GeoGraphy: a real-time, graph-based composition environment

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

ABSTRACT

This paper is about GeoGraphy, a graph-based system for the control of both musical composition and interactive performance and its implementation in a real-time, interactive application. The implementation includes a flexible user interface system.

Keywords

Musical algorithmic composition, composition/performance interfaces, live coding

1. INTRODUCTION

During the last fifty years, the use of computer for creating music has replicated a typical Western opposition between composition and performance. On one side, the computer allows an in-depth exploration of algorithmic composition techniques. Algorithmic composition is typically a non real-time process which strictly follows the two-step model of traditional composition, in which the composer creates a score to be played by the musician. On the other side, computers have been used for live performing from the late '60s, but it is only from the '90s that the increasing computational power of processors has consistently allowed the widespread of real-time audio softwares which can be used by the player to perform in a live concert: as a consequence, an "instrumental" use of audio software has grown, requiring the definition of interfaces allowing human-machine interaction, which have mainly taken the form of GUIs. Commercial software is based on various constrictive GUIs closing any access to the software apart from the built-in functionalities: more, they are typically focused at the textural level, thus making difficult the work at the note level, which is typical of the live instrumental performing ([4]). In order to get around the rigidity of musical software, two solutions have been devised: the development of new musical interfaces, escaping the limits of commercial/traditional ones ([4]), and live-coding, where the UI is the code window and the direct access to the chosen language allows for a fluid and flexible approach to the definition of composition algorithms as a performing act ([5]). Live coding emerges as a very innovative perspective, as it merges aspects of composition-based and instrumental-based approaches. Nevertheless, it has been noted that live coding may share some of the problematic issues of both. A general observation ([5]) is that the real-time, improvisational approach per se gains in instrumental expressivity but often lacks in structure: structure is what a non real-time approach to composition is aimed at, typically focusing on large-time scale definition and providing an overall architectural framework.

This paper describes GeoGraphy ([11]), a real-time environment for graph-based algorithmic composition, i.e a formal system for the algorithmic control of sound organization, and a real-time implementation that takes advantage of both GUI and live coding possibilities in terms of composition and interaction.

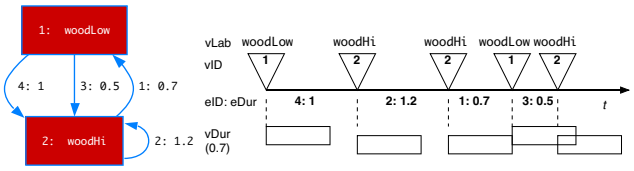

Figure 1: A graph (left), vertex durations and coordinates omitted) and a resulting sequence (right).

2. *GEOGRAPHY*: THE SEQUENCING MODEL

In the most general sense, GeoGraphy generates sequences of sound objects. The generation process relies on graphs. Graphs have proven to be powerful structure to describe musical structures ([7]): they have been widely used to model sequencing relation between musical elements belonging to a finite set. A common feature of all these graph representations is that they do not model temporal information: on the contrary, graphs as defined by GeoGraphy are intended to include it, so that time-stamped sequences of sound objects can be generated. The sequencing model is based on direct graphs (Figure 1) where each vertex represents a sound object and each edge represents a possible sequencing relation on pairs of sound objects. This direct graph is actually a multigraph, as it is possible to have more than one edge between two vertices; it can also include loops (see Figure 1 on vertex 2). Each vertex is given a label representing the sound object duration and each edge a label representing the temporal distance between the onset time of the two sound objects connected by the edge itself. The vertices are given an explicit position in terms of coordinates of a Euclidean dimensional space. This metric information allows to distinguish between two graphs being identical from a topological point of view, as they can have different positions in the space. Other information can be optionally associated to vertices and edges. The graph defines all the

possible sequencing relation between adjacent vertices. A sequence of sound objects is achieved through the insertion of dynamic elements into the graph, called "graph actants". A graph actant is initially associated with a vertex (that becomes the origin of a path); then the actant navigates the graph by following the directed edges according to some probability distribution. Each vertex emits a sound object as determined by the passage of a graph actant. Multiple independent graph actants can navigate a graph structure at the same time, thus producing more than one sequence. In case a graph contains loops, sequences can also be infinite. As modeled by the graph, the sound object's duration and the delay of attack time are independent: as a consequence, it is possible that sound objects are superposed[1]. A composition is a a set of sequences, like voices in polyphonic music: in a composition there are as many sequences as graph actants. The music generation process can be summarized as follows. Graph actants circulates on the graph: there are as many simultaneous sequences of sound objects as active graph actants. An example is provided in Figure 1. The graph (top) is defined by two vertices and four edges. The duration of both vertices is set to 0.7 seconds (it is not shown in the graph, only in the resulting sequence, Figure 1: bottom[2]). In Figure 1 (top), vertices are labeled with an identifier ("1", "2"). More, each vertex is given a string as an optional information ("woodLow", "woodHigh"): this string can represent a meaningful property of the referred sound object (e.g."wood" can stands for a woodblock audio samples), to be used in sound synthesis (see later). A composition starts when an actant begins to navigate the graph, thus generating a sequence. Figure 1 (bottom) represents a sequence obtained by inserting a graph actant on vertex 1. The actant activates vertex 1 ("woodLow"), then travels along edge 4 and after 1 seconds reaches vertex 2 ("woodHi"), activates it, chooses randomly the edge 2 between the available ones (edges 1 and 2), re-activates vertex 2 after 1.2 seconds (edge 2 is a loop), then chooses edges 1, and so on. While going from vertex 1 to vertex 2 by edge 3, vertex duration (0.7) is greater then edge duration (0.5) and sound objects overlap.

3. THE REAL-TIME COMPOSITION ENVIRONMENT

In the current implementation the challenge has been to use GeoGraphy as a real-time composition environment. In order to bring together these diverging issues, the SuperCollider application (SC) has been chosen among other possible candidates (e.g. Chuck[3], Impromptu [9]), as it features a high-level, object-oriented, interactive language together with a real-time, efficient audio server. The SuperCollider language summarizes features common to other general and audio-specific programming languages (e.g. respectively Smalltalk and Csound), but at the same time allows to generate programmatically complex GUIs ([12]). The whole system relies heavily on the Observer design pattern ([1]) for event handling. The pattern allows loose coupling between –to speak in SC– a "model" (the observed) and its "dependants" (the observers). The dependency mechanism is fundamental in allowing the maximum flexibility in the interaction with the system, as dependants, whatever their nature could be, can be interactively added or removed on-the-fly during a performance. Due to the interpreted, interactive nature of SC, the following discussion on the architecture, by introducing code examples, is at the same time a presentation of one of the possible ways of interacting with the system in a live situation. The overall architecture is represented in Figure 2: the components will be discussed progressively in the rest of the paper.

[1]This happens when the vertex label is longer than the chosen edge label.

[2]Vertex coordinates are omitted too.

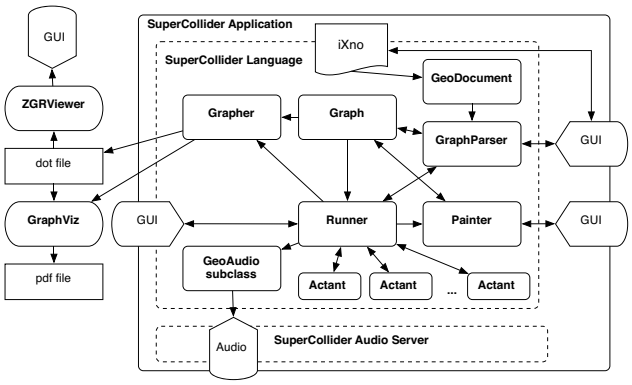

Figure 2: Overall architecture of the SC implementation.

The core of the system are the two classes Graph and Runner. The Graph is deputed to all (static) graph manipulations (adding/removing vertices/edges, etc). A graph stores information about its structure in a dictionary associating a vertex ID to the vertex definition: this includes coordinates, vertex duration, label, options and also the definition of all the edges starting from it. The graph in Figure 1 can be created by the following code, where a Graph instance `g` is created and some elements (vertices, edges) are added:

```
g = Graph.new ;
g.addVertex(x: 10, y: 20, dur:0.7, label: "woodLow") ;
g.addVertex(x: 10, y: 10, dur:0.7, label: "woodHi") ;
g.addEdge(start:2, end:1, dur:0.7) ;
g.addEdge(start:2, end:2, dur:1.2) ;
g.addEdge(start:1, end:2, dur:0.5) ;
g.addEdge(start:1, end:2, dur:1) ;
```

The Runner is a dependant of a Graph instance, i.e. each time a graph changes, its runner is updated. A runner is an interface towards instances of the Actant class (representing the graph actants) and manages all the parallel real-time, sequencing processes. An Actant instance is a wrapper for a routine which schedules events by traversing the graph and reading edge temporal information. The sequence depicted in Figure 1 can be generated by executing this code:

```
r = Runner.new(g) ;
r.addAndSetup(aStartingVertexID: 1) ;
r.start(aID:1) ;
```

First, a Runner instance `r` is created; then, an actant is placed on vertex 1; finally the actant is started. By instantiating an actant, the Runner becomes its dependant. Thus, each time a vertex is activated, the Actant instance sends a message to the dependent Runner. The runner adds to the message other information and, in turn, forwards it then to all its dependants. The whole message passing process can be exemplified by discussing audio synthesis (Figure 3).
It is apparent that GeoGraphy is intended as a general sequencing system, as it does not make any assumption about sound objects, whose generation is demanded to an external component. Properly, it defines a mechanism to generate sequences of *referred* sound objects (grouped in sequences). Audio synthesis can be achieved by subclassing the GeoAudio abstract class (see Figure 2). GeoAudio handles internally the relations with the system, while each subclass,

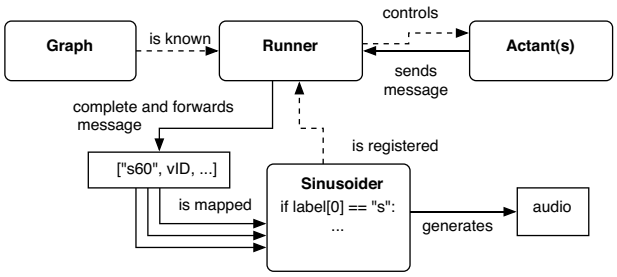

Figure 3: Dependency mechanism and message dispatching for a GeoAudio subclass ("Sinusoider").

to be provided by the user, requires the definition of three methods, respectively for initialization, synthesis definition, and mapping from data passed by the messages sent by the Runner. By subclassing GeoAudio, the user can create his/her own audio module library. The user can instantiate (and remove) in real time as many audio instances as desired (from different GeoAudio subclasses), as each of them will be registered to a runner and react to specific messages, like in a plug-in mechanism (Figure 2). Each time an actant activates a vertex, a message is sent from the runner to the audio device, which reacts by spawning an audio event. The audio mapping is left to the user but typically relies heavily on the vertex string label. From a user-centered perspective, to give a name to a sound object (e.g. "woodLow" and "woodHi") is much more meaningful than to assign a numerical identifier to a vertex.
The discussed architecture is crucial not only for decoupling audio synthesis from sequencing, but –more generally– for allowing multiple approaches to real-time interaction within the GeoGraphy environment: user interaction, in fact, can be accomplished through a three-layer interface system.

4. THE THREE-LAYER INTERFACE SYSTEM

As all the system is written in SC, the interaction can be accomplished by the same SC application, by using its textual interface to write and evaluate code in input, and to receive in output feedback from SC through its post window (Figure 4, no. 3). The previous section has, in fact, already demonstrated an interactive SC session with GeoGraphy. Also, the code textual interface is the most powerful way to control algorithmic processes over time, e.g. directly manipulating graphs or instantiating actants. As common in live coding practice, snippets of codes can be evaluated in real-time.

Even if a live coding approach can be extremely powerful, GUIs enhance real-time interaction by providing visualization and gestural control: as a consequence, GeoGraphy includes many GUI classes. All the GUIs are "throw-away user interfaces" ([8]), as they share the discussed dependency mechanism: thus they can be created/deleted in real-time without affecting the system state. As an example (Figure 4), the Runner can be given a GUI representing all the actants: in Figure 4, no. 1, there are four active actants traversing the graph. For each of them (labeled through the ActID field), the GUI provides a button to start/stop the generating process, plus a slider for continuous control of amplitude. Another fundamental GUI is provided by the Painter class, which is deputed to graph visual representation (Figure 4, no. 2): it draws the vertices with their ID and their strings, and the edges with their ID and durations. The Painter also provides information on running actants by visualizing activated vertices in red. The Painter works not only as a viewer (which is crucial in letting the user explore the graph structure), but also as a controller: vertices' positions can be modified by dragging their relative GUI elements. As the coordinates of the activated vertex are sent by the Runner to its dependants (GeoAudio included), they can be mapped to audio parameters: if so, by dragging the vertices in the Painter's space, the user can modify the graph metrics (i.e. the vertices' coordinates) and thus control gesturally the synthesis. A different GUI/code mixed approach is implemented in the visual interface provided by the Grapher class. Thanks to SC's operating system interfacing capabilities, the Grapher class can interface GeoGraphy with the GraphViz command line utilities for graph drawing (see Figure 2, [2]). The Grapher class creates, while working in real-time, a description of the graph in the dot language. This allows the user to explore interactively the graph with a specialized viewer such as ZGRViewer ([6]) or to render it to an image file. In this case, the result is Figure 1, left.
Even if mixed, code and GUI still remain in themselves two quite distinct interface layers. On the other hand, Taube has proposed for Commom Music a three layer interface system, based on "procedural, textual and gestural modes" ([10]). Common Music is written in Lisp and the procedural mode indicates the control of the system directly by programming in Lisp. Gestural mode refers in Common Music to its dedicated GUI system Capella. More, an intermediate layer between code and GUI is provided, by means of a "textual" interface: a list of commands is defined that can be interpreted by Stella, a shell-like interface. In GeoGraphy, an analogous approach has led to the definition of the "iXno" scripting language (*ichnos*, greek "trace"). The main purpose of iXno is to allow a simplified control of GeoGraphy for real-time usage. iXno commands take the form `c@ p1 ... pn`, where `c` is a single-letter identifier for a command, `@` can be replaced by `+` or `-`, and the following symbols represent command-specific parameters. The following line

```
e+ woodLow 0.4 woodHi 0.7 woodLow 1 woodHi 1.2 woodHi
```

contains the iXno command necessary to create the graph of Figure 1: `e+` is used to create edges by specifying a series of (vertex, edge duration, vertex) triples which can be concatenated (as in the example). iXno code is interpreted by the GraphParser class (Figure 2) and translated into SC code. iXno is intended as a fast language, to be typed in real-time: to overcome the slowness of typing, it favors terseness against clarity and it is based on the principle "the less you type the faster you play". More, many parameters receive a preset value and on the same line more commands can be concatenated. As iXno does not provide a full access to the SC classes, its behavior can be defined in terms of "live scripting", less powerful than SC but most immediate. The iXno language defines an intermediate layer between code and GUI and not by chance can be controlled by the user by two distinct interfaces. On one side, the commands can be written in a GUI textfield associated to the GraphParser (Figure 4, no. 4). Here, so to say, the user is placed at the GUI level. On the other side, to a more code-oriented interface, a special SC document class (GeoDocument) is available. GeoDocument provides a unified interactive code window interface for SC and iXno and benefits of all the text editing capabilities of the SC application (copy/paste, find, etc).
To sum up, the three layers of the interface system are in practice extremely permeable. Real-time control fully benefits from the intermingling of the different possibili-

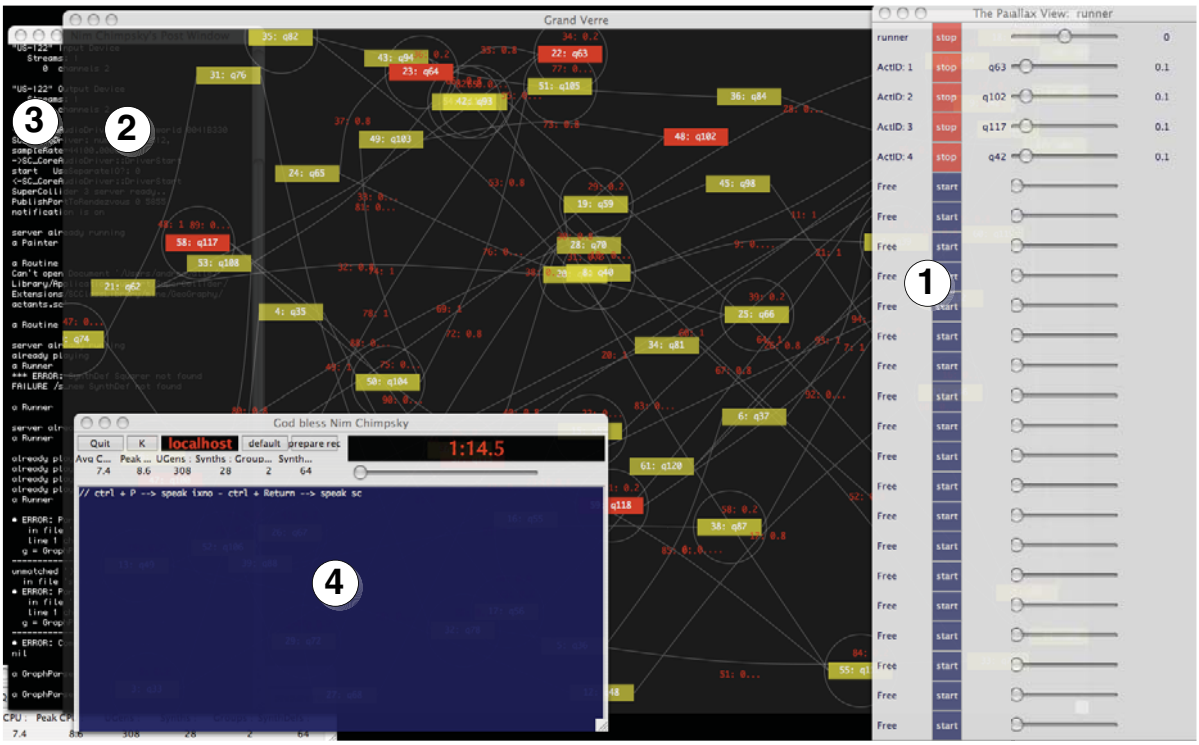

Figure 4: GeoGraphy GUIs. 1. Runner GUI, 2. Painter, 3. SC Post Window, 4. GraphParser GUI.

ties. Interactive graph manipulation can mix iXno scripting with GUI control through the Painter space. GUI creation/deletion can be scheduled by SC programming. As iXno is defined on top of SC Geography classes, it represents a simplified interface to SC classes which can be used directly in SC programming, by constructing strings and passing them to GraphParser instances. Through iXno, code snippet readability is highly improved with respect to the SC-only code. This is useful for live coding both in terms of typing speed and code management.

5. CONCLUSIONS

The real-time, interactive implementation of the GeoGraphy system aims at merging different attitudes towards composition. Real-time usage has required the development of different interfacing systems: GUI, code and script. This has been possible through a strictly modular architecture, which favors the insertion of modules specialized for different tasks. The resulting three-layer structure has proven to be useful in order to allow the user a maximum control flexibility, providing a smooth transition between the two extremes of code typing and GUI. In this way, it is possible for the musician to merge features typical of "out-of-time" algorithmic composition with a real-time control over performance.

GeoGraphy is distributed as a "quark, a public SC extension (see `http://quarks.sourceforge.net/`). See also `http://www.cirma.unito.it/andrea/notation/`.

6. ACKNOWLEDGMENTS

A thank is due to Vincenzo Lombardo for his continuous support.

7. REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.

[2] E. Gansner, E. Koutsofios, and S. North. *Drawing graphs with dot*, 2006.

[3] W. Ge and P. Cook. Chuck: A concurrent, on-the-fly audio programming language. In *Proceedings of the International Computer Music Conference (ICMC)*, Singapore, 2003.

[4] T. Magnusson. The ixiQuarks: Merging code and gui in one creative space. In *Proceedings of the International Computer Music Conference 2007, Copenhagen, August 27-31*, 2007.

[5] C. Nilson. Live coding practice. In *NIME '07: Proceedings of the 7th international conference on New interfaces for musical expression*, pages 112–117, New York, NY, USA, 2007. ACM.

[6] E. Pietriga. A toolkit for addressing hci issues in visual language environments. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 00:145–152, 2005.

[7] C. Roads. *The computer music tutorial*. The MIT Press, Cambridge, Mass., 1996.

[8] J. Rohrhuber, A. de Campo, R. Wieser, J.-K. van Kampen, E. Ho, and H. Hölzl. Purloined letters and distributed persons. In *Music in the Global Village Conference*, Budapest, December 2007.

[9] A. Sorensen. "impromptu: An interactive programming environment for composition and performance". In *Proceedings of the Australasian Computer Music Conference 2005*, pages 149–153. ACMA, 2005.

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

[11] A. Valle and V. Lombardo. A two-level method to control granular synthesis. In *Proceeding of the XIV CIM 2003*, pages 136–140, Firenze, 2003.

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