*Contemporary Music Review*, 2013
Vol. 32, No. 1, 41–48, http://dx.doi.org/10.1080/07494467.2013.774221

Routledge
Taylor & Francis Group

# Real-Time Computer-Aided Composition with *bach*

Andrea Agostini and Daniele Ghisi



*Environments for computer-aided composition (CAC), 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 shall show in this article that this distinction is by no means natural and that interactivity is an essential performative aspect of the musical discovery process. The reunification of the performative and speculative aspects is obtained via a Max library named* bach: automatic composer's helper*, which is a set of tools for symbolic processing and graphical music representation, designed to take advantage of Max's facilities for sound processing, real-time interaction and graphical programming.*

*Keywords: Computer-Assisted Composition; CAC; bach; Real-Time; Max*

## Introduction

The relationship between music and computation is extremely well known, and it is indeed one of the basic conceptual tools in the understanding and speculation about music itself. The highly refined, quasi-algorithmic systems developed by Flemish composers; the combinatorial complexity of dodecaphonic and serial music; the calculation of the frequency contents of sound, as a starting point for the spectral movement of composers: these are only few examples of how computation can be an invaluable tool for music composition. Moreover, other closely related domains, such as musical analysis, ethnomusicology and psychoacoustics, make an extensive use of computational tools.

It is not surprising that, since the advent of computers, there has been a great interest on how to take advantage of the superior precision, speed and power of electronic computers in music-related activities. Probably the best-known (and commercially successful) direction has proven to be 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. In fact, an ever-growing number of recording and production studios have replaced most of their equipment with a small number of commercial, general-purpose computers.

On the other hand, several systems have been developed to process symbolic data or acoustic ones—'notes' or 'sounds'. These systems can be divided roughly into tools for computer-assisted music engraving (Finale, Sibelius, Lilypond, etc.), tools for sound generation or transformation (Csound, GRM Tools, Waves plugins, etc.), sequencers (allowing recording, editing and playback of audio or MIDI data, like ProTools, Logic or Cubase), and tools for computer-aided composition (CAC, allowing generation and transformation of symbolic musical data, like OpenMusic, PWGL, Common Music, etc.). 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 of dealing with arbitrary sets of data, generic input/output devices and video. Indeed, the boundaries between these categories are extremely fuzzy: music engraving systems often allow non-trivial data processing; some sequencers 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 Pd have very crude native support for sequencing, and essentially none for symbolic musical notation.

Another, orthogonal distinction should be made between real-time systems, which 'immediately' react to interface actions (such as Finale, Max, ProTools, etc.) and non-real-time systems, where these actions have no effect until a certain 'refresh' operation is performed (such as LilyPond, OpenMusic and 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. Indeed, interactivity is an essential performative aspect of the musical discovery process.



## The *bach* Paradigm

Real-time properties of a CAC environment deeply affect the very nature of the compositional process. Composers working with sequencers definitely need them to react immediately as they change envelopes or delete sound files; likewise, composers working with symbolic data might want the machine to adapt quickly to new parameter configurations. The underlying paradigm is that the creation and modification of a musical score is not an out-of-time activity, but follows the composer's discovery process and develops accordingly.

This issue has already been faced by Miller Puckette (Puckette, 2004), and subsequently tackled by Arshia Cont (Cont, 2008). A good overview of the questions raised by the real-time/non-real-time dichotomy can be found in (Seleborg, 2004). In this article, we aim to show that real-time CAC is not only theoretically possible,

but also practically achieved via a library for Max named *bach: automatic composer's helper*. *bach* is a set of tools for symbolic processing and graphical music representation, designed to take advantage of Max's facilities for sound processing, real-time interaction and graphical programming. At the forefront of the *bach* system are two highly interactive and customizable musical notation interfaces, which at the same time serve as sequencers for both musical events (notes and chords) and arbitrary data (e.g. sound files, instructions to a synthesizer, etc.).

Being a Max library, *bach* follows the general Max dataflow paradigm. The essential principle is that users do not call explicitly for operations upon data; rather, they build a sort of 'assembly chain' made of very specialized operators, and as soon as data are received at the beginning of the chain, they are serially processed by the operators, one after another, and the result is retrieved at the end of the chain. This kind of programming paradigm actually reflects, and is a metaphor for, a wide range of non-computer-related experiences, such as the behaviour of the mail system—at the moment we bring a parcel to the post office, we are confident that the correct chain of operations (whose details we neither know, nor care about) will be performed by a correspondingly structured chain of humans and machines, eventually leading to the delivery of the parcel in some remote part of the world. Or, the mechanics of a piano, where the pressure on a key immediately moves a chain of mechanical devices that eventually, in a measurable but usually negligible time, set the corresponding strings in motion so that we can hear the resulting sound. Or, again, a traditional sound recording, mixing and amplification chain, in which the sound entering the chain is transformed into electrical signals, added to other sounds, amplified, filtered and eventually re-transformed into sound. It should be noted that the two latter examples are particularly relevant to Max, as the 'musical instrument' and 'mixing chain' metaphors have informed its very conception, and still inform its current development.

Although the graphical interfaces of the PatchWork (Laurson & Duthen, 1989), OpenMusic (Assayag, Rueda, Laurson, Agon, & Delerue, 1999) and PWGL (Laurson & Kuuskankare, 2002) look quite similar to that of Max, their programming paradigm is deeply different, as the entry of the 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 from the conceptual and the practical point of view. Actually, 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 keep track of a stream of incoming events properly, be they a sequence of keys pressed by a player or a series of instructions guiding the composer's thought.

As a final note on this subject, it should be remarked that the differences in the user-side behaviours of these paradigms actually reflect totally different underlying structures, and the rift between them is far deeper than it might appear. In this sense, *bach* is essentially and structurally different from all the major existing CAC environments (see Figure 1 for a

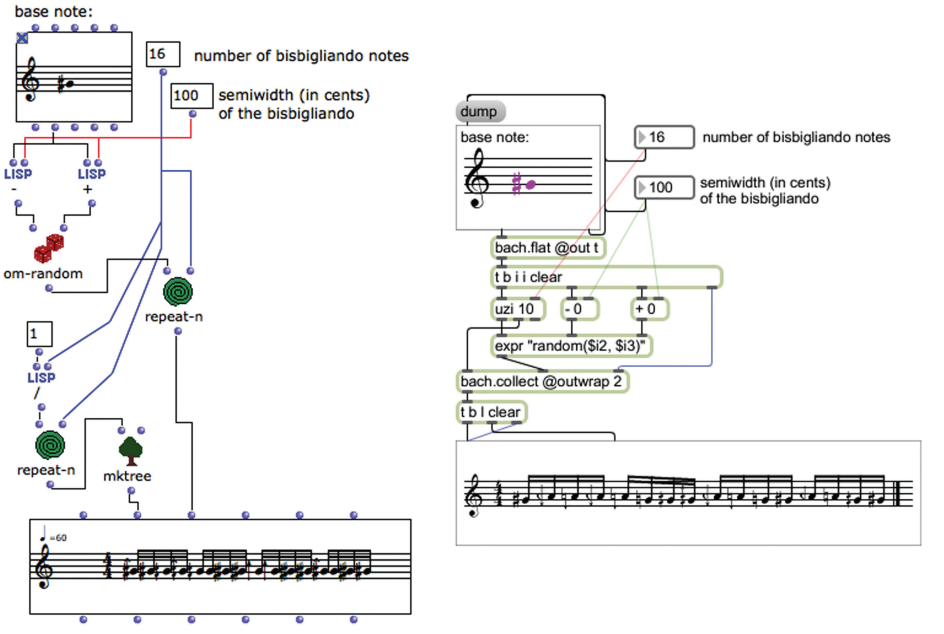


**Figure 1** A Comparison between an OpenMusic Patch (left) and a *bach* Patch (right) Performing the Same Process (Creation of a *bisbigliando* Around a Base Note). In the OpenMusic Patch, the Resulting Score is Left Untouched Until it is Re-evaluated. In the *bach* Paradigm, as One Changes a Parameter, the Result is Updated.

comparison example). The consequences of this real-time approach to CAC, from a musician's point of view, will be further explored in the following section of this article.

## Performative and Speculative Aspects

If the composer's interface choices affect the symbolic result in real-time, 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 when they compose electronic music. 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. In this sense, *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. It is worth noticing that what we call here the 'speculative' aspect of music writing is what Miller Puckette (Puckette, 2004) refers to as the 'compositional' aspect. Since we believe that compositional aspects are not necessarily in counterposition with performative ones, we prefer to use the 'speculative' adjective to dispel all doubts.

At the same time, as discussed above, *bach* is a citizen of the Max environment, and as such it can be very easily embedded into systems whose inputs and outputs are different from the traditional computer console. For instance, data coming from a MIDI device, such as almost any modern electronic musical instrument, can be processed by taking advantage of *bach*'s whole set of advanced musical representation facilities, and transformed into audio in real-time. Or the results of a real-time audio analysis can be filtered, processed and displayed as notes on a staff, perhaps using different colors and shapes for the note heads to represent different parameters of the analysis. If this approach is taken to the extreme, *bach* can be used as a tool for 'computer-aided improvisation', either as a preliminary phase to the actual writing or as an autonomous form of musical expression.

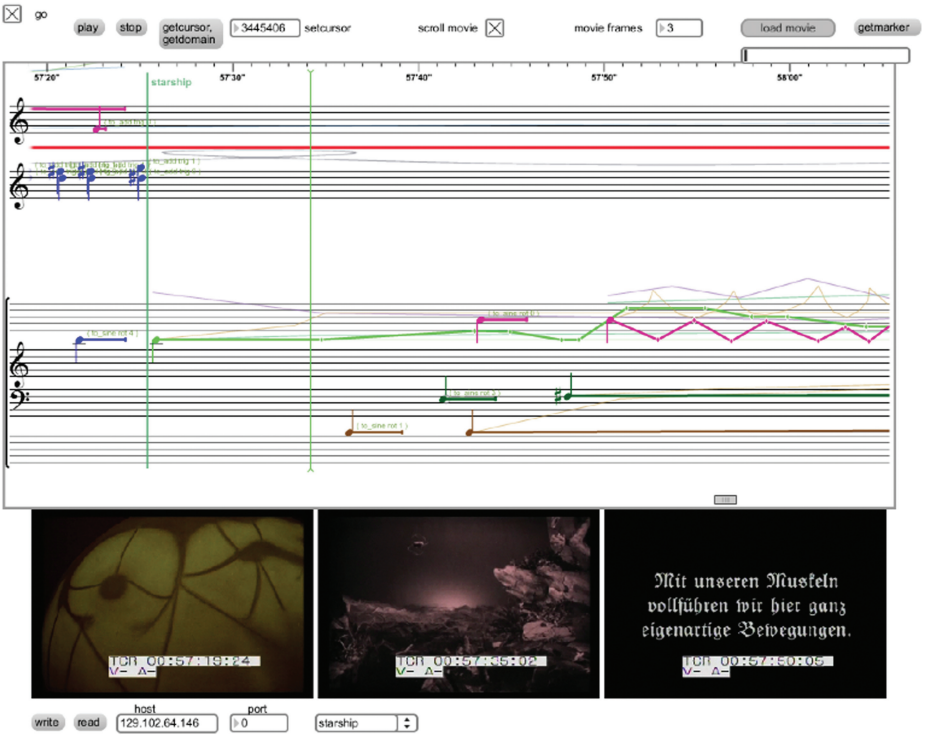


**Figure 2** A Patch Used by Andrea Agostini to Manage the Electronic Score for a Film Music. Each Note Contains a Set of Instructions for a Synthesizer, Expressed as Text Commands and Graphical Marks. When the Score is Played, All the Information Connected to Each Note is Sent to the Appropriate Synthesizer Voice, Represented on Screen 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 Fine Control Over the Sound and Image Synchronization.

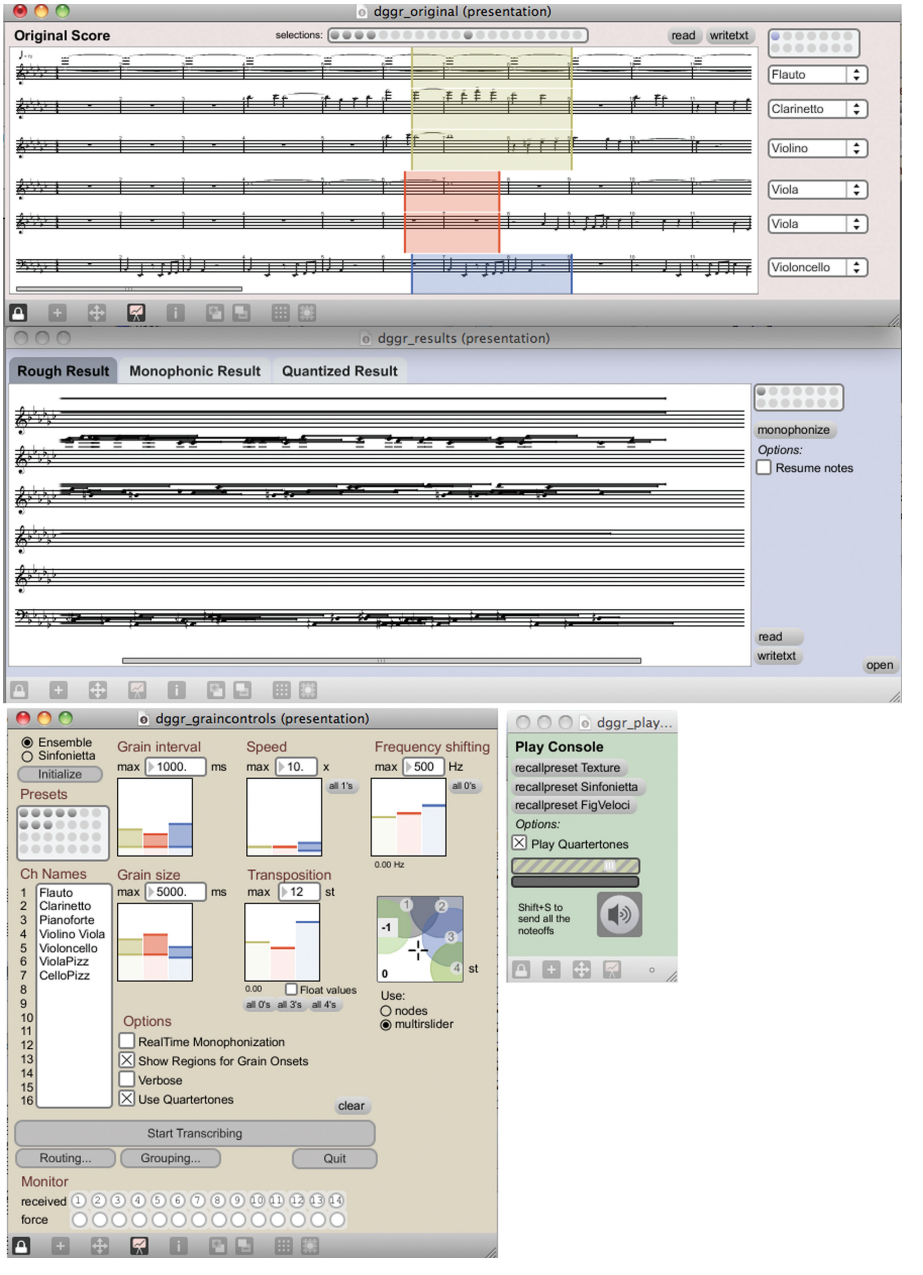


**Figure 3** A Patch, Used by Daniele Ghisi, to Achieve Real-Time Symbolic Granulation. The Original Score (Upper Window) has Some Markers to Determine and Modify the Grain Regions. Parameters are Handled in the Lower Left Window. When the User Clicks the 'Start Transcribing' Button, the Result Appears and Accumulates in the Middle 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 'Rough' Result, as in Any Electroacoustic Granulation Machine.

On the other hand, it is worth underlining 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 'refresh' operation (as would be the case in non-real-time environments). This means that, in principle, nothing prevents the user from using *bach* as 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 disrupt the immediacy of response to the user's actions.

## Some Examples

We give some screenshot examples of how the performative and speculative aspects can be unified, convinced that this might show, better than any words, the possibilities

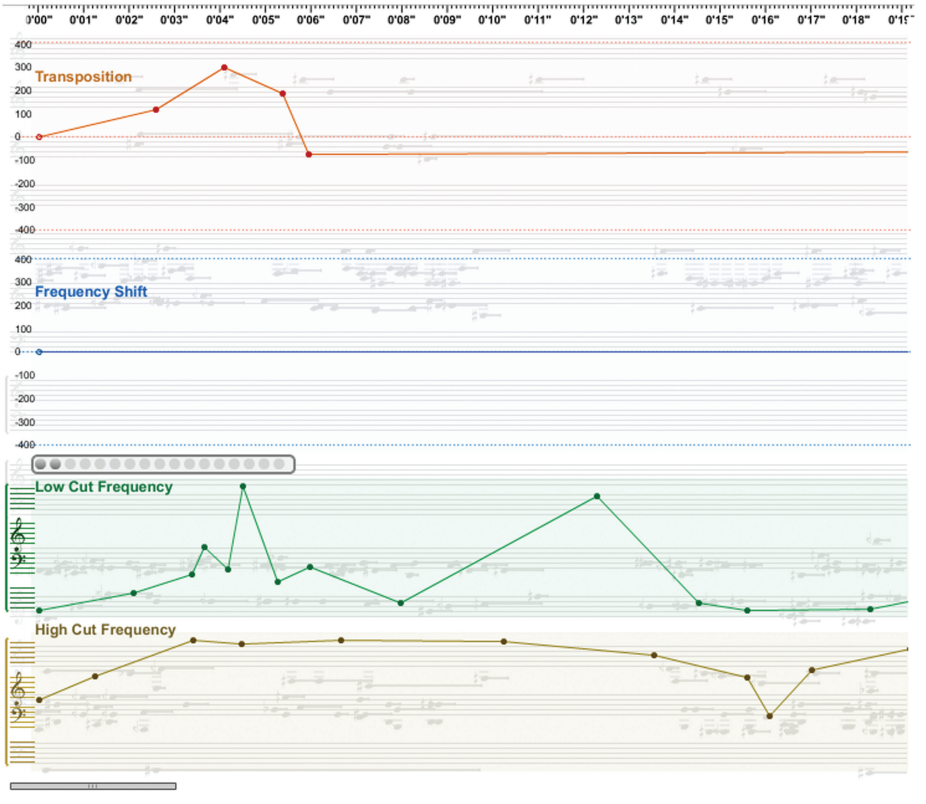


**Figure 4** A Patch Performing Real-Time Symbolic Transposition, Frequency Shift and Filtering. As the User Changes One of the Envelopes, the Underlying Score is Updated with the New Filtered Values (for Instance, Notice that as the Low-Cut Frequency Increases, the Notes Gets More and More Rarefied). The Patch is Obtained by a Close Interaction of *bach* Objects (such as the Underlying Score) and Standard Max Objects (such as the Breakpoint Functions Used for the Envelopes).

of the new model. The first example (Figure 2) shows that the boundary between sequencers and scores is no longer rigid: a score can be a customizable sequencer, whose content is completely open to any real-time process the user might want to realize. Notes carry extra information, specifying the parameters for the processes which will concern them. 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.

In the second example (Figure 3), we set up mechanisms to apply granulation (a typical electroacoustic treatment) to symbolic data. An original score is used as a reading buffer, where granulation regions are defined; the result is immediately visible in a second, constantly growing score, and is affected in real-time by any parameter change. In the last example (Figure 4), we set up a system to have counterparts to typical audio-domain transformations, such as transposition, frequency shifting and filtering, applied in real-time to symbolic data.



## Acknowledgements

We are deeply grateful to the following people for their precious support and advice: Carlos Agon, Arshia Cont, Eric Daubresse, Emmanuel Jourdan, Serge Lemouton, Jean Lochard and Mikhail Malt. We also wish to thank DaFact for actively sponsoring the development of *bach*.

## References

Assayag, G., Rueda, C., Laurson, M., Agon, C., & Delerue, O. (1999). Computer assisted composition at Ircam: From PatchWork to OpenMusic. *Computer Music Journal*, *23*(3), 59–72.

Cont, A. (2008). Modeling musical anticipation: From the time of music to the music of time (PhD thesis in Acoustics, Signal Processing, and Computer Science Applied to Music (ATIAM), University of Paris 6 (UPMC), and University of California San Diego (UCSD) (joint), Paris, 2008).

Laurson, M., & Duthen, J. (1989). Patchwork, a graphical language in preform. *Proceedings of the international computer music conference* (pp. 172–175). Ann Arbor: International Computer Music Association.

Laurson, M., & Kuuskankare, M. (2002). *PWGL: A novel visual language based on common lisp, CLOS and OpenGL*. Proceedings of international computer music conference, Gothenburg, Sweden, pp. 142–145.

Puckette, M. (2004). *A divide between 'compositional' and 'performative' aspects of Pd.* First internation Pd convention, Graz, Austria.

Seleborg, C. (2004). *Interaction temps-réel/temps différé*. Marseille: mémoire ATIAM.