Integrated Algorithmic Composition

Fluid systems for including notation in music composition cycle

Andrea Valle
CIRMA, Università di Torino
via Sant'Ottavio, 20 - 10124
Torino, Italy
andrea.valle@unito.it

ABSTRACT

This paper describes a new algorithmic approach to instrumental musical composition that will allow composers to explore in a flexible way algorithmic solutions for different compositional tasks. Even though the use of computational tools is a well established practice in contemporary instrumental composition, the notation of such compositions is still substantially a labour intensive process for the composer. Integrated Algorithmic Composition (IAC) uses a fluid system architecture where algorithmic generation of notation is an integral part of the composition process.

Keywords

Algorithmic composition, automatic notation

1. INTRODUCTION

Algorithmic composition can be defined as a composition practice that employs formalized procedures (algorithms) for the generation of the representation of a musical piece. Apart from the many *ante litteram* examples, algorithmic musical composition has been proposed and practiced widely starting from the '50s. In particular, from the late '50s a computational perspective started spreading across the two Western continents (see [1] for a detailed discussion). An interesting shift in perspective has occurred roughly from the '60s up to present day. The first approaches to algorithmic composition were driven by instrumental scoring. But, even if computer tools are largely widespread in contemporary instrumental scoring through computer-assisted composition systems (hence on CAC, e.g. PatchWork, Open Music, [2], PWGL [9], but also Common Music, [11]), the idea of a purely algorithmic approach, in which a strict formalization rules the whole composition process, is no more pursued in its integrity and has migrated from the instrumental domain to the electroacoustic one. In fact, considering the final output of the composition process, while in the electroacoustic domain the synthesis of the audio signal is a trivial task per se, in the instrumental domain the generation of musical notation still remains a very difficult task ([4], [10]). This notational issue has prevented the diffusion of real algorithmic practice for instrumental composition. Such an approach, in which the composition process is turned into a completely algorithmic workflow –from the first idea to the final score–, can be defined as Integrated Algorithmic Composition (IAC). An IAC approach pursues the integration of notation generation with musical data manipulation, so that any manual process could be removed from the composition pipeline.

The paper is organized as follows: first IAC/CAC approaches are discussed in relation to different software architectures; then, the need for a specific architecture is motivated in relation to automatic generation of music notation; finally, two cases are presented.

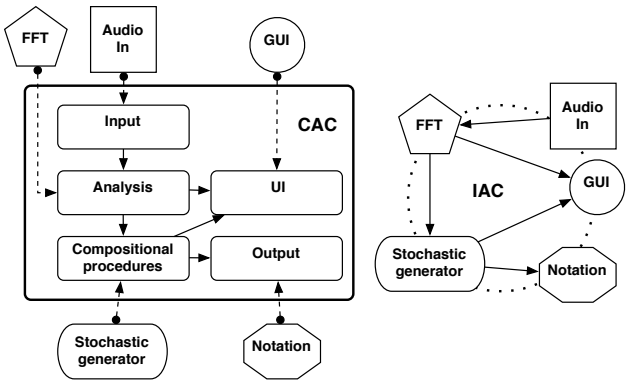

Figure 1: Rackbox v.s fluid architecture.

2. RACKBOX VS. GLUE ACHITECTURES

CAC systems are intended to aid the composer in the computational manipulation of musical data: these data, at the end, can be exploited in traditional score writing. Typically based on Lisp, CAC systems offer a large body of functionalities: pitch/rhythm operations remain the core of the system, with the inclusion of input modules for audio analysis and sound synthesis modules in output. All this functionalities are typically accessible through a GUI environment. While GUI is the main interface to the system, offering an easier access for the less programming-oriented composer, a high degree of flexibility is offered by enabling the user to extend the program via the Lisp language. Still, CAC architectures are based on the assumption that new functionalities must in some way be adapted to the hosting environment. A CAC application architecture can be thought as a rackbox containing a certain number of modules (Figure 1, left): the box can leave large room for other modules to be inserted in it. Still the container is solid and consequently rigid, its capacity is finite, and the modules, in order to be inserted, must meet the requirements of the box geometry. By reversing the perspective, a different approach to computer-based composition environments can

be conceived. It can be noted that each of the elements of the diagram in Figure 1 (left) may be replaced by an undetermined plethora of components. As a consequence, instead of starting from a solid framework where to insert modules, it is possible to start from an indefinite variety of available modules to be plunged –when necessary– into an open environment (Figure 1, right). Such an environment is fluid because it is intended as a glue capable of attaching different modules together. This reversed perspective implies a different approach by the user. By definition, a fluid system such as the discussed one cannot be implemented in a closed application. On the contrary, a programming language is the most flexible way to glue together different software modules. Thus, in a fluid system, the communication among the modules can be operated by a gluing language: the language writes scripts addressing each module's specific interface and executes them by calling the related program from OS. The language is responsible for symbolic manipulation representing the selected music features and for the communication with the modules in input/output. Not every programming language is suitable for such a gluing task. Requirements can be summarized as follows: high-level, dynamic typing, richness in dynamic data types, interactivity, string processing, interfacing to many system calls and libraries. The first two requirements are needed to let the composer concentrate on composition algorithms and not to deal e.g. with compilation or complex project structure issues, and to allow for continuous feedback while experimenting in composition. The last two are necessary in order to ensure the gluing mechanism. The requirements for softwares modules are in fact programmability and command line interfacing. Indeed, there is a strong coupling between CAC systems and rackbox architectures: CAC applications are oriented toward composers interested in a computational approach to symbol manipulations but are not programmers and are not interested in automatic composition. On the other side, an IAC approach, i.e. a fully-integrated computational approach to music composition including notation, can evidently benefit from the flexibility of a fluid system. More specifically, the case of automatic notation generation demonstrates that a completely algorithmic approach to composition can be achieved only through a fluid architecture.

3. AUTOMATIC MUSICAL NOTATION

Algorithmic composition requires to define a mapping from data structures (the output of composition algorithms) to a subset of notation symbols (the final output of instrumental composition practice). In the "classic" approach to algorithmic composition (Figure 2), this sensitive step is performed in first person by the composer, whose work defines the two extremes of the workflow: s/he provides composition parameter in input and defines algorithms; from a certain data structure the computer generates a data list in textual form; the composer controls the adequateness of the output, eventually modifying his/her composition strategy; then, s/he proceeds to transcribe the data list in musical notation. Finally, s/he evaluates the notational result, eventually modifying some steps of the process. To summarize, with respect to the computer output, the composer works both as a compositional/notational controller and as a transcriber.

The composition cycle thus requires two controls which are iterated in two different moments. Before the transcription, the composer can evaluate the generated data and foresee specific issues that would raise up during transcription. After the transcription, the same control can be carried out

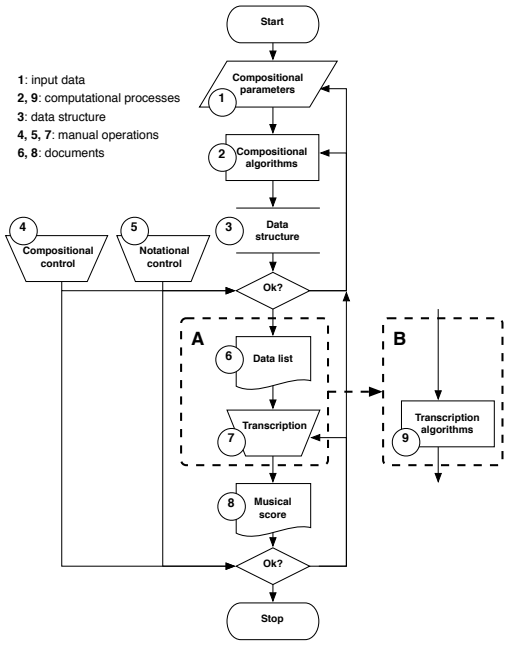

Figure 2: Algorithmic composition cycle.

by looking at the resulting notation. The control processes are high level musical tasks, and potentially they can be performed very quickly, even in terms of the few seconds necessary to have a glance at the resulting notes. On the other hand, the transcription step is a low level musical task, which is always very time-consuming –its timescale being typically measurable in hours. This slowness depends both on the complexity of notation in se and on the difficulty to clearly anticipate from a data list the peculiarity of the resulting notation. The crucial move towards an IAC approach consequently requires to automatize the transcription step (Figure 2, from A to B). In this way, the composer can focus on the higher level aspects of control: this would speed up the composition cycle (1-8) which can be executed until a satisfying result is obtained, thus leading to an interactive, trial-and-error methodology. The task of automatically generating musical notation is a privileged example of the power and, at the same time, of the necessity of a glue-connected, fluid system. CAC systems proposes generic transcription algorithms, intended to provide a draft of a possible notational output. Consequently, even if the resulting scores can be quite sophisticated, they are still drafts to be reworked manually by the composer. In an IAC approach, this handmade notational work is reversed into the definition of an algorithmic procedure for the control of a notation module. It must be stated clearly that music notation cannot be derived exclusively from musical data structures because notation information involves graphic data which are autonomous from musical data, but are important at the same degree for the final composition output: in short, music notation is not only music representation ([7]), and the composer must take into account both. More, in real practice composition and notation are related by a feedback loop, so that any decision on one side has always to be verified on the other one. A fluid architecture is indeed needed to automate such a task, so that transcription modules can be opportunely defined and fine-tuned. More, specific modules can be "plunged into the fluid" to meet the requirements of different notations: as an example, in the case of graphic notation, a drawing module may fit better that a musical notation one, even if the latter is

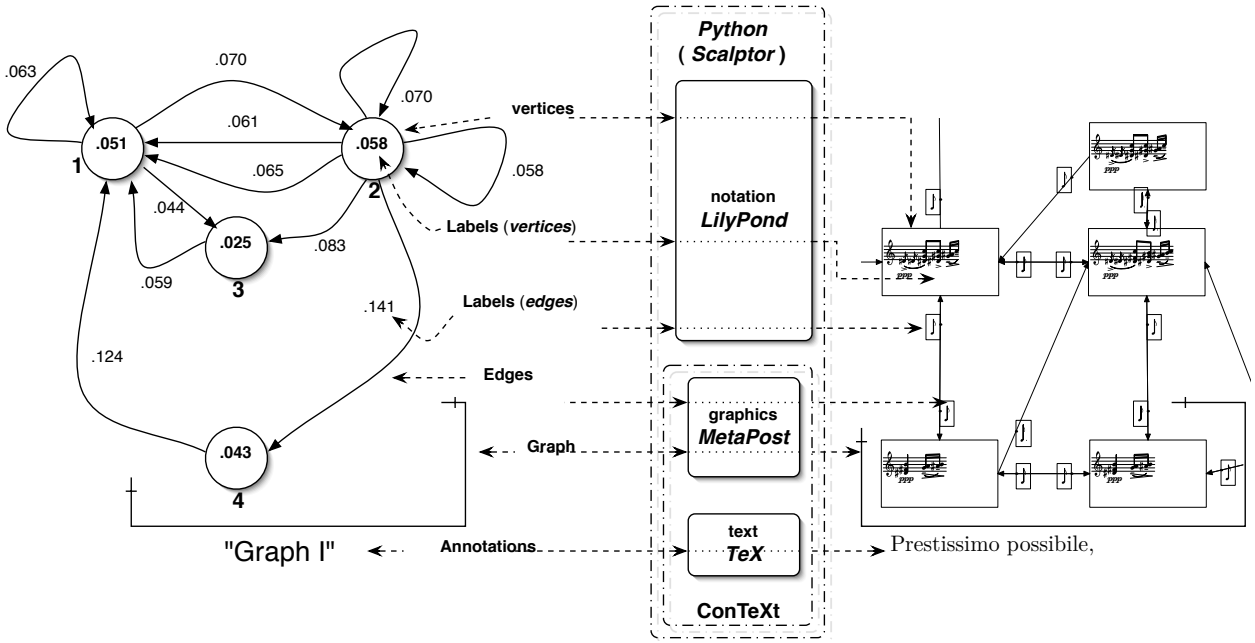

Figure 3: A graph model is fed into Scalptor, gluing LilyPond and ConTEXt to generate a graphic notation.

provided with some drawing capabilities. Examples of fluid architectures implementing IAC systems are described in [13] (where they are referred as Automatic Notation Generators), [5], and [3]. In the rest of the paper, we discuss two cases of IAC where different fluid systems are designed to fit different needs, allowing for a complete algorithmic control over the final score.

4. GRAPHIC NOTATION

In the first project, the final scores (for piano solo) is composed of a page in very large format (A0) containing graphical notation. The formal composition model is a graph and the notation mirrors visually the graph structure (Figure 3). All information associated to the graph data structure in the model has to be mapped into music/notation information, so that notation can be generated automatically. The score is made up of musical notation (vertices and edge labels), graphics (graph drawing), text (performing annotations) (Figure 3, right): all these components must be provided by programmable modules and their output integrated in an unique document. A strong constraint is that musical tradition requires high typographic quality both for the overall document and for the specific musical notation elements. As all the involved components are alphabetic or geometric, vector graphic solutions are consequently needed. In generale, as standard GUI applications are here not relevant, the possible candidates shares a TEX-based approach ([8]), i.e. they are command languages, to be input via textual interface and to be compiled in order to generate a vectorial output. Concerning musical notation, among the possible candidates (for a review see [10]), LilyPond, while still sharing a TEX-oriented approach, ensures very high typesetting quality but on the same time can be tailored for advanced uses, has a simple, human-readable syntax, it has undergone a fast development and it is now the most common text-based music notation application. LilyPond scripting solves the problems of generating standard notation for the vertices of the graph, but the resulting files (one for each vertex, in pdf/ps format) must then be included into the drawing of the graphic notation. It is interesting to see that many candidates are fitting in this case. LATEX and ConTEXt are two typesetting systems for document preparation implemented as a set of TEX macros. Both allow to work together with advanced graphic packages. ConTEXt ([6]) has been chosen as it provides direct support for the MetaPost graphic language and extends it by adding a superset of macros (named "Metafun") explicitly oriented towards design drawing (e.g. allowing pdf inclusion). For this particular project, Python has been chosen as the gluing language: it has a remarkably clear syntax and meets all the previously discussed requirements for an IAC language. Python takes into account all composition data processing, i.e. graph generation and manipulation algorithms, and also the gluing, scripting process. The Python module, named Scalptor ("engraver"), generates the score by writing text files containing code for each of the involved modules and calling each module in order to render it.

5. SPECTRAL COMPOSITION

As previously noted, an IAC system should provide room for inserting modules specialized in audio analysis. Analysis parameters can then be processed and used as starting material for musical composition. Figure 4 represents an implementation of an IAC fluid system for a composition project involving parameter extraction from audio signals. In particular, the commission was to use as starting material an excerpt from Sophocles' *Antigon*, which was read by a philologist so to respect as possible the reconstructed Greek classic pronunciation. Three voices sing melodies generated from data resulting from the analysis of the original audio file, in particular from the fundamental frequency and the first two formants. The Praat software has been chosen for the analysis task, being it specialized in phonetic processing. The SuperCollider application ([12], hence on SC) has been chosen both as system glue and as an audio module: as a language, SuperCollider is rich in data structure, highly expressive, provides an interfaces to the OS environment, allows for string manipulation; as an audio server, it provides state-of-the-art sound processing. Most importantly, from a UI perspective, SC allows for interactive sessions and provides also programmable GUIs. Com-

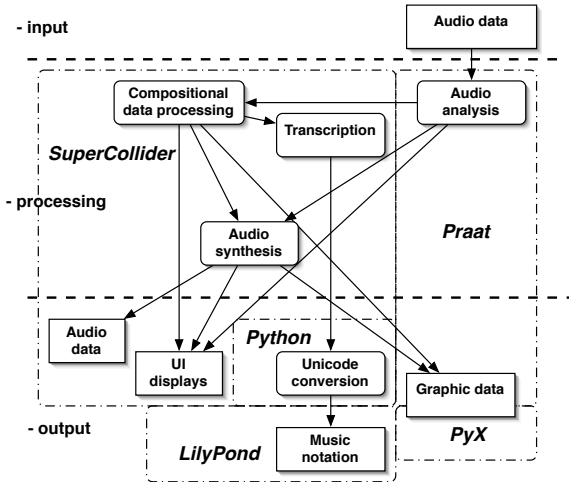

Figure 4: From audio to notation: modules.

munications between SC and Praat has been carried out via text files: Praat can be easily scripted by passing text files and it can, in turn, export text files, which can be read back from inside SuperCollider. The whole composition cycle can then be executed interactively from inside SC. As before, a transcription module is responsible for the generation of LilyPond files which can then be rendered to final pdf score file. The transcription algorithm also performs a melodic contour evaluation on the input data, so that continuous pitch increases/decreases are converted into ascending/descending glissandos (see Figure 5, bottom). For each note of a voice, a vowel symbol is assigned, as a result of a global evaluation of the two formants. As SuperCollider actually does not support Unicode, the LilyPond file has been post-processed by a Python module replacing special ASCII strings sequences with necessary Unicode glyphs. SC provides facilities to sonify in real time all the data, i.e. before and after processing and, through GUI packages, the same data can be displayed on screen. For purposes of documentation, high quality, vector graphics has been generated by writing in SC the opportune modules. Such modules allow to interface Praat, which is able to create graphics from all its data, and the PyX Python graphics package, which has been used to plot compositonal data structure. Figure 5 shows (from top to bottom) a GUI from SC plotting formant data, the same data exported by Praat into an eps file, and an excerpt from the final score by LilyPond. This rich system output provides a constant feedback to the composer, allowing her/him to control interactively the composition process.

6. CONCLUSIONS

The case of musical notation is particularly relevant in demonstrating the need (and the strength) of a fluid architecture for an integrated algorithmic composition system. In itself, notation is not a simple mapping from musical data to notation symbols, as it requires the composer to provide specific typographic information. An IAC approach allows the composer to develop case-specific solutions to such a problem, by plunging the selected modules into a fluid system: indeed, these can include not only notation but e.g. many different UIs. The maximum flexibility is evidently gained by using an interactive language as a system glue.
Some examples of automatic generated notation can be found at
`http://www.cirma.unito.it/andrea/compositionNotation/`.

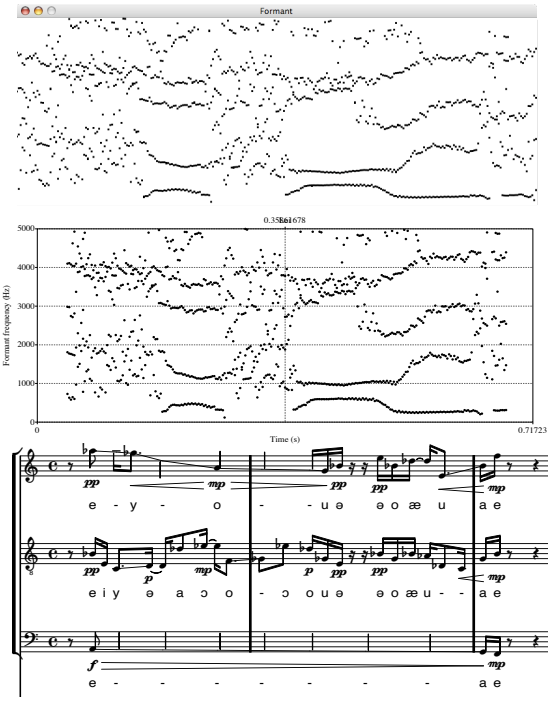

Figure 5: Different outputs. SC GUI, Praat graphics, LilyPond notation.

7. REFERENCES

[1] C. Ames. Automated composition in retrospect: 1956-1986. *Leonardo*, 20(2):169–185, 1987.

[2] G. Assayag, C. Rueda, M. Laurson, C. Agon, and O. Delerue. Computer-assisted composition at IRCAM: From PatchWork to OpenMusic. *Computer Music Journal*, 23(3):59–72, 1999.

[3] T. Baça. Re: Lilypond for serial music? LilyPond mailing list (lilypond-user@gnu.org), Nov. 28 2007.

[4] D. Byrd. Music notation software and intelligence. *Computer Music Journal*, 18(1):17–20, 1994.

[5] N. Didkovsky. Java Music Specification Language, v103 update. In *Proceedings of the International Computer Music Conference 2004*, Miami, 2004.

[6] H. Hagen. *ConTeXt the manual*. PRAGMA Advanced Document Engineering, Hasselt NL, 2001.

[7] K. H. Hamel. A design for music editing and printing software based on notational syntax. *Perspectives of New Music*, 27(1):70–83, 1989.

[8] D. Knuth. *The TeXbook*. Addison Wesley, Reading, Mass., 1984.

[9] M. Laurson, V. Norilo, and M. Kuuskankare. PWGLSynth: A visual synthesis language for virtual instrument design and control. *Computer Music Journal*, 29(3):29–41, 2005.

[10] H.-W. Nienhuys and J. Nieuwenhuizen. LilyPond, a system for music engraving. In *Proceeding of the XIV CIM 2003*, pages 167–172, Firenze, 2003.

[11] H. Taube. An introduction to Common Music. *Computer Music Journal*, 21(1):29–34, 1997.

[12] S. Wilson, D. Cottle, and N. Collins, editors. *The SuperCollider Book*. The MIT Press, Cambridge, Mass., 2008.

[13] H. Wulfson, G. D. Barrett, and M. Winter. Automatic notation generators. In *Proceedings of the 7th international conference on New interfaces for musical expression*, pages 346–351, New York, 2007. ACM.