GESTURES, EVENTS AND SYMBOLS IN THE *BACH* ENVIRONMENT

Andrea Agostini
Freelance composer

Daniele Ghisi
Composer - Casa de Velázquez

RÉSUMÉ

Environments for computer-aided composition (CAC for short), allowing generation and transformation of symbolic musical data, are usually counterposed to real-time environments or sequencers. The counterposition is deeply methodological: in traditional CAC environments interface changes have no effect until a certain 'refresh' operation is performed, whereas real-time environments immediately react to user input. We can call the former approach *speculative*, as it is associated with a careful, preemptive formalization of musical ideas, and the latter *performative*, as it is intrinsically apt to deal with extemporariness and able to react to gestures - in the broadest sense of the term - as soon as they are performed. This distinction is by no means natural, since interactivity is an essential performative aspect of the musical discovery process. The reunification of the performative and speculative aspects is obtained via a MaxMSP library named *bach: automatic composer's helper* [1], which is a set of tools for symbolic processing and graphical music representation, designed to take advantage of MaxMSP's facilities for sound processing, real-time interaction and graphical programming. The *bach* environment is capable to extend the typically speculative paradigm of CAC, embracing the performative one as well, thus becoming a very natural tool for composers both during their own discovery process and in concert situations: external gestures, codified in input as events, handle any parameter change within the symbolic framework, immediately affecting a resulting real-time score.

1. INTRODUCTION

It is not surprising that, since the advent of computers, there has been great interest on how to take advantage of the superior precision, speed and power of electronic computers in music-related activities. The probably best-known (and commercially successful) direction has proven being the generation and transformation of sound. In recent years, inexpensive personal computers (and lately even top-end mobile phones) have gained the ability to perform professional-quality audio transformation and generation in real-time. On the other hand, several systems have been developed to process symbolic data rather than acoustic ones - 'notes' rather than 'sounds'. These systems can be roughly divided into tools for computer-assisted music engraving (Finale, Sibelius, Lilypond...) and tools for computer-aided composition (CAC for short, allowing generation and transformation of symbolic musical data, like OpenMusic, PWGL, Common Music...). Moreover, at least two graphical programming environments, the closely related Max and PD, have MIDI control and sound generation and transformation among their main focuses - but at the same time they are capable to deal with arbitrary set of data, input/output devices and video. Indeed, the boundaries between all these categories are extremely fuzzy: music engraving systems often allow non-trivial data processing; some sequencers also provide high-quality graphical representation of musical scores and sound treatment; modern CAC environments include tools for sound synthesis and transformation. It should though be remarked that Max and PureData have very crude native support for sequencing, and no support whatsoever for symbolic musical notation. Moreover, an orthogonal distinction should be made between real-time systems, which 'immediately' react to interface actions (such as Finale, Max, ProTools...) and non-real-time systems, where these actions have no effect until a certain 'refresh' operation is performed (such as Lilypond, OpenMusic, PWGL). The latter is the case of typical CAC environments; yet, this is by no means natural: there is no deep reason why symbolic processing should not be performed in real-time, the only reason being what we could label a 'technological anachronism'. In fact, advanced symbolic computation and musical representation can easily become very costly in terms of processing power, and personal computers could not stand its computational weight until a few years ago. This situation has established a traditional separation that, although still lingering on, is no longer justified, since interactivity is an essential performative aspect of the musical discovery process, allowing any input gesture to immediately affect a given score.

2. GESTURES, EVENTS, SYMBOLS

Before getting to the core of the article, we have to briefly investigate the meaning of the word 'gesture' - or, more appropriately, the broader meaning of a family of words derived from the Latin 'gestus', in turn the past participle of the verb 'gerere', in several modern European languages: the French *geste*, the German *Geste*, the Italian and Spanish *gesto*... While there is a broad area of superposition in the meanings of all these words, some specific connotations are more apparent in some languages than in others. Generally, we will use the English 'gesture' in its broader sense, encompassing all the possible different

[1] http://www.bachproject.net

shades of connotations.

We talk about hand gestures, musical gestures, demonstrative gestures, hardly separating the literal meaning and the metaphorical usage of the word. In any case, it appears to us that roughly all gestures have at least one of these two 'features' inside their meaning:

- temporality: gestures are seen or interpreted in some temporal development. This can either involve actual movements in space (a finger hitting a key on a piano, or clicking on a mouse), or acoustic time-processes (a violin crescendo), or any synaesthetic extension (a black line which gets thicker on a white background, interpreted as the operation in time of the painter who drew it);

- extrapolation: a gesture is an identified portion of a whole, which stands out for some particular reason. This extrapolation is *always* an operation of meaning: gestures comes out by according a particular meaning with respect to the whole: a straight white line on a black canvas is a gesture since it stands out from the background; the gesture of a finger hitting and then releasing a piano key is an extrapolation from the all the movements of the finger in time; an orchestral fortissimo in a pianissimo situation is a gesture inasmuch as it stands out from it; even an entire piece, like Beethoven's *Grosse Fuge* may be a gesture in itself, if seen inside the whole Beethoven's production, or inside the entire history of music.

These two characterizations still reflect two etymological aspect within the Latin *gerere*: temporal behaviour (to bring, to pass, to spend...) and event-centering (to do, to produce, to show...).

Focusing on the musical case, we are not only interested in all the physical gestures (the violinist moving the bow, the pianist hitting the key, the composer clicking with the mouse in the patch), but also the musical gestures, conceived as meaningful properties of a score, an interpretation, an improvisation. Indeed, a real-time CAC involving symbolic musical gestures is not only possible but also achieved by the *bach* environment.

3. REAL-TIME GESTURE HANDLING AND REPRESENTATION WITH *BACH*

The creation and modification of a musical score is not an out-of-time activity, but it follows the composer's discovery process and develops accordingly. Persuaded that a composer-friendly CAC environment needs to cope with this fact, it was our priority that any data could be handled in real-time: composers working with symbolic gestures may need the machine to quickly adapt to new configurations, as they change some parameters.

Of course, *bach* is not the first attempt to address this dichotomy. Several interesting projects have been developed, linking real-time environments to graphical representations of both classical and non-classical (and potentially non-musical) scores, including OpenTimeLine [2] [6], MGC [3] [10], INscore [4] [5]. In at least one case, namely MaxScore [5], this is augmented by a very sophisticated editing interface. With *bach*, as shown in [1], we have tried to achieve a coherent system explicitly designed for computer-assisted composition: the advanced editing interface is conceived to take advantage of the combination of interactive writing and algorithmic control of symbolic musical material. Thus the set of graphical editors is focused on the representation of the underlying structures of music, rather than their graphical nuances. Therefore *bach* is not a single editor object, rather a rich set of tools for symbolic processing and graphical musical representation; each element in the environment is designed to take advantage of Max's facilities for sound processing, real-time interaction and graphical-programming.

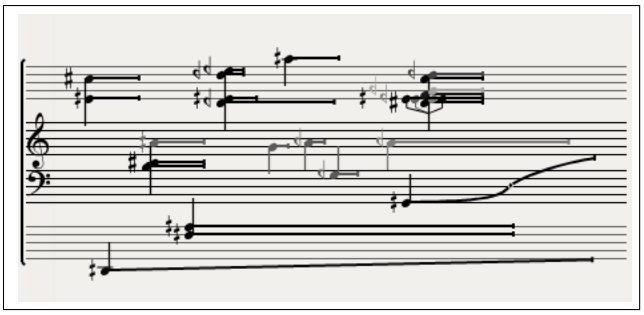

Figure 1. A simple *bach.roll*, representing a musical score in proportional notation

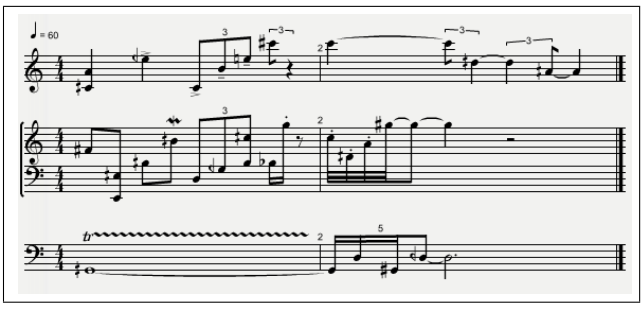

Figure 2. A simple *bach.score*, representing a traditional score (with measures, time signatures and tempi).

bach follows the general data-flow paradigm of Max: users build a sort of 'assembly chain' made of very specialized operators. In fact, any action feeding data into the 'assembly chain' can be considered as a specific event, happening at a certain moment in time. This applies particularly well to the typical performative usage of Max: events can be, for instance, actions performed on a MIDI controller, or in front of a camera, or sounds captured by a microphone - as well as mouse clicks and letters typed

[2] http://dh7.free.fr
[3] http://www.numediart.org/projects/12-2-mouse-gesture-composer
[4] http://inscore.sourceforge.net
[5] http://www.computermusicnotation.com

on an alphanumeric keyboard, of course. If the reaction to an event is immediate enough, we have the perception that the system is responding in real-time to our input. This leads to the idea that, if we assume that a gesture can be seen as a particularly structured sequence of punctual events, on a higher level of abstraction Max can indeed be controlled in real-time by gestures.

If we think at standard CAC environments (such as Patchwork, OpenMusic and PWGL), instead, we can observe that their programming paradigm is deeply different, as the entry of data does not trigger any immediate reaction: in these systems, an explicit evaluation command must be given to the machine in order to perform the desired operation. The difference is less subtle than it might appear, both on the conceptual and practical point of view. Of course, the evaluation command is just a key pressure, or a mouse click; but whereas this single action might not be very critical in a non-real-time context, it becomes crucial when synchronicity matters, or when the data flow comes, for instance, from a MIDI keyboard, or a microphone, or a video camera. The non-real-time approach is thus unable to properly keep track of a stream of incoming events, be they a sequence of keys pressed by a player or a series of instructions guiding the composer's thought.

It should be clear at this point that the choice of implementing a CAC environment within a real-time event-driven system leads to important consequences in the kind of scenarios it can be applied to. Firstly, if the composer's interface choices affect in real-time the symbolic result, the machine feedback is way more extensive and intuitive, allowing the user to handle and validate a much larger set of possibilities. In this sense, *bach* allows a sort of 'trial and error' approach to symbolic data, in a similar way as composers generally do while writing electronic music. But, most importantly, *bach* introduces the possibility of a gestural approach to symbolic generation of, and intervention upon, musical materials. Of course one has to strike a balance between the amount of feedback obtained from the machine and the amount of information that the composer himself can favorably process. As noticed in [1], *bach* is meant to close the gap between the 'performative' and 'speculative' aspects of tools for computer music, which has pushed the real-time and CAC communities apart.

As a final note on the subject, it is worth to underline that the real-time paradigm is a resource rather than an obligation: score changes are handled by the user, who is completely free to make them happen immediately or only after some sort of 'refresh' operation (as it would be the case in non-real-time environments). This means that, in principle, nothing prevents from using *bach* just like any other CAC environment, and there are cases (such as a fine rhythmic quantization) in which one is obliged to settle in the non-real-time paradigm, since the significant amount of time needed for performing a particular task might actually disrupts the immediateness of response to the user's actions.

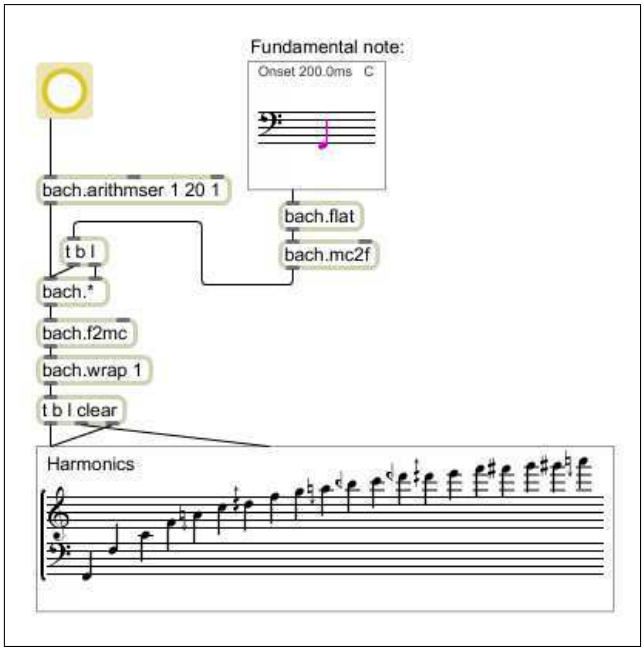

Figure 3. A simple patch calculating the first twenty harmonics of a given note. Either clicking the *button* or dragging the fundamental note will immediately update the result.

4. THE *BACH* ENVIRONMENT

4.1. The components of *bach*

As already stated, *bach* is a library of objects and abstractions for the software Max, the distinction between objects and abstractions concerning more the implementation than the actual usage of these modules. Both appear to the user as graphical widgets that can be linked to others in order to build the 'assembly chain' that will perform the desired operations. The difference lies in the fact that objects are written in C and - unless the source code is made available - behave like non-editable 'black boxes', whereas abstractions are combinations of objects and other abstractions, which users are always allowed to open and edit; the drawback for this flexibility is that abstractions are usually far less efficient than properly written objects, and some functionalities simply cannot be implemented without recurring to some C coding: this is particularly true when dealing with custom data types and advanced graphical operations, which is the case for a large portion of the *bach* environment. In Fig. 3 we show a sample patch containing some *bach* objects and abstractions.

In addition to that, great care has been posed in the documentation of *bach*. Every object and abstractions comes with a fully detailed help file; a general help patch can be recalled from the 'Extras' menu of Max; a growing set of tutorial patches provide a step-by-step introduction to most features of the system. Moreover, a user forum exists and a new set of video tutorials is currently under development.

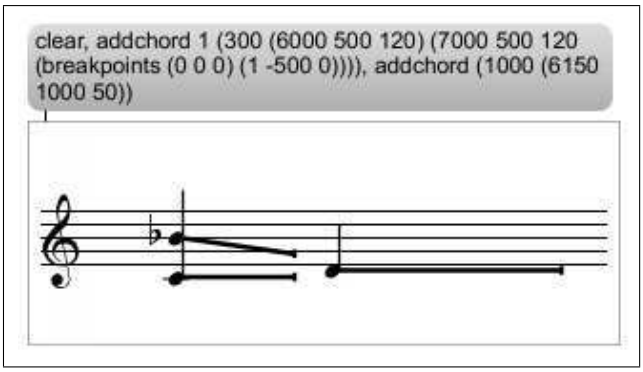

Figure 4. Any notation object in the bach library can be edited either graphically or via Max messages. In this case we clear the *bach.roll*, and then add two chords.

4.2. The GUI objects

We already introduced the *bach.score* and *bach.roll* objects. They both provide graphical interfaces for the representation of music: *bach.score* expresses time in terms of traditional musical units, and includes notions such as rests, measures, time signature and tempo; *bach.roll* expresses time in terms of absolute temporal units (namely milliseconds), and as a consequence has no notion of traditional temporal concepts: this is useful for representing non-measured music, and it also provides a simple way to deal with pitch material whose temporal information is unknown or irrelevant. It should also be noted that the implementation of traditional temporal concepts in *bach.score* is quite advanced, as it allows multiple simultaneous time signatures, tempi and agogics (see Fig. 7). Besides this fundamental difference, the two objects offer a large set of common features, including the following:

- Editing by both mouse-and-keyboard interface and Max messages (see Fig. 4): the editing facilities combine features typical of WYSIWIG music engraving tools (e.g., creation and editing of notes and chords by graphical interaction) and MIDI and audio sequencers (e.g., immediate playback of a note along with its set of meta-data - see below), and allow the user to program algorithmic, computer-driven actions upon virtually every parameter of the musical data and graphical representation. Moreover, any interaction by Max message results in the immediate update of the graphical display, and any mouse-and-keyboard interaction is properly notified: in this way, the two possible ways of interaction are seamlessly integrated with each other. Both mouse-and-keyboard and message editing operations are undoable.
- Support for microtonal accidentals of arbitrary resolution (see Fig. 5).
- Wide possibility of intervention over the graphical parameters of musical notation, including spacing (see Fig. 6), fonts and colors.

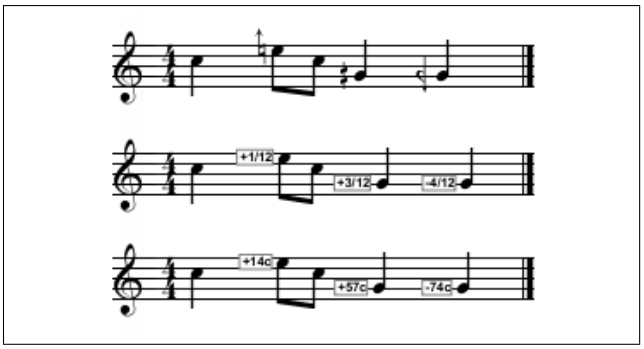

Figure 5. Semitonal, quartertonal and eighthtonal divisions are supported via the standard accidental symbols (upper example). All other microtonal division are supported as well, as long as symbolic accidentals are replaced by a label with the explicit fraction of tone (middle example), or with the cents difference from the diatonic note (lower example).

- Ability to associate to each note various types of meta-data, including text, numbers, files and breakpoint functions (see Fig. 11).
- Variable-speed playback capability: both *bach.score* and *bach.roll* can behave as advanced sequencers, and the whole set of data (such as pitch, velocity and duration information) and meta-data associated to each note is output at the appropriate time during playback, thus making both objects extremely convenient for controlling synthesizers and other physical or virtual devices.

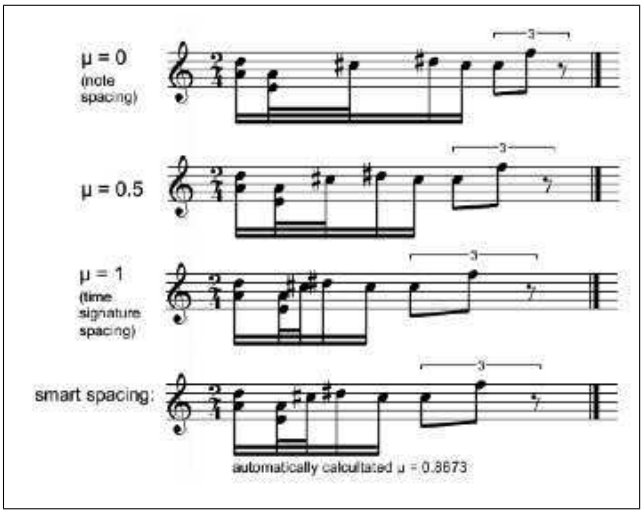

Figure 6. *bach.score* offers different spacing algorithms. The one showed here, as the parameter μ ranges in $[0, 1]$, interpolates between a note spacing ($\mu = 0$) and a time signature spacing ($\mu = 1$). The 'smart spacing' algorithm offers an automatic computation for μ (lower example).

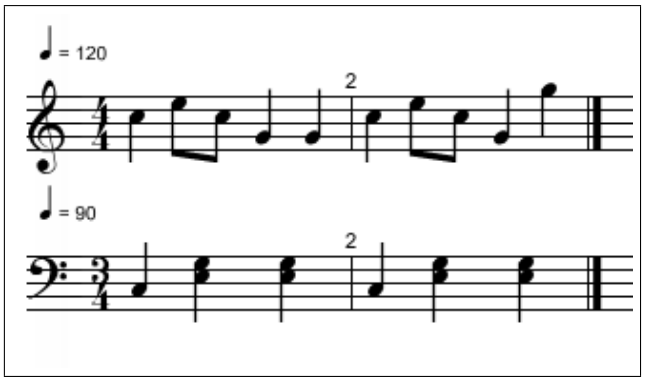

Figure 7. *bach.score* supports the cohabitation of different time signatures and tempi, and adapts the graphical alignment accordingly.

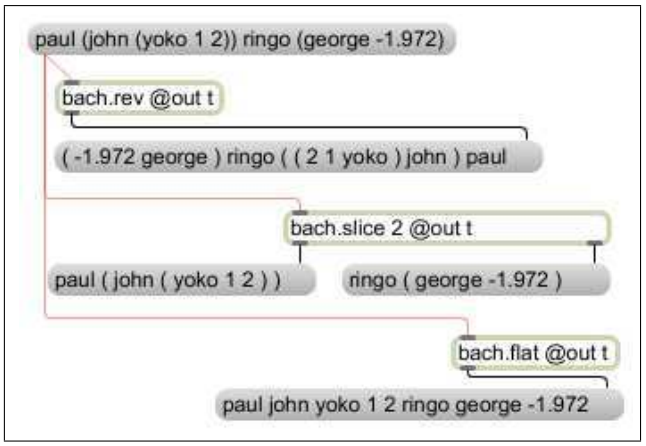

Figure 8. *bach* has a wide range of objects capable to perform standard structure operation on *llll*s, such as reversing, slicing, flattening, rotating and so on. In the picture, we see the results of reversing, slicing and flattening a list. Moreover, most operations can be constrained to some levels of depth only (e.g., one can easily remove the parentheses of all the intermediate levels, leaving the outermost and the innermost levels untouched).

4.3. Data types, representations and operators

bach also provides Max with two new data types: rational numbers and a tree structure called *llll*, an acronym for *Lisp-like linked list*. Rational numbers are extremely important in music computation, as they express traditional temporal units such as 1/2, 3/8 or 1/12 (that is, a triplet eight note) as well as harmonic ratios. The *llll* is an ordered collection of elements, each being a number (integer, floating-point or rational), a symbol (that is a text item, roughly equivalent to the concept of character string) or an *llll* - thus allowing recursively nested structures of arbitrary depth and complexity. The *llll* has been chosen for both similarity with the Lisp language, in a way to ease communication with the major existing CAC environments, and the need to establish a data structure powerful enough to represent the complexity of a musical score, but flexible enough to be a generic data container lending itself to arbitrary manipulations through a relatively small set of primitives.

In fact, the large majority of the modules of the *bach* library are tools for working upon *llll*s, performing basic operations such as retrieval of individual elements, iteration, reversal, sorting, splicing, merging and so on (see Fig. 8). Some subsets of the library are applicable to *llll*s satisfying certain given conditions: e.g., it is possible to perform mathematical operations over *llll*s solely composed by numbers; a set of operators for matrix calculus only works with appropriately structured *llll*s; and so on.

At the intersection between the modules for musical notation and the list operators is a family of objects performing operations upon *llll*s containing musical data. It is worth noting that different *bach* objects exchange musical scores in the form of specifically-structured *llll*s, whose contents is entirely readable and editable by the user; this is different from what happens e.g. in OpenMusic, where the exchange of musical data usually involves opaque objects. As a consequence, in *bach* strictly musical operations such as rhythmic quantization are just extremely specialized operations upon *llll*s, which of course can be performed only if the *llll* itself is structured as a musical score, and if its contents are consistent from the point of view of musical notation.

The structure of a *llll* representing a *bach.score* can appear quite complex at first sight, but the organization of its contents is meant to be extremely rational: after a header section containing global information such as the clefs or the types of meta-data appearing in the score, we find a sub-tree whose branches correspond to one voice each; each voice branch contains branches for each measure; each measure branch contains some measure-specific information (such as time signature) and branches for each chord; each chord branch contains some chord-specific information (such as duration) and branches for each note; and each note branch contains pitch and velocity leaves, as well as possible further specifications, such as glissando lines, enharmonic information and meta-data. The *llll* representing a whole *bach.roll* has essentially the same structure, except that the measure level is not present. With the provided set of list operators, specific pieces of information referring to single elements or sections of the score are not difficult to locate and manipulate. Moreover, both *bach.score* and *bach.roll* provide simplified ways to retrieve and enter only specific sets of values to operate upon (e.g. pitches or velocities only), which greatly ease the implementation of most algorithmic operations.

4.4. Other remarks

A choice has been made of not providing a pre-defined set of utilities performing basic musical transformations. So, *bach* does not contain specific tools for, e.g., transposition or retrogradation: but they can easily be programmed in terms of, respectively, a sum operation upon each note's pitch, or the reversal of the list containing the chords.

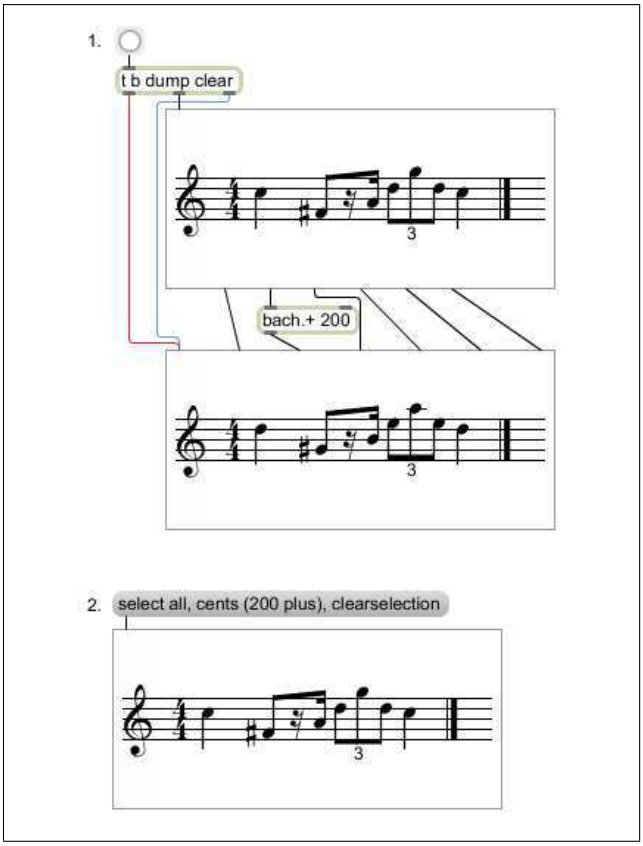

Figure 9. Two standard ways to perform a transposition in *bach*. The first one rebuilds a score having the same parameters of the first score, except for the pitches (we add a tone to each pitch). The second one achieve the transposition in-place, by selecting (by message) all notes, shifting them up, and then deselecting them (as soon as one clicks on the grey message, the result will be updated in the score).

Of course these are extremely simple cases, but the documentation provided with the library contains extensive information and examples showing how to set up several kinds of musical operations; programming *bach* should generally not be harder than, for instance, OpenMusic or PWGL, and the aim of the project is to provide a framework of tools enabling the user to program virtually any desired operation. In some future, we might develop a library of high-level tools covering a generic set of strictly musical operations; but it should be made clear that this library will be a separate project based upon *bach*, rather than a part of it. As a final note, a C API will shortly be available, allowing programmers to build new external objects for Max that will be able to access the *bach*'s data structures, low-level list operators and musical notation tools.

All this should show that *bach* is somehow placed at the intersection of several categories of musical software, as outlined in the introduction of this paper. On one hand, its capabilities of graphical representation of musical scores typically belong to music engraving systems, although it should be noted that *bach* lacks (at least in its current state) some essential features of this kind of programs, such as the ability to deal with a whole page layout, and - as a consequence - to print musical scores. On the other hand, most of its features are conceived in order to make it a tool for CAC as powerful as the traditional Lisp-based environments. Finally, it can be used as the core of an extremely advanced and flexible sequencer, with the ability to drive virtually any kind of process and playback system. And, of course, it can lend itself to innovative applications exploiting the unique convergence of these different paradigms and the real-time behavior.

5. EXAMPLES

We give here some examples of how the *bach* framework can work in practice.

5.1. A sequencer 2.0

As a first example, a score can be a customizable sequencer, whose content is completely open to any real-time process the user might want to realize.

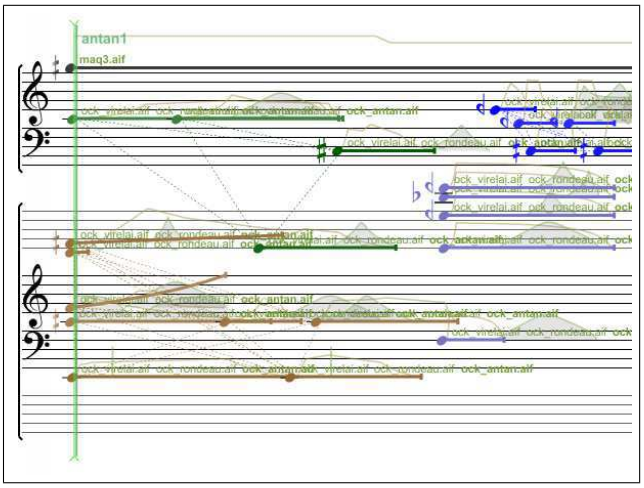

Figure 10. A portion of the electronic score for the piece *Chansons*, by D. Ghisi, consisting in a *bach.roll* object. Notes are grouped into larger structures (differently coloured), in order to better represent electronic gestures in the score.

Notes carry extra information (in some user-accessible *slots*), specifying the parameters for the processes which will concern them. This information can have various forms; to name a few: function envelopes, numeric data, filter envelopes (both static and dynamic), text instructions for a synthesizer, file paths, matrix content for routing, *llll*s, spatialization trajectories, colors. Not only all this information is accessible via messages, but it is also directly modifiable via the interface: by hitting a specific (customizable) key, one can pop out a note's slot window, and edit the information directly (see Fig. 11). The structure of each slot (data type, name, hotkey, window size, domain, range, slope...) is completely up to the user; the content of some important slots can be always kept in the

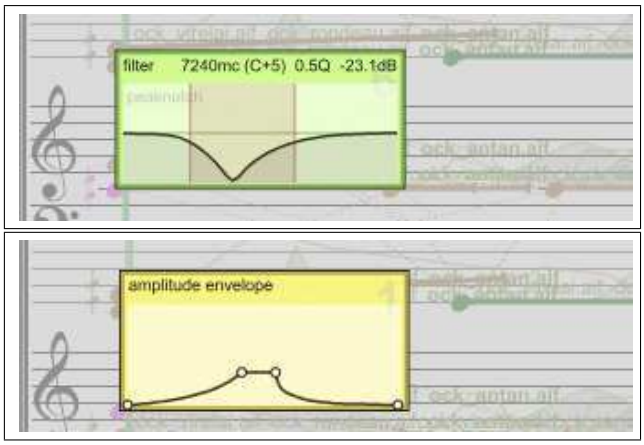

Figure 11. Two examples of slot windows.

score, so that one can constantly sees a representation of the slot content, and not only when the slot window is open (for instance, in Fig. 10 we always have under our eyes some yellow amplitude envelopes, the names of some soundfiles associated to the notes, and some filters gain responses).

With respect to other approaches to the writing of electronic music, this representation lets the user stick with traditional notation. Larger musical gestures can be handled by grouping chords (grouped chords may assume the same color and be linked with dashed lines, as one can see in Fig. 10); a group has the advantage to be shiftable as a single element, even if each note inside the group still retains its specific parameters and slot content. One can also set markers to segment portions of the score.

The slot structure is flexible enough to handle symbolic feedback. For instance, by selecting a note and pressing a key, one can link a search in a database in order to somehow match the specific pitch behavior of a note, or some data contained in its slots; one can then re-inject the result of the search in a note's slot (see Fig. 12).

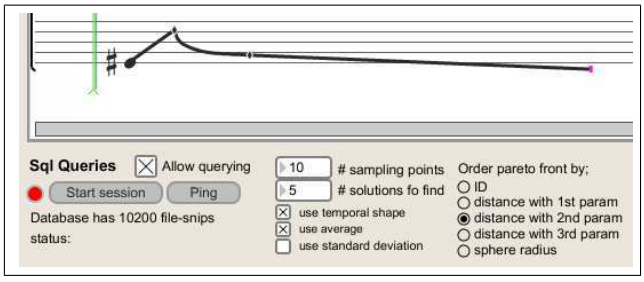

Figure 12. Portion of panel, for the piece *abroad* by D. Ghisi, querying a soundfile database, searching some closest match (with respect to the main frequency). The search engine has been developed by Philippe Esling at Ircam.

At the same time, thanks to the possibility to retrieve in real-time all the information related to the details of the graphical display of the score, it is straightforward to keep a video sequence constantly aligned to the musical score. In this way, when working with a video, one can always be aware of which video frame each musical event is synchronized to (see Fig. 13).

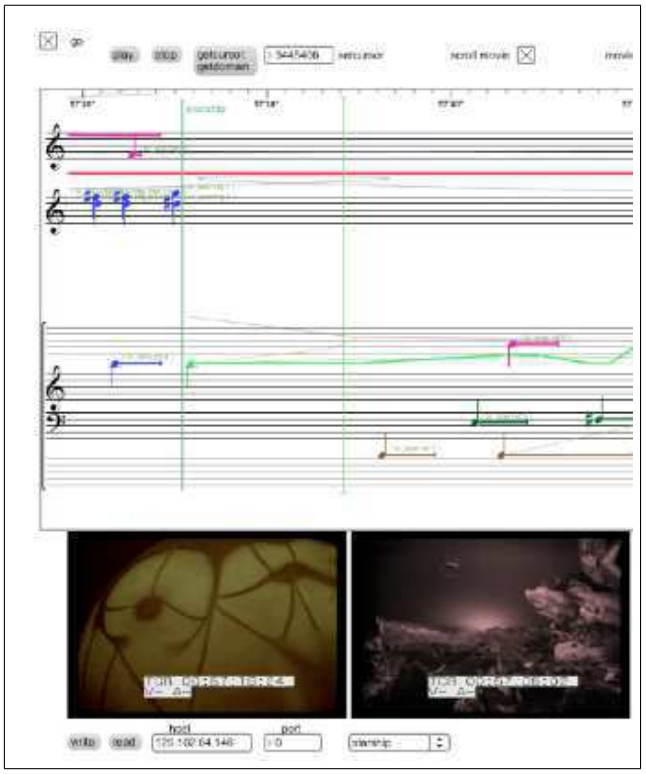

Figure 13. A portion of the patch used by A. Agostini to manage the electronic score for the film *Wunder der Schöpfung*. Each note contains a set of instructions for a synthesizer, expressed as text commands and graphical marks. When the score is played, all the information connected to each note is sent to the appropriate synthesizer voice, represented by the note color. Below the score, some frames of the movies are shown, providing a visual reference to the position of the musical events. In a separate window, not shown here, the frame corresponding to the exact position of the play cursor (the thin vertical bar) is shown in real-time, allowing for fine control over the sound and image synchronization.

5.2. A real-time symbolic granulation module

A large category of processes concerns CAC operations being performed strictly in real-time. As an example, one can easily build with *bach* a machine performing real-time granulation on symbolic data (Fig. 14). Granulation parameters can mirror the standard electroacustic granulation parameters, such as grain size, onset position, grain interval or density, grain speed, and more. When the machine is turned on, some portions of a score are extracted from a given original score (each fragment corresponding to a single grain), and then immediately integrated into a bach.roll containing the accumulated result of the granulation. A coherent playback system can be also set up, in order to have a simulated result playing accordingly. Of course, parameter changes can be handled in real-time,

depending on incoming data, such as an audio stream or a gesture tracking.

6. FUTURE DEVELOPEMENTS

At the time of writing, bach is in the phase of development called alpha in technical jargon: although the system is usable, not all the intended features have already been implemented. Namely, some important points are necessary for a complete symbolic gesture handling:

- An improved gesture hierarchy, allowing the user to give names to symbolic elements (such as chords, notes, groups), and allowing groups to be in turn grouped at higher levels. The group-wise operations need to be improved, and following this path even further, groups can have specific properties or slots.
- Support for MIDI, MusicXML and SDIF files, which constitute some of the most common ways of exchanging scores and symbolic data with other programs and devices.
- A solver for Constraint Satisfaction Problems, which are a very intuitive way to express problems according to the desired features of the solution, rather than the steps needed to build it.

Notice that the software developement situation might have changed at the time of publication, and some or all of the hereby proposed features might already be partly or fully implemented.

7. ACKNOWLEDGEMENTS

This project has been partially undertaken during the permanence of the authors at IRCAM, first as students at the Cursus in Composition and Musical Informatics and then as composers in research. We wish to thank Carlos Agon, Arshia Cont, Eric Daubresse, Emmanuel Jourdan, Serge Lemouton, Jean Lochard, Mikhail Malt and all the other people at IRCAM who have supported and encouraged the work, although not an internal IRCAM project. We wish to thank Mika Kuuskankare for his precious advice and support. We also wish to thank DaFact [6] for actively supporting and sponsoring the development of *bach*.

8. REFERENCES

[1] A. Agostini, D. Ghisi, *Real-time computer-aided composition with bach*, Contemporary Music Review, to appear.

[2] G. Assayag et al., "Computer assisted composition at Ircam: From patchwork to OpenMusic", *Computer Music Journal*, 23 (3), pages 59-72, 1999.

[3] A. Cont, *Modeling Musical Anticipation*, PhD Thesis, Paris, 2008.

[4] P. Esling, "Time Series Data Mining", *ACM Computing Surveys*, to appear.

[5] D. Fauber, Y. Orlarey, S. Letz, "INScore - An Environment for the Design of Live Music Scores", *Proceedings of the Linux Audio Conference*, LAC 2012

[6] D. Henry, "PTL, a new sequencer dedicated to graphical scores", *ICMC Proceedings*, pages 738-741, Miami, International Computer Music Association, 2004.

[7] M. Laurson, J. Duthen, "Patchwork, a graphical language in preform", *Proceedings of the International Computer Music Conference*, pages 172-175, Ann Arbor, International Computer Music Association, 1989.

[8] M. Puckette, "A divide between 'compositional' and 'performative' aspects of Pd", *Proceedings of the First Internation Pd Convention*, Graz, Austria, 2004.

[9] C. Seleborg, *Interaction temps-réel/temps différé*, mémoire ATIAM, Marseille, 2004.

[10] F. Zajéga, L. Rebousière, "Mouse Gesture Composer", *QPSR of the numediart research program*, pp. 63-70, 2010.

[6] http://www.dafact.com

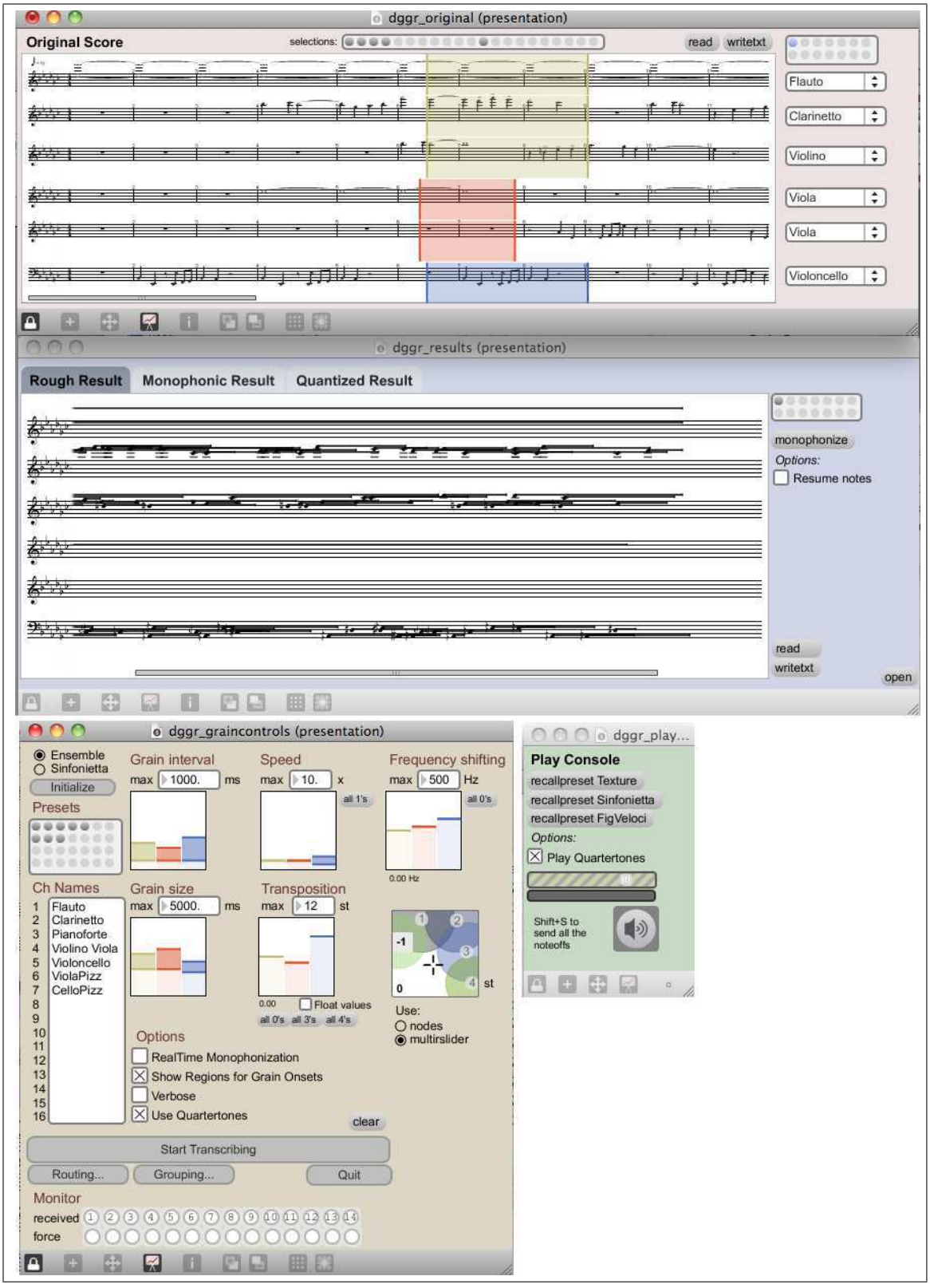

Figure 14. Screenshot of a patch achieving a real-time symbolic granulation. The original score (upper reddish window) has some markers to determine and modify the grain regions. Parameters are handled in the lower ochre window. When the user press the 'Start transcribing' button, the result appears and accumulates in the middle blue window. When desired, one may then make it monophonic (if needed), refine it, and finally quantize it. Every parameter is user-modifiable and affects in real-time the result, as in any electroacoustic granulation machine.