Generation and Representation of Data and Events for the Control of Sound Synthesis

Jean Bresson*, Marco Stroppa†, Carlos Agon*
*IRCAM - CNRS STMS / Music Representations, Paris, France
†Hochschule für Musik und Darstellende Kunst/Komposition, Stuttgart, Germany

***Abstract*— This article presents a system created in the computer-aided composition environment OpenMusic in order to handle complex compositional data related to sound synthesis processes. This system gathers two complementary strategies: the use of sound analysis data as basic material, and the specification of abstract rules in order to automatically process and extend this basic material.**

I. Introduction

The generation of control structures for sound synthesis is a widespread issue in computer music. Creating sounds with a given software synthesizer requires that composers describe complex sound representations with high precision. An important task thus consists in devising methods that define and generate such descriptions in a musically relevant way.

The parameters of a synthesis process can be specified extensively, a generally tedious and uninteresting chore, or algorithmically. The latter case may also reveal the musical intention more directly. This is principally where computer-aided composition can help to provide a higher level of control with respect to the environments proposed by the synthesizers themselves.

Another method for obtaining complex structures to control sound synthesis is to generate them from signals coming from the physical world, since they already contain the richness and diversity sought for. Spectral analysis is particularly well suited for this purpose, as it provides an accurate description of sound in an intuitive representation (time / frequency / amplitude). Here again, however, the analysis data need to be integrated in and handled by more abstract contexts in order to perform compositional manipulations on them.

The system we present in this paper fuses the two methods mentioned above in a compositional framework that builds and handles generic sound descriptions bound to sound synthesis processes. Embodied in OpenMusic, a visual programming language dedicated to music composition [1] [3], it is widely inspired by Chroma (an environment developed by Marco Stroppa for his compositional activities [12], and ported to Common Lisp/CLOS by its author and Serge Lemouton in the 1990's) and is built on an earlier adaptation of a part of Chroma in OpenMusic, the OMChroma project [13] [2].

Our current work consists in a further step in the integration of Chroma within the visual programming paradigm of OpenMusic. High-level structures are implemented: they are neither true data structures, nor programming tools, but hybrid entities sharing structural and behavioural features of an abstract conception of sounds.

Section II will introduce a structure providing the interface between the realms of symbolic composition and of sound analysis and description. Section III will show the currently available tools to manipulate this structure. Section IV bridges the gap between this high-level framework and the domain of sound synthesis, and finally section V will detail the visual programming tools allowing to define the dynamic modalities of this bridge.

II. CR-Model: a Sound Description Structure

The `cr-model` is a new data structure aiming at representing a sound from the standpoint of sinusoidal modelling. This general representation was designed so as to provide generic tools to process and organise abstract sound descriptions. It is constructed from two data sets: a frequency structure, and a time structure. Figure 1 shows an example of a `cr-model`.

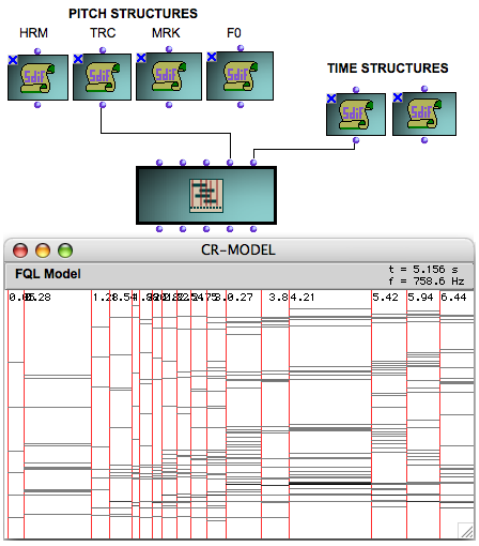

Fig. 1. `cr-model`: a sound representation built from frequency and time structures.

A. Frequency Structure

The frequency information comes from a spectral representation of a sound, that can be obtained from different types of analysis. These can be performed either within OpenMusic [5], or via external systems. The AudioSculpt software [4] provides particularly powerful means to obtain these data.

The analysis data are stored in external files using the SDIF format (Sound Description Interchange Format [14]), represented in OpenMusic via appropriate `SDIFFile` objects.

Four types of analysis are currently supported: harmonic and inharmonic partial tracking [8],"chord sequence", and fundamental frequency estimation [6]. Other types will be implemented in the future.

The `cr-model` data are instantiated by connecting an SDIF file to one of its input slots and by selecting one of these types of analysis (provided that the SDIF file contains the corresponding data type).

In figure 1, the object is created from partial tracking data, which results in a sequence of chords (static frequencies within each time frame, that correspond to average loudest partials). In figure 2, the same object is constructed from a fundamental frequency estimation analysis.

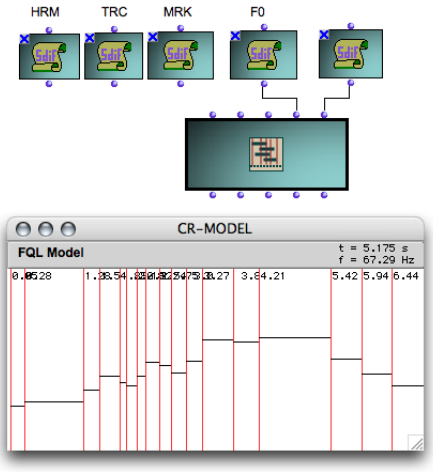

Fig. 2. `cr-model`: alternative frequency structure (from fundamental frequency estimation).

B. Time Structure

The time structure of the `cr-model` is a simple list of temporal markers, derived from either a compositional rule or a "temporal" sound analysis, such as transient detection [10] or the onsets of a chord sequence.

As shown in figures 1 and 2, a `cr-model` can be constructed with different types of spectral analysis and time structures: frequency and time are independent, which is a great musical advantage. One might envisage, for instance, to generate a sound representation combining the frequency contents of a given sound with temporal information of another sound or resulting from a user-defined process.

Figure 3 illustrates the latter case: the frequencies of figure 1 are associated with a regular pulse computed from a simple arithmetic series (0, 0.5, ... , 5.5, 6).

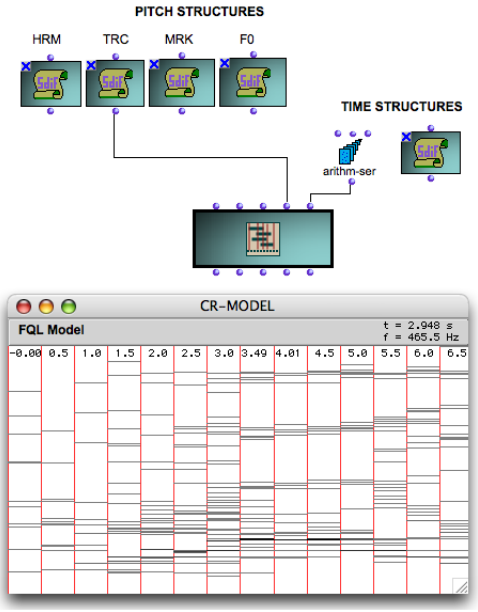

Fig. 3. `cr-model`: alternative time structure.

C. Abstract Representations

A complete `cr-model` thus requires a reference to a frequency and time structure (both structures may be external analysis or sound description files.)

The frequency information is organised in a sequence of data chunks corresponding to the temporal segmentation. The data within each chunk are converted into an abstract format called VPS (*Vertical Pitch Structure* [11]), a polymorphic structure, allowing for the smooth and efficient manipulation of spectral and formant data, as well as "traditional" symbolic chords. The different kind of data are stored in a unique format and dealt with abstractly in downstream compositional processes.

The time structure defines chunks in the sound description, during which the internal data are considered to be relatively stable, or at least likely to belong to the same meaningful segment. What "meaningful" means is a compositional choice, and ought not to be further discussed here. A composer should be "allowed" to segment a sound as she or he feels like. In the end, each chunk will be treated by the system as a primitive event (see section IV). It is certainly not always straightforward to segment a continuous sound into abstract events; however, it is a very useful task, especially when tackling complex or large-scale temporal processes.

At this stage, it is also essential to mention that we do not aim at a faithful reproduction of the analysed sound. Analysis-synthesis environments that are well suited to this task already exist (e.g. [4], [7]). They allow for a fairly reliable resynthesis of the original sound, but offer only very rudimentary processing capabilities. Our purpose is to inverse the priorities: a `cr-model`

will probably not allow for very accurate resyntheses, but will provide the composers with powerful algorithms to process data symbolically, as if they were a mere compositional material. The long-standing experience of Marco Stroppa with this kind of structures has shown that, although the final result may be quite far from the original sound, the latter can be easily recognised, since its cognitive features can be maintained. We also neither attempt to analyse and transform "sound objects" or "scenes" considering their own embedded semantics (for instance, fore- and background elements, as in [9]), nor wish to build any implicit musical knowledge during this phase: it is again the composer's task to give these structures the semantics that she or he imagines. This allows the system to concentrate on the flexibility, as well as on the efficiency and the ergonomics of its components.

III. Manipulations on the Model Data

Starting from a `cr-model`, a compositional process will consist in modelling and transforming it and in yielding personalised structures, more or less distorted and disconnected from the original one.

A. Frequency-Domain Processing

Being an abstract structure, a `cr-model` can be also automatically generated by a program, or constructed from previously-defined VPS, associated to an arbitrary time structure. For instance, the figure 4 shows a simple example of a VPS extracted from a `cr-model`, transformed and used to create a new one.

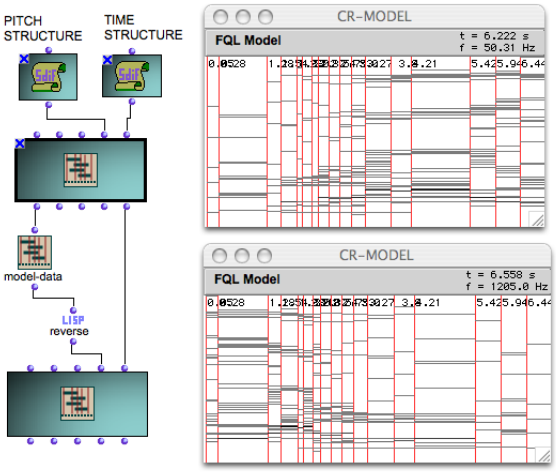

Fig. 4. `cr-model` data processing: the frequency structure from a `cr-model` is reversed and applied to another `cr-model` with the original temporal segmentation.

The `model-data` function reads the data of a `cr-model` as a list of VPS, likely to be processed individually. A second optional input (not shown in figure 4) allows to connect an auxiliary function to be applied within this scope. Some pre-defined functions are available, such as high-pass or low-pass filters (see figure 5), transpositions, frequency stretching, and others. User-defined functions, or combinations of pre-defined functions, can also be created via a sub-patch (a functional abstraction defined in OpenMusic) connected in place of the auxiliary function. Other functions are thus easy to define, depending on the needs of each user. In this way, we preserve the maximum freedom: the composer can rely on built-in functions, create new ones or write patches that locally specify the behaviour of each `model-data` function.

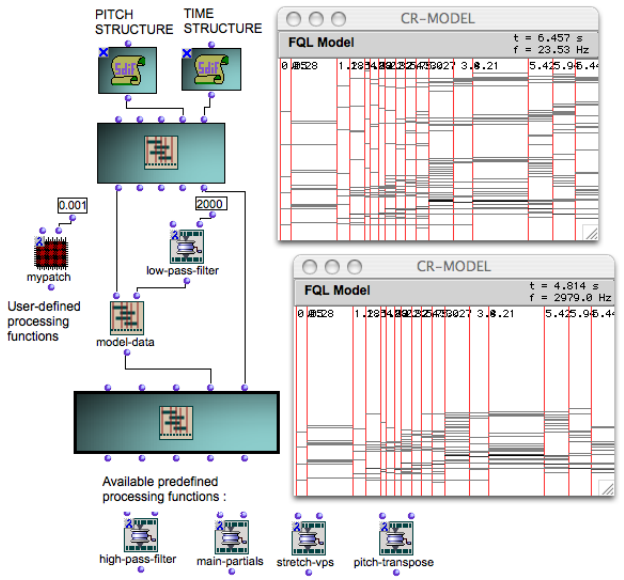

Fig. 5. Frequency processing of a `cr-model`: a symbolic low-pass filter is applied to each VPS of the initial structure. Other available functions are displayed at the bottom of the figure, and can be used (as can the user-defined patch on the left) instead of the one connected to `model-data`.

B. Time-Domain Processing

Time structures can also be subject to modifications and transformations independent of the frequency domain. There are no specific limitations to their complexity. For instance, permutations, compression, stretching, an so on, can be applied on the list of markers, or markers can be quantized into symbolic rhythms, processed, then further converted into absolute time values. Some functions are also pre-defined (time-varying scaling factors, random variations, and the like). Figure 6 illustrates some time-domain operations.

IV. Connection to the Sound Synthesis System

A. Primitive Events

In order to connect a `cr-model` to a software synthesizer, we have to pass through another level of control and convert the model into a set of primitive events.

This notion, which translates the abstract realm of a model into the concrete instructions needed by a given synthesiser is a crucial issue: what does "primitive event" mean for a composer? And how "large", efficient and expressive should it be approximately? In the case of music for acoustic instruments, this is relative clear, albeit

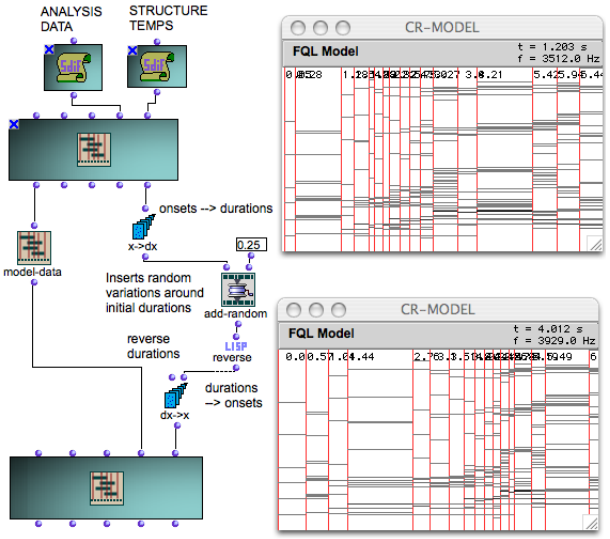

Fig. 6. `cr-model` time processing: the durations of the initial time structure are reversed and subjected to random variations.

subject to discussion: these are the symbols notated in the score, whether traditional or not, and decoded by a performer. In the case of sound synthesis, however, the reality is much more ambiguous: a composer might prefer to generate short events and assemble them together at a later stage (something comparable to, say, each single note on a score for piano), while another, on the contrary, might wish to start with relative large and complex events (comparable to, for instance, a phrase of a violin). Both approaches are important and deserve to be implemented within a system dedicated to the control of sound synthesis.

We started by defining as "event" data structures provided with temporal information; it is their unfolding in time that makes them different from "objects". "Primitive" means that the structure is able to convert its inputs into the appropriate instructions of a synthesizer. After analysing several cases, we decided not to take any aesthetical decision as far as a primitive event is concerned, but to limit ourselves to the construction of a general enough data structure, a matrix, so as to let the composer deal with the task of which meaning is needed. In this way, she or he has the highest degree of freedom and power, and the computer environment will not even try to embark itself upon a task that it cannot carry out correctly.

B. Matrices as Events

We meet here the level presented in [2] as part of the OMChroma system. In this system, a primitive event is handled exclusively with special matrix classes. Each class is linked to a specific implicit synthesis technique and software: it gathers the required information and converts it to the actual parameters required by the synthesizer. OMChroma provides a large amount of predefined classes, but new ones are easy to instantiate either directly in a patch, or, for instance, from a csound orchestra file, or to program in LISP. The public slots of these classes correspond to the compositional parameters of the synthesis process. They are dynamically computed by various methods of evaluation, and can be specified either directly (with lists of values) or symbolically. For instance, in a matrix of a given number of elements (rows), a parameter (line) can be defined with a breakpoint function (which will be sampled according to the number of elements), a list of values (repeated until the correct number of elements is reached), or a mathematical or functional expression, represented with a LISP function or a user-defined program (also evaluated "number of elements" times). Figure 7 shows synthesis processes in OpenMusic using the OMChroma matrices.

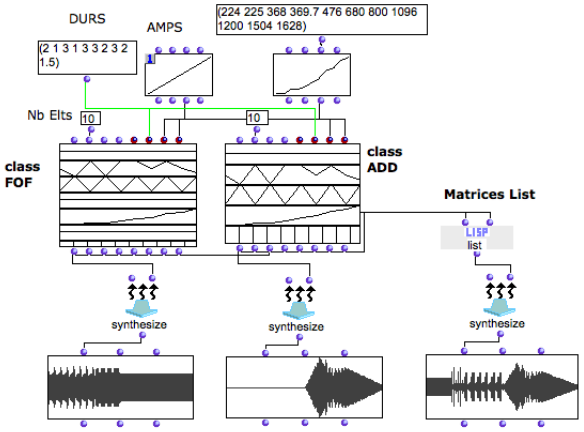

Fig. 7. Synthesis processes using the OMChroma matrices: the matrices' values are filled accordingly to the slots entries, and interpreted by the `synthesize` method. The same values in different matrix classes imply a different underlying processing of these data. On the right, the list merges the two matrices in a same process.

Each matrix has a slot named `action-time`, which acts as an onset for the event. Synthesizing a list of matrices thus can be seen as gathering timed events, as shown in the right-most example in figure 7. Two local slots complete the matrix temporal information: `e-dels`, which represents the event's local time and corresponds to the entry delay of each element with respect to the global onset, and `durs` which corresponds to the duration of each element.

C. User-Defined Functions within the Process

Rule-based synthesis is achieved by assigning an additional function (*user-fun*) to a special input of the matrix. This function specifies how each element is to be processed, modified or extended in order to produce the actual final data to be sent to the synthesizer. It can be defined in LISP or graphically in an OpenMusic patch. Evaluated at the beginning of each computation of a new element of the matrix, this function can access the current state of the whole matrix, change it, perform tests on the data, generate new elements or eliminate some. Figure 8 shows a simple *user-fun* that tests the duration of each element and corrects it when needed.

```
(defmethod ed+dur? ((c component))
"Correct the component if its duration is beyond durtot.
 (e-dels[i]+durs[i] > durtot[e]  durs[i] => durtot[e] - e-dels[i])"

 (let ((curr-dur (comp-field c "durs"))
       (curr-ed (comp-field c "e-dels"))
       (durtot (durtot (event c))))
   (if (<= (+ curr-ed curr-dur) durtot)
     c
     (list
      (comp-field c "durs" (- durtot curr-ed))
      (format () ";---> WARNING / Reduced DUR: old-dur = ~a, new-dur = ~a~% "
                 curr-dur (- durtot curr-ed)))
      )))
```

Fig. 8. LISP user-defined function testing the duration of each element: if it ends beyond a maximum value assigned to the event, the value will be corrected to the maximum allowed and a warning message will be printed in the score sent to the synthesizer.

D. From Abstract Representations to Sound Synthesis

The connection of a `cr-model` to sound synthesis is done by converting it to a list of matrices. The function `expand-model` matches the `cr-model` data to a given class of matrix: this class will determine a targeted synthesis process. At each time interval, the structure is translated into one instance of this class, following a matching rule that will be discussed in the next section. The resulting list of matrices is finally supplied to the `synthesize` method (see figure 9).

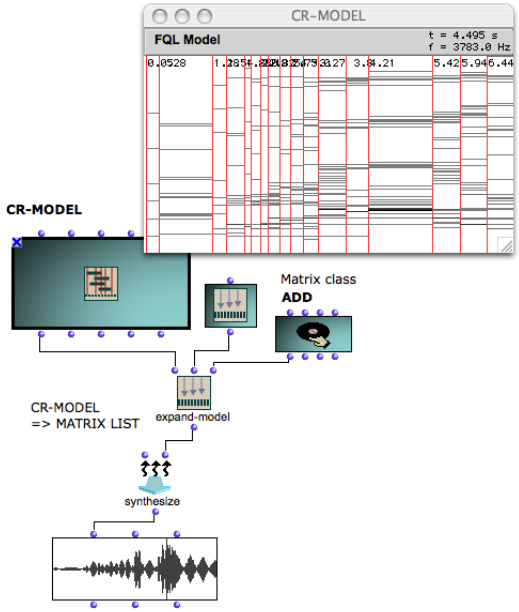

Fig. 9. Converting the contents of a `cr-model` into a list of OMChroma matrices for synthesizing sounds: the type of matrix is determined by the object connected in the rightmost input of `expand-model`, the matching rule is defined in the `cr-control` object connected to the second input.

V. CR-CONTROL: BEHAVIORAL RULES FOR DATA PROCESSING

The conversion of a `cr-model` into primitive events (matrices) is a process where compositional interaction can play another significant role, since it allows to specify how the actual inputs of the events are computed from the abstract initial data.

A closer look to the example of figure 9 shows that `expand-model` has 3 inputs: the `cr-model`, a second object which we call `cr-control`, and an instance of one of the OMChroma classes discussed in the previous section (called `ADD` in the figure).

This class determines the type of matrices into which the `cr-model` will be converted. The `cr-control` specifies how this conversion is to be carried out: it consists in a set of rules applied to the `cr-model` at each step of the conversion, i.e. for each data chunk defined by the time structure. The `cr-control` is associated to a special editor, which allows to define the conversion rules in a way similar to a classical OpenMusic patch, with visual programming tools. Figure 10 shows such an editor.

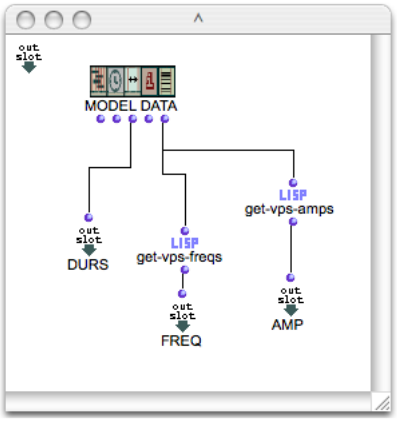

Fig. 10. Matching specification between the `cr-model` data and the matrix parameters in a `cr-conrol` editor.

In a `cr-control` editor, the box `model-data` represents the data of a `cr-model`, from which either general or specific information related to the current data chunk (i^{th} chunk) can be extracted. During the execution of the program, the value of this box will be updated at each iteration and will provide information such as the current rank of iteration, the duration of the current interval, the frequency values, etc. All this data can be used to instantiate the matrices/events.

The outputs of the `cr-control` editor are added by the user, and given a name corresponding to a targetted slot of the matrix. They will be considered only if the matrix class specified to the `expand-model` function effectively owns a slot of this name, and then be supplied to this slot at the time of instantiating the matrix.

For simplicity's sake, figure 10 implements a very straightforward conversion from a `cr-model` to a list of matrices: the frequencies of each VPS as well as the amplitudes and durations are assigned to the `freq`, `amp` and `durs` slots of the matrix. The figure 11 shows a slightly more elaborated rule: one of the slots (`amp`) is determined by a static data structure, independent from the `cr-model` data, whereas the durations are stretched (multiplied by two).

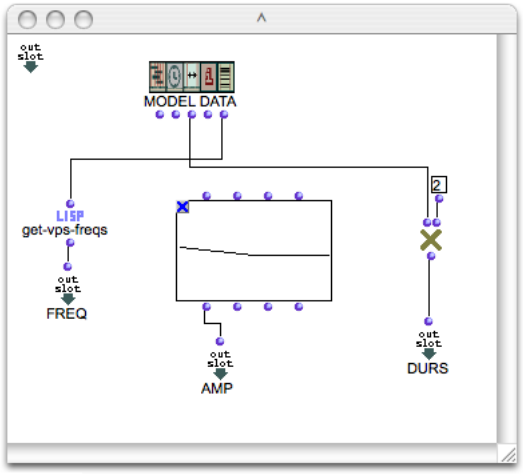

Fig. 11. Custom matching processes defined in the `cr-control` editor

Finally, figure 12 uses other specific tools for a further elaboration of the matching process. The corresponding rule uses the relative position of an event (i.e. the position of the current data chunk) within the global `cr-model` in order to compute the values of `durs` and `amp` by interpolating between predefined data (numbers and breakpoint functions, respectively).

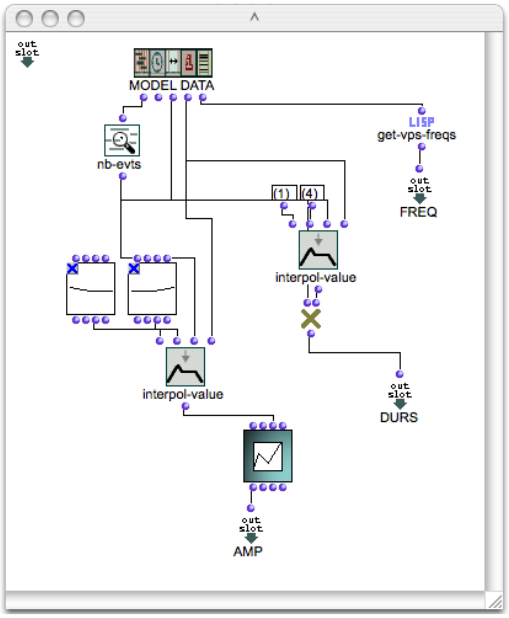

Fig. 12. Using the chunk's position to compute interpolations: the durations are multiplied by a factor varying from 1 to 4 according to the position; the amplitudes evolve from one profile (low frequencies louder) to another (high frequencies louder).

Since the slots of the OMChroma matrices respect a common naming convention (e.g. `freq`, `durs`, `amp`, etc.), it is conceivable to connect the contents of a `cr-model` to any matrix owning these slot names, while using the same `cr-control` matching rules. This is what is illustrated in figure 13.

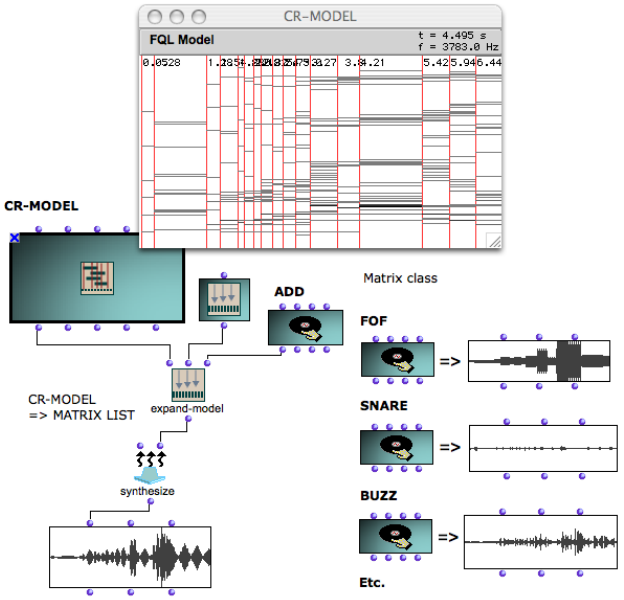

Fig. 13. Data conversion from a `cr-model` to different types of matrices.

VI. Conclusion

The system we presented provides high-level compositional tools dealing with the generation of parameters for sound synthesis. One of its main features is that most of the user interaction is done indirectly, via programming and behavioural rules, and not directly in the actual data (which are generally too complex to be handled manually). The concrete data usually come from analysis processes, although they could also be algorithmically generated. However, neither `cr-model` nor `cr-control` require a direct manipulation on them.

Programs and data are hence mixed in the structures in order to integrate the structural and behavioural properties of the sound descriptions.

It may be worth noticing that the notion of "model" used here has a different meaning in comparison to the usual approach of sound synthesis, where it is implicitly linked to the type of synthesis technique, and therefore determines the way sounds must be described and represented. In our system, a "model" is mainly constituted by the data, and the compositional rules will specify how it will be converted into sounds. The synthesis technique is now a mere variable of the overall process.

Although recurring to external programs generating SDIF-files is a standard practice, using the sound-analysis features available in OpenMusic might also permit to carry out a complete process "from sound to sound", that is, starting from a sound, using the analysis tools to create a `cr-model`, modifying it and converting it back into a sound. Figure 14 summarises all these features in a unique OpenMusic patch.

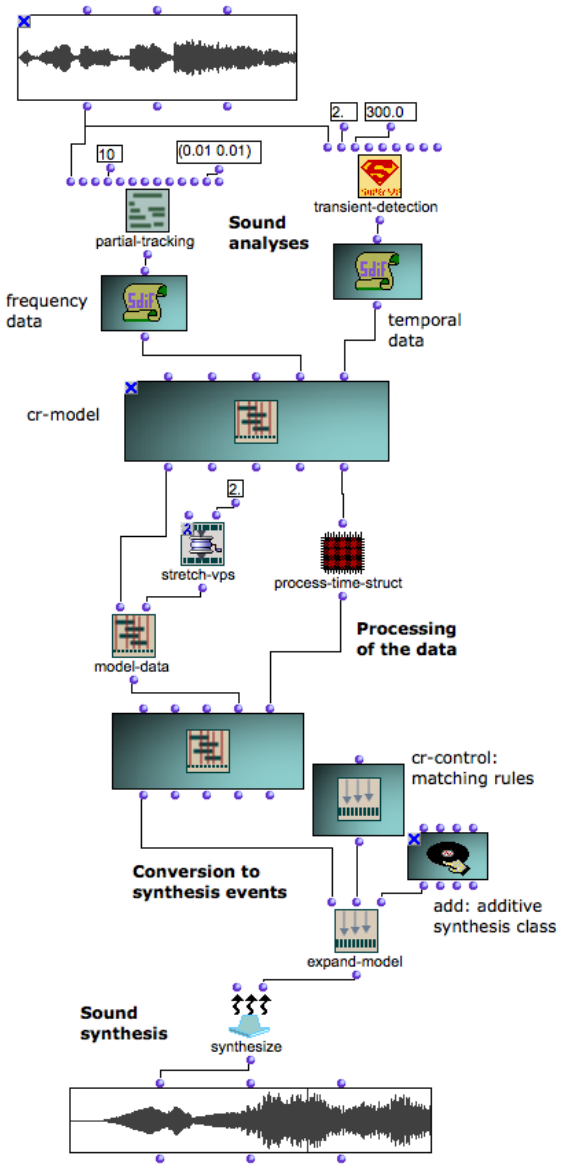

Fig. 14. Example of a full sound analysis/processing/synthesis process carried out in OpenMusic: sound descriptions are obtained from analysis tools, processed in the compositional framework, and converted back to synthesis parameters in order to generate other sounds.

The possibility to define personalised functions at various stages of the process, depending on the needs of each user, ensures that the composer has the maximum degree of freedom: she or he can rely on built-in functions, create new ones, or write patches that modify the behaviour of each structure locally. Since the imagination of a composer is potentially infinite, there is no way to come up with a closed system. Indeed, the necessity of having built-in tools, for reasons of efficiency and generality, while allowing for a total personalisation of the system is a crucial issue in all the environments for the control or sound synthesis. It often leads to thorny issues of ergonomics, and of quantity of data vs. representation and manipulability. Devising the appropriate interface was a major concern in the design of the whole Chroma project.

REFERENCES

[1] Agon, C. *OpenMusic : un langage de programmation visuelle pour la composition musicale*, Ph. D. Thesis, Université Pierre et Marie Curie, Paris, 1998.

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

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

[4] Bogaards, N. and Röbel, A. ”An interface for analysis-driven sound processing”. *AES 119th Convention*, New York, USA, 2004.

[5] Bresson, J. ”Sound Processing in OpenMusic”. *Proceedings of the International Conference on Digital Audio Effects (DAFx-06)*, Montreal, Canada, 2006.

[6] Doval, B. and Rodet, X. ”Estimation of Fundamental Frequency of Musical Sound Signals”. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Toronto, Canada, 1991.

[7] Klingbeil, M. ”Software for Spectral Analysis, Editing and Synthesis”. *Proceedings of the International Computer Music Conference*, Barcelona, Spain, 2005.

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

[9] Misra, A., Cook, P. R. and Wang, G. ”A New Paradigm for Sound Design”, *Proceedings of the International Conference on Digital Audio Effects (DAFx-06)*, Montreal, Quebec, Canada, 2006.

[10] Röbel, A. ”Transient detection and preservation in the phase vocoder”, *Proceedings of the International Computer Music Conference*, Singapore, 2003.

[11] Stroppa, M. ”Structure, Categorization, Generation and Selection of Vertical Pitch Structures: a Musical Application in Computer-Assisted Composition”, *IRCAM Document*, Paris, 1988.

[12] Stroppa, M. ”Paradigms for the high level musical control of digital signal processing”, *Proceedings of the International Conference on Digital Audio Effects (DAFx-00)*, Verona, Italy, 2000.

[13] Stroppa, M., Lemouton, S. and Agon, C. ”OmChroma ; vers une formalisation compositionnelle des processus de synthse sonore”, *Actes des Journées d’Informatique Musicale JIM’02*, Marseille, France, 2002.

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