develop complete processes in relation with sound processing issues. In this context, the functional program structures let them manage the complexity by maintaining hierarchical control from musically relevant abstractions down to DSP processes.

However so far we have omitted a fundamental aspect of composition models: the relation of musical material and processes with time structures. Music composition is indeed essentially a question of organizing time structures; and sound synthesis also requires a particular attention to the temporal evolution of the parameters. The control of sound synthesis processes thus obviously involves control over time aspects including the temporal behaviours of the individual components as well as the organisation, sequencing, and articulations of a global musical form.

In the environment we have described up to now, time is only considered as one of many musical parameters in the programs. The musical objects could be compared to the *out of time* structures from (Xenakis, 1992), i.e. structures containing their internal composition rules, but that have not yet been unfolded *in time*. They have an internal temporal dimension but a musical time structure remains to be defined in order to develop musical forms.

For this purpose, we use the *maquette* interface of OpenMusic, which is able to put visual programs in a temporal context, thus improving the modelling framework with time structures. The *maquette* is an extension of the notion of visual program with additional spatial and temporal dimensions, allowing to put the elements of the composition framework (data structures and processes) are in close relation to these new dimensions.

In a maquette editor, the boxes (called *temporal boxes*) represent functional units (programs) producing musical outputs (see Figure 4). These boxes can either refer to a standalone object factory, or to a program (patch). The position and graphical properties of these boxes are associated with a temporal and structural sense; particularly, the horizontal axis of the editor represents time, so that the position and horizontal extension can be related to offsets and durations. The graphical and temporal characteristics (size, position, etc.) of the temporal boxes can be edited manually in the maquette editor and are also accessible within the boxes' built-in patches. The calculus of musical outputs could then possibly depend on these external properties.

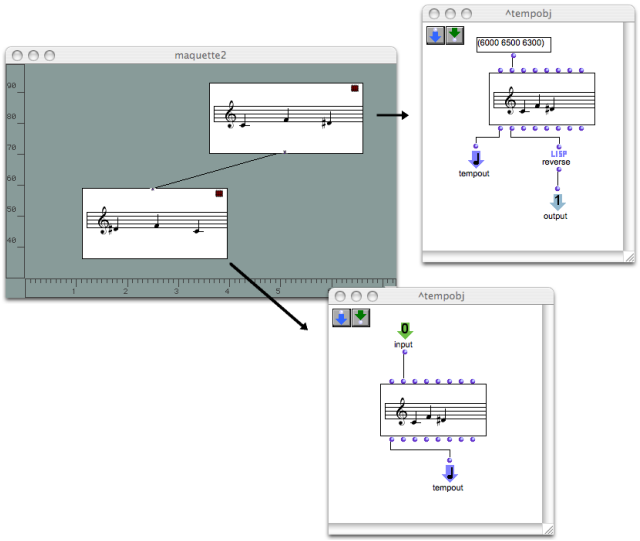

Figure 4. Example of a maquette in OpenMusic. Each temporal box encloses a patch that produces a musical output. Some of them can use data coming from other boxes using functional connections. In this example the pitches from a chord sequence are reversed and used in the second one.

The temporal boxes can also be linked by functional connections so that the whole maquette may finally be considered as a program, holding functional and temporal semantics. Temporal relations and constraints (synchronization, relative offsets, etc.) can therefore be set between the boxes by setting the temporal parameters in their corresponding patches. Hence, the calculus can determine the time structure. Processes are unfolded in time, but they can also process temporal information making the computing time of the program, the musical time of the data, and the general time structure interact (Assayag et al., 1997b). This point may be related to the need formulated in (Boulez, 1987) to let *external compositional criteria* (organisation of the musical objects) *act on internal criteria* (construction of the musical objects) *and modify the objects in order to link them in a coherent development and place them in a formal context.* In addition, the possibility of embedding another maquette in a temporal box allows for the construction hierarchical temporal structures.

Upon execution of the program, each box is computed individually or following the possible functional connections. Provided their outputs have a musical meaning, they can be gathered in an inclusive object integrating them, given their temporal organisation. This object can be related to the object of a composition model, that results from a particular configuration of the components (temporal boxes) and the structure of the maquette. It can also be played and heard as a musical object.

6.2. Temporal issues and sound synthesis processes

One of the first questions when trying to define temporal structures for sound synthesis is that of the nature of their components. As explained in (Honing, 1993) *the notion of structuring depends on the possibility of decomposing a representation into meaningful entities*, which the author calls *primitives*. With sound synthesis, these primitives are not necessarily self-meaningful musical entities (like notes, as in Figure 4, or sounds) but can be any particular aspects of a sound representation. The definition of these primitives will, moreover, condition the representation since it will determine the part of the total information hidden in terms of explicit control (in the time structure), and that which will pass up to the foreground of the organisation.

Most of the sound synthesis parameters require a description of their continuous evolution in time. Here *continuous* denotes an evolution in which the user/composer cannot distinguish independent (discrete) meaningful entities. This can be compared to the previously discussed subsymbolic/symbolic division; and here also, the discrete aspect of the structures, perhaps better expressed as discontinuity, provides landmarks for transitions and separations, and at the end is what *gives rise to composition in its many dimensions* (Boulez, 1987).

For the construction of a sound representation unfolded in time, we are thus led to define symbolic event-like objects, as abstract sound description data computed by programs and that take part in the temporal structure, and then to consider their (possibly continuous) expansions and interpretation in another lower structural level.

A new element is attached to the maquette for this purpose, in the form of an auxiliary patch. This patch defines a process responsible for the interpretation of the components that make up the time structure. Typically, it will make use of sound processing and low-level features in order to compute a sound with the information provided by the primitive elements (events) temporally organized in the maquette (Bresson and Agon, 2006).

Figure 5 shows a simple example of a maquette including the aforementioned aspects. In this maquette each box is an abstraction defined by the user, i.e. a program having some possible inputs but, above all, an output that produce some part of a sound description (in this case, a matrix of parameters), which constitutes an event in the high-level temporal structure. The execution of the program represented in the maquette is therefore now carried out in two successive steps. After the events were computed and temporally organized, the second step of the program execution processes them in order to expose their complete contents and/or to interpret them in order to recover a sound. The access via programming to this process (represented by a patch visible at the bottom left-hand part of the maquette editor) therefore permits specific and subjective interpretations of the data. Continuous phenomena and evolutions can thus be handled in the inner program contents of the events, or in this final synthesis process (particularly when transitions between different events are to be specified).

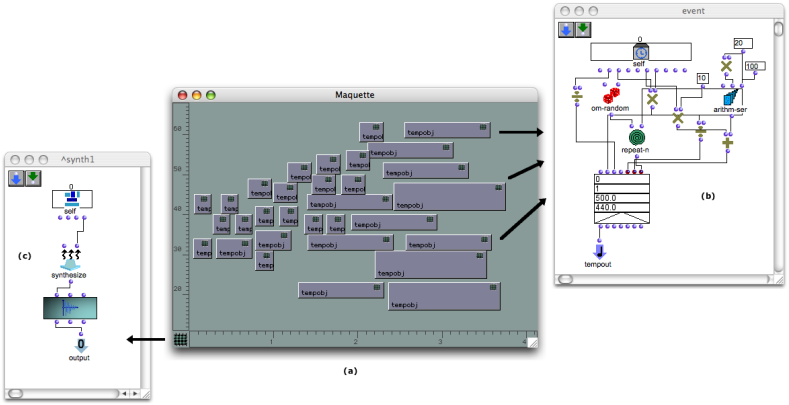

Figure 5. Extended maquette for the representation of sound synthesis processes. The boxes in the maquette temporal structure (a) refer to the various instances of patches producing sound description data (b) and interpreted by a general synthesis process (c). This process is represented in the bottom left-hand corner of the maquette editor, and is visible on the left-hand side of the figure.

The "discretization" strategy (i.e. the definition of the primitives of the temporal structure) therefore depends on the user of the system: the events can correspond to any kind of data, and not necessarily to musically independent structures, i.e. structures carrying their full musical meaning (this meaning being recovered later on by the interpretation process). In addition, such a maquette can concern a microscopic structure (to be integrated in a higher-level musical form) as well as a macro-structure defining the form of a whole musical piece.

There are then three conceptual levels on which the composer can interact in the global composition/synthesis process. At a "subsymbolic" level he can work on the definition of events as abstract programs or data structures (Figure 5-*b*). At a "symbolic" level, he can manipulate the maquette components in order to move, resize, duplicate, and organize the different types of previously defined events (Figure 5-*a*). Finally the interpretation process, back to the subsymbolic level, makes it possible to work and experiment on the semantic of the overall structure, i.e. on the process by which it will be converted into a sound (Figure 5-*c*).

The maquette can therefore constitute a general representation of a sound in the compositional modelling framework. In this representation, the high-level compositional processes are developed along with the DSP processing of the data, which is an independent and flexible part of the model. At the same time, it plays the role of the score, where symbolic objects (events) are generated by programs, and organized following a structure defined by temporal or functional relations. The dimensions of the sound representation space are reduced in the high-level interface (the maquette editor), which constitutes a personalised representation of a musical form. This external visible part puts forward a subjectively relevant aspect of the global process, principally by the choice of the internal parameters related to the external box characteristics. On this external part, direct hand manipulations can be applied, enabling the experimentation with forms and temporal organizations. While the external representation is simplified in the high-level interface, the access to the low level through calculus is not restricted and still allows for a complete control of the deployment of the descriptions at the moment of their transformation to a sound signal.

6.3. Discussion: further works on time structures

Composition models facilitate the creation and development of some partial aspects of a musical work. Various models may then possibly be considered when put together in higher-level musical forms, that is, in higher-level models including sub-models as elemental components. The temporal dimension and musically relevant result of the maquette (the object of the model) indeed make it possible to imagine higher-level temporal contexts, or more generally higher-level composition models, in which a representation such as that carried out in the maquette might be considered only as a part of a whole compositional process. Moreover, we have previously mentioned that a sound could possibly be the result as well as the entry-level material of a compositional process, which theoretically supports this notion of self-nested structures in the composition framework.

Maquettes (as well as simple patches) can embed other sub-maquettes in hierarchical temporal structures. In this case, the various sub-models must also be able to interact with one another in order to constitute a logical structure. The hierarchy is thus extended to the general structural domain by the possibility of setting up abstractions in the maquettes (i.e. to make some of their parameters variable, as would be done with classical programs – patches) and to define functional relations. However, this hierarchical view poses some interesting problems that will need to be solved in further works on the system's temporal structures. Particularly, the role of the synthesis process added to the maquette (which would not necessarily produce sound but could be a component in a higher-level structure) must be re-envisaged, and raises the question of the real consideration of (temporal) programs as first-class objects in the visual programming language: a maquette must be seen as a higher-order function, i.e., a function dealing dynamically with other functions to produce a musical object (sound), but also a first-class object able to be integrated in other higher-order functions. Moreover, this "functional programming" view must be considered in parallel with the temporal structures which organization must be preserved and accessible through the different hierarchical levels. The possible relations between elements that do not share a direct hierarchical relation might also be problematic.

Although functional and temporal relations can also be implemented between the temporal boxes in a maquette (through the corresponding patches) some tools for a direct integration of logical relations, not subjected to the calculus of the maquette and of the included boxes, might also improve considerably the musical efficiency of this tool. This is the focus of current works by Allombert *et al.* (2007) on the integration of real-time logical relations in the maquette.

Finally the combination of heterogeneous temporal systems such as that of rhythmic musical notation is also the object of developments in a new score editor to be integrated in OpenMusic. This editor should make it possible to manage rhythms, linear time systems (such as that of the maquette), as well as other (discrete or continuous) musical object possibly involved and interrelated in musical and sound synthesis processes (Agon and Bresson, 2007).

7. Conclusion

Digital sound synthesis techniques provide powerful sound representations likely to be manipulated by calculus. Meanwhile, computer-aided composition considers the formal models of musical ideas and provides representations for their development in a compositional context. The computer constitutes the environment where these two domains take place: it makes it possible to work on the formalisation of musical structures and on the creation of synthetic sounds, that is, at all the different levels of music creation. The issue we tried to address in this paper therefore goes beyond this elementary coexistence and moves towards environments able to integrate the concepts, data, and processes from these two domains in a complete and coherent compositional approach.

We first underlined the relevance of programs as a possible representation for music, revealing its compositional aspects, and that of the programming languages as corresponding compositional tools. The concept of computer modelling applied to sound processing and to compositional fields then allowed us to devise a musical representation of sound, determined by the structure of a compositional model. Sound considered within this conceptual basis is no longer represented only by some data intended to be sent to a synthesis program, but by a program corresponding to a compositional process likely to generate these data and the subsequent sound result. As the object of a compositional model,

a sound does not actually have any predefined representation, this representation being defined along with the creation of the model.

On these theoretical bases we have presented the main development lines of a computer-aided composition framework oriented towards sound synthesis, created within the OpenMusic visual programming environment. In this framework, the main advantages we have emphasised are the modularity allowed by abstractions and structuring of the processes, the relations between structures, time, and calculus, as well as the possibility of dealing with continuous and subsymbolic issues in a symbolic environment. The complex objects involved in the setting of sound synthesis parameters are handled concretely or by intension starting from symbolic initial data structures associated with programs or behavioural rules, and connections to the sound domain are provided through communication with sound synthesis and analysis programs. Furthermore, the general-purpose computer-aided composition environment ensures that the sound synthesis issues can be directly connected to the macro-compositional processes or structures.

The temporal outlook and structuring of the processes allow the free determination of a symbolic musical level and further manipulations on it, as well as on the resulting subsymbolic levels of the process. It includes the different elements of a musical work—from sound conceptions to the global piece forms—and enables a constant interaction with the materials and processes at every step and structural level.

This framework therefore constitutes a complete system for modelling compositional processes and creating musical sound representations. It makes it possible to compose and manipulate sounds within a symbolic framework defined by composers. In *The OM Composer's Book 2* (2008) composers like Tolga Tüzün or Hans Tutschku give some good examples of pieces composed using parts of the visual programming and temporal representation tools presented in this paper. We believe that future improvements may also provide other innovative potentials in the field of sound composition and contemporary music creation.

Acknowledgements

We would like to express thanks to Gérard Assayag and Olivier Warusfel for their helpful comments and suggestions on the initial drafts of this paper, and to Déborah Lopatin and Catherine Durand for proofreading the manuscript.

References

Agon, C. (1998) *OpenMusic, un langage visuel pour la composition musicale assistée par ordinateur*. PhD thesis. Université Pierre et Marie Curie (Paris 6), Paris, France.

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

Agon, C. and Assayag, G. (2002) Programmation visuelle et éditeurs musicaux pour la composition assistée par ordinateur, *in 14ème Conférence Francophone sur l'Interaction Homme-Machine – IHM'02*, Poitiers, France.

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

Agon, C., Assayag, G. and Bresson, J. (Eds.) (2006) *The OM Composer's Book .1*, Ircam – Editions Delatour France.

Agon, C. and Bresson, J. (2007) Mixing Time Representations in a Programmable Score Editor, *in Proceedings of Sound and Music Computing Conference SMC'07* , Lefkada, Greece.

Allen, J. B. and Rabiner, L. R. (1977) A unified approach to short-time Fourier analysis and synthesis, *in Proceedings of the IEEE*, 65(11).

Allombert, A., Assayag, G. and Desainte-Catherine, M. (2007) A system of interactive scores based on Petri nets, *in Proceedings of the Sound and Music Computing Conference SMC'07* , Lefkada, Greece.

Arfib, D. (1991) Analysis, Transformation, and Resynthesis of Musical Sounds with the Help of a Time-Frequency Representation, *in* De Poli, G., Piccialli, A. and Roads, C. (Eds.) *Representations of Musical Signals*, MIT Press.

Assayag, G. (1995) Visual Programming in Music, *in Proceedings of the International Computer Music Conference*, Banff, Canada.

Assayag, G. (1998) Computer Assisted composition today, *First Symposium on Music and Computers*, Corfu, Greece.

Assayag, G., Agon, C., Fineberg, J. and Hanappe, P. (1997a) An Object Oriented Visual Environment for Musical Composition, *in Proceedings of the International Computer Music Conference*, Thessaloniki, Greece.

Assayag, G., Agon, C., Fineberg, J. and Hanappe, P. (1997b) Problèmes de notation dans la composition assistée par ordinateur, *Actes des rencontres musicales pluridisciplinaires, Musique et Notation*, Lyon (GRAME), France, 1997.

Assayag, G., Rueda, C., Laurson, M., Agon, C. and Delerue, O. (1999) Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic, *Computer Music Journal*, 23(3).

Barendregt, C. (1984) The Lambda-Calculus. Volume 103. Elsevier Science Publishing Company.

Berthier, D. (2002) *Le savoir et l'ordinateur*, Editions L'Harmattan.

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

Boulez, P. (1987) Timbre and composition – timbre and language, *Contemporary Music Review*, 2(1).

Bresson, J. (2006) Sound Processing in OpenMsuic, *in Proceedings of the 9th International Conference on Digital Audio Effects DAFx-06*, Montreal, Canada.

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

Bresson, J., Agon, C. and Assayag, G. (2005) OpenMusic 5: A Cross-Platform release of the Computer-Assisted Composition Environment, *in Proceedings of the 10th Brazilian Symposium on Computer Music*, Belo Horizonte, Brasil.

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

Bresson, J., Stroppa, M. and Agon, C. (2007a) Generation and Representation of Data and Events for the Control of Sound Synthesis, *in Proceedings of Sound and Music Computing Conference SMC'07* , Lefkada, Greece.

Bresson, J., Agon, C. and Assayag, G. (Eds.) (2008) *The OM Composer's Book .2*, Ircam – Editions Delatour France.

Camurri, A. (1990) On the Role of Artificial Intelligence in Music Research. *Inerface*, 19(2-3).

Chazal, G. (1995) *Le miroir automate*. Ed. Champ Vallon.

Chowning, J. M. (1973) The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Engineering Society*, 21(7).

Dannenberg, R. B. (1989) The Canon Score Language, *Computer Music Journal*, 13(1).

Dannenberg, R. B. (1997) Machine Tongues XIX: Nyquist, a Language for Composition and Sound Synthesis. *Computer Music Journal*, 21(3).

Dannenberg, R. B., Desain, P. and Honing, H. (1997) Programming Language Design for Music, *in* Roads, C., Pope, S. T., Piccialli, A. and De Poli, G. (Eds.) *Musical Signal Processing*, Swets and Zeitlinger.

Depalle, Ph. and Poirot, G. (1991) A modular system for analysis, processing and synthesis of sound signals, *in Proceedings of the nternational Computer Music Conference*, San Francisco, USA.

De Poli, G., Piccialli, A. and Roads, C. (Eds.) (1991). *Representations of Musical Signals*, MIT Press.

De Poli, G. and Rochesso, D. (2002). Computational Models for Musical Sound Sources, *in* Assayag, G., Feichtinger, H. G. and Rodrigues J. F. (Eds.) *Mathematics and Music – A Diderot Mathematical* Forum, Springer.

Doval, B. and Rodet, X. (1991) Estimation of Fundamental Frequency of Musical Sound Signals, *in Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing – ICASSP 91*, Toronto, Canada.

Eckel, G. (1992) Manipulation of sound signals based on graphical representation. A musical point of view, *in Proceedings of the 1992 International Workshop on Models and Representations of Musical Signals*, Capri, Italia.

Eckel, G. (1993) La maîtrise de la synthèse sonore, *in Les cahiers de l'Ircam 2 – La synthèse sonore*, Ircam – Centre Pompidou.

Eckel, G. and Gonzalez-Arroyo, R. (1994) Musically Salient Control Abstractions for Sound Synthesis, *in Proceedings of the International Computer Music Conference*, Aarhus, Denmark.

Florens, J. and Cadoz, C. (1991) The physical model : modeling and simulating the instrumental universe, *in* De Poli, G., Piccialli, A. and Roads, C. (Eds.) *Representations of Musical Signals*, MIT Press.

Girard, J.-Y., Taylor, P. and Lafont, Y. (1989) *Proofs and Types*, Cambridge University Press.

Hanappe, P. (1999) *Design and implementation of an integrated environment for music composition and synthesis*, PhD Thesis, Université Pierre et Marie Curie, Paris VI.

Hiller, L. (1969) Revised MUSICOMP Manual. Technical Report 13, University of Illinois Experimental Music Studio.

Honing, H. (1993) Issues in the representation of time and structure in music, *Contemporary Music Review*, 9.

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

Laurson, M. (1996) PWConstraints, *Symposium: Composition, Modélisation et Ordinateur*, Ircam, Paris, 1996.

Laurson, M. and Kuuskankare, M. (2002) PWGL, a Novel Visual Language Based on Common Lisp, CLOS, and OpenGL, *in Proceedings of the International Computer Music Conference*, Gothenburg, Sweden.

Laurson, M., Norilo, V. and Kuuskankare, M. (2002) PWGLSynth: A Visual Synthesis Language for Virtual Instrument Design and Control, *Computer Music Journal*, 29(3).

Laske, O. (1981) Composition Theory in Koenig's Project One and Project Two, *Computer Music Journal*, 5(4).

Leman, M. (1993) Symbolic and subsymbolic description of music, *in* Haus, G. (Ed.) *Music Processing*, Oxford University Press.

Malt, M. (2003) Concepts et modèles, de l'imaginaire à l'écriture dans la composition assistée par ordinateur, *in Séminaire postdoctoral Musique, Instruments,* Machines, Paris, France.

Mathews, M. V. (Ed.) (1969) *The Technology of Computer Music*, MIT Press.

McAulay, R. J. and Quatieri, T. F. (1986) Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4).

McCartney, J. (1996), Rethinking the Computer Music Language: SuperCollider. *Computer Music Journal*, 26(4).

Moorer, J. A. (1977) Signal processing aspects of computer music : a survey, *Proceedings of the IEEE*, 65(8).

Orlarey, Y., Fober, D. and Letz, S. (2004) Syntactical and Semantical Aspects of Faust, *Soft Computing*, 8(9).

Pope, S. T. (Ed.) (1991) The Well Tempered Object (Musical Applications of Object-Oriented Software Technology), MIT Press.

Puckette, M. (1991) Combining Event and Signal Processing in the MAX Graphical Programming Environment, *Computer Music Journal*, 15(3).

Puckette, M. (1996) Pure Data : another integrated computer music environment, *in Proceedings of the Second Intercollege Computer Music Concerts*, Tachikawa, Japan.

Risset, J.-C. (1991) Timbre Analysis by Synthesis, *in* De Poli, G., Piccialli, A. and Roads, C. (Eds.) *Representations of Musical Signals*, MIT Press.

Roads, C. (1996) *The Computer Music Tutorial*, MIT Press.

Roads, C. (2002) *Microsound*, MIT Press.

Röbel, A. (2003) Transient detection and preservation in the phase vocoder, *in Proceedings of the International Computer Music Conference*, Singapore.

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

Rodet, X., Potard, Y. and Barrière, J.-B. (1984) The CHANT project: From the synthesis of the singing voice to synthesis in general, *Computer Music Journal*, 8(3).

Rueda, C. and Bonnet, A. (1998) Un langage visuel basé sur les contraintes pour la composition musicale, *in* Chemiller, M. and Pachet, F. (Eds.) *Recherches et applications en informatique musicale*, Hermes.

Rueda, C., Alvarez, G., Quesada, L. O., Tamura, G., Valencia, F. D., Diaz, J. F. and Assayag, G. (1991) Integrating Constraints and Concurrent Objects in Musical Applications: A Calculus and its Visual Language, *Constraints*, 6(1).

Schaeffer, P. (1977) *Traité des objets musicaux*. Editions du Seuil.

Steele, G. L. (1998) *Common LISP The language*, Second Edition, Digital Press, USA.

Stockhausen, K. (1988a) Situation de l'artisanat (critères de la musique ponctuelle), *Contrechamps*, 9, Editions L'Age d'Homme, Paris, pp. 10-15.

Stockhausen, K. (1988b) Musique électronique et musique instrumentale, *Contrechamps*, 9, Editions L'Age d'Homme, Paris, pp. 66-77.

Stroppa, M., Lemouton, S. and Agon, C. (2000) OMChroma ; vers uneformalisation compositionnelle des processus de synthèse sonore, *Actes des Journées d'Informatique Musicale*, Marseille, France.

Taube, H. (1991) Common Music : A Music Composition Language in Common Lisp and CLOS, *Computer Music Journal*, 15(2).

Truchet, C., Assayag, G. and Codognet, Ph. (2003) OMClouds, a heuristic solver for musical constraints, *in MIC'03 Metaheuristics International Conference*, Kyoto, Japan.

Wright, M. and Freed, A. (1997) Open SoundControl : A New Protocol for Communicating with Sound Synthesizers, *in Proceedings of the International Computer Music Conference*, Thessaloniki, Greece.

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, *in Proceedings of the International Computer Music Conference*, Ann Arbor, USA.

Xenakis, I. (1992) *Formalized Music: Thought and Mathematics in Composition*, Indiana University Press.