1. INTRODUCTION

Since the advent of computers there has been great interest on how to take advantage of their superior precision, speed and power in music-related activities. The probably best-known (and commercially successful) direct heuristic hebbian learning,” Nature, vol. 377, no. 6651, pp. 725–728, 1995.


are based on simplistic interactive setups that hinder the notion of interactivity. This fact does not degrade the artistic value of such works in any sense but underlies the lack of momentum therein for serious considerations of interactivity among the second group.

Of course, this dichotomy has already been addressed. 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\(^4\) [9] and InScore\(^5\) [7]. In at least one case, namely MaxScore\(^6\) [6], this is augmented by a very sophisticated editing interface. A more general approach is FTM’s [18], which provides a powerful framework for data representation and processing with a focus on musical structures, including some facilities for graphical display of simple scores.

Resuming the ideas of [2, 3], with the library bach:\textit{automatic composer’s helper} we have tried to achieve a coherent system explicitly designed for computer-assisted composition. bach takes advantage of Max’s facilities for sound processing, real-time interaction and graphical programming, combining interactive writing and algorithmic control of symbolic musical material.

2. PROGRAMMING PARADIGMS

bach complies with the graphical data-flow programming paradigm of Max, in which a kind of information is represented as a vertical, top-down flow. Data, typically coming from some user interaction, enter the program at its top, are acted upon by a chain of specialized operators connected by lines called ‘patch cords’ and exit the program at its bottom. A simplified model of this mechanism, as seen from a lower-level point of view, might appear as follows: each operator is a function, usually written in C or C++, and the data entering are the arguments of the function call. After performing its work upon the data it has received, each operator calls the function corresponding to the next operator in the chain, passing it the acted-upon data. In this way a call stack is built, in which the operator at the top of the graphical path corresponds to the function at the base of the stack, and the operator at the bottom of the graphical path corresponds to the function at the top of the stack. It is crucial to note that all these functions have no return value: the last operator of the chain simply passes the data to an arbitrary output device. In this way, the perception of the user’s side is that the program essentially behaves like a musical instrument, in which an action (e.g., pressing a piano key) triggers a sequence simply passes the data to an arbitrary output device.

At the top of the stack. It is crucial to note that all these functions have no return value: the last operator of the chain simply passes the data to an arbitrary output device. In this way, the perception of the user’s side is that the program essentially behaves like a musical instrument, in which an action (e.g., pressing a piano key) triggers a sequence simply passes the data to an arbitrary output device.

3. THE BACH ENVIRONMENT

As already stated, bach is a library of objects and patches for the software Max, the distinction between objects and patches concerning the more the implementation than the actual usage of these modules. At the forefront of the system are the bach\_score and bach\_roll objects. They both provide graphical interfaces for the representation of musical notation: 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 (in milliseconds), and as a consequence has no notion of traditional temporal concepts: this is useful for representing non-measured music, and also provides a simple way to deal with pitch markup whose temporal information is known or irrelevant. It should also be noted that the implementation of traditional temporality concepts in bach\_score is in fact quite advanced, as it allows multiple simultaneous time signatures, tempi and agogics. Besides this fundamental difference, the two objects offer a large set of common features, among which:

- editing by both mouse and keyboard interface, and by Max messages (see Fig. 1);
- support for microtonal accidents of arbitrary resolution (see Fig. 2);
- wide possibility of intervention on the graphical parameters of musical notation;
- ability to associate to each note various types of meta-data, including text, numbers, files and break-point functions (see Fig. 6);
- variable-speed playback capability: both bach\_score and bach\_roll can be seen 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.

3.1. Data types

bach also provides Max with two new data types: rational numbers and a nested list structure called \textit{illl}, an acronym for \textit{illl-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 nested list has been chosen for both similarity with the Lisp language, in a way to ease communication with the major existing CAC environment, and the need to establish a data structure powerful enough to represent the complexity of a musical score, but flexible enough to be a general data structure whose temporal information is known or irrelevant. The structure of a \textit{illl} represents a bach\_score (Fig. 4) might appear quite complex at first sight, but the or

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{Any notation object can be edited by both GUI interaction and Max messages. In this case we’re clearing the bach\_roll, and then adding two chords.}
\end{figure}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Semitonal, quartertonal and eighthtonal divisions are supported via the standard accidental symbols (upper example). All other microtonal divisions are supported as well, but will be replaced by labels with the explicit fractions of tone (lower example), or with cents differences from the diatonic note.}
\end{figure}
are based on simplistic interactive setups that hinder the notion of interactivity. This fact does not degrade the artistic value of such works in any sense but underlies the lack of momentum therein for serious considerations of interactivity among the second group.

Of course, this dichotomy has already been addressed. 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\textsuperscript{9} and InScore\textsuperscript{7}. In at least one case, namely MaxScore\textsuperscript{6}, this is augmented by a very sophisticated editing interface. A more general approach is FTMs\textsuperscript{10, 12}, which provides a powerful framework for data representation and processing with a focus on musical structures, including some facilities for graphical display of simple scores.

Resuming the ideas of [2, 3], with the library bach:\textsuperscript{4} automatic composer’s helper we have tried to achieve a coherent system explicitly designed for computer-assisted composition. bach takes advantage of Max’s facilities for sound processing, real-time interaction and graphical programming, combining interactive writing and algorithmic control of symbolic musical material.

2. PROGRAMMING PARADIGMS

bach complies with the graphical data-flow programming paradigm of Max, in which information is represented as a vertical, top-down flow. Data, typically coming from some user interaction, enter the program at its top, are acted upon by a chain of specialized operators connected by lines called ‘patch cords’ and exit the program at its bottom. A simplified model of this mechanism, as seen from a lower-level point of view, might appear as follows: each operator is a function, usually written in C or C++, and the data entering it are the arguments of the function call. After performing its work upon the data it has received, each call instructs the next operator in the chain, passing it the acted-upon data. In this way a call stack is built, in which the Lisp lists certainly are based.

The difference between the two paradigms is crucial: if we assume that parameters are handled at the beginning of the process, a bottom-up process (like within the Patchwork paradigm) will ultimately be a non-real-time process, since parameter changes cannot immediately affect anything below them, unless some bottom-up operation is requested on some lower elements. Moreover, the Max paradigm, not having to depend on return values, easily allow for much more complexly structured patches: a single expression can trigger multiple reactions in different operators (a function can call several other functions, one after another has returned). The Patchwork paradigm, on the other hand, has the advantage of allowing seamless integration with textual coding, which can be an extremely useful resource when conceptually complex operations must be implemented. Moreover, representing musical notation (from single notes to an entire score) requires sufficiently powerful and flexible data structures, which the Lisp lists certainly are.

3. THE BACH ENVIRONMENT

As already stated, bach is a library of objects and patches for the software Max, the distinction between objects and patches concerning more the implementation than the actual usage of these modules. At the forefront of the system are the bach.score and bach.roll objects. They both provide graphical interfaces for the representation of musical notation: 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 values (in milliseconds, for instance) and, as a consequence has no notion of traditional temporal concepts: this is useful for representing non-measured music, and also provides a simple way to deal with pitch markers whose temporal information is known or irrelevant. It should also be noted that the representation of traditional temporal concepts in bach.score is in fact quite advanced, as it allows multiple simultaneous time signatures, tempi and agoegos. Besides this fundamental difference, the two objects offer a large set of common features, among which:

- editing by both mouse and keyboard interface, and by Max messages (see Fig. 1);
- support for microtonal accidents of arbitrary resolution (see Fig. 2);
- wide possibility of intervention over the graphical parameters of musical notation;
- ability to associate to each note various types of meta-data, including text, numbers, files and break-point functions (see Fig. 6);
- variable-speed playback capability: both bach.score and bach.roll can be seen 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.

3.1. Data types

bach also provides Max with two new data types: rational numbers and a nested list structure called illi, 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 nested list has been chosen for both similarity with the Lisp language, in a way to ease communication with the major existing CAC environment, and the need to establish a data structure powerful enough to represent the complexity of a musical score, but flexible enough to be a general purpose container whose temporal information is known or irrelevant. The structure of an illi representing a bach.score (Fig. 4) might appear quite complex at first sight, but the or

\textsuperscript{9}http://303.xex.be
\textsuperscript{7}http://inscore.sourceforge.net
\textsuperscript{4}http://www.computermusicnotation.com

Figure 1. Any notation object can be edited by both GUI interaction and Max messages. In this case we’re clearing the bach.roll, and then adding two chords.

Figure 2. Semitonal, quartertonal and eighthtonal divisions are supported via the standard accidental symbols (upper example). All other microtonal divisions are supported as well, but will be replaced by labels with the explicit fractions of tone (lower example), or with cents differences from the diatonic note.
Moreover, most operations can be constrained only in some levels of depth. The structure of a simple score in Figure 3, containing information, articulations and meta-data. The meta-content contained in each note, appearing in the score starting with the slots symbol. The form (type, range, domain...) of each slot appears in the header, which has not been dumped.

As the goal of bach is allowing real-time interaction, a great amount of work has been spent to improve the stability and efficiency of the system. All the operations in bach are thread-safe in the context of the Max threading model, and the passing ofills between objects happens by reference, rather than by value, unless the user explicitly requests otherwise, which is the case whenever the contents of a ill need to be passed to a non-bach Max object (that only accepts data passed by value). Thus, Ills are copied only when strictly necessary, and in all the other cases a reference counting mechanism is used to ensure that the lifetime of data structures and the usage of memory are correctly managed. On the other hand, all this is transparent to the user, who never needs to cope with the cloning of Ills, or the distinction between destructive and non-destructive operations - as, on the contrary, it is often the case with Lisp.

Taking all this into account, it should be clear that bach is somewhere placed at the convergence of several categories of musical software. Its capabilities of graphical representation of musical scores typically belong to music engraving systems - although it should be noted that, in its current state, bach lacks some essential features of this kind of programs, first of all a page view. On the other hand, most of its features are conceived in order to make it a tool for Computer Aided Composition as powerful as the traditional LISP-based environments, and able to communicate with them. 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. Finally, it can of course lend itself to innovative applications exploiting the unique convergence of these different paradigms and its specific real-time behavior (such as the symbolic granulation example shown in Fig. 7).

4. FUTURE DEVELOPMENTS

At the time of writing, bach is in its alpha development phase: although the system is usable, not all the intended features have already been implemented. Some of the planned additions are:

- Support for rhythmic tree representation, which will allow, for example, nested tuplets to be represented, whereas now a triplet containing a quintuplet is represented as a flat 15-uplet. This feature is currently under development, together with an intuitive measure linear editing system for the note insertion. The underlying challenge is to keep the tree and linear representations of durations always compatible, so that users should concretely deal with the tree representation only when they explicitly ask to (e.g. when they insert as rhythm a nested rhythmic structure), or when they perform hierarchical operations (e.g. when they split a chord). Users will also be able to rebuild a default rhythmic tree from the linear representation at any moment.

- Implementation of hierarchical structures within a score, allowing the user to group elements by name, where an element can be a chord, a note, a marker, or another group.

- Support for import and export of MIDI, MusicXML and SDF files.

- A solver for constraint satisfaction problems.

Notice that the software development 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.

5. REFERENCES


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 its 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, articulations and meta-data. The illl represents a whole bach.roll has essentially the same structure.

3.4. Practical applications

Taking all this into account, it should be clear that bach is somehow placed at the convergence of several categories of musical software. Its capabilities of graphical representation of musical scores typically belong to music engraving systems - although it should be noted that, in its current state, bach lacks some essential features of this kind of programs, first of all a page view. On the other hand, most of its features are conceived in order to make it a tool for Computer Aided Composition as powerful as the traditional LISP-based environments, and able to communicate with them. 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. Finally, it can of course lend itself to innovative applications exploiting the unique convergence of these different paradigms and its specific real-time behavior (such as the symbolic granulation example shown in Fig. 7).

4. FUTURE DEVELOPMENTS

At the time of writing, bach is in its alpha development phase: although the system is usable, not all the intended features have already been implemented. Some of the planned additions are:

- Support for rhythmic tree representation, which will allow, for example, nested tuplets to be represented, whereas now a triplet containing a quintuplet is represented as a ‘flat’ 15-uplet. This feature is currently under development, together with an intuitive measure linear editing system for the note insertion. The underlying challenge is to keep the tree and linear representations of durations always compatible, so that users should completely deal with the tree representation only when they explicitly ask to (e.g. when they insert as rhythm a nested rhythmic structure), or when they perform hierarchical operations (e.g. when they split a chord). Users will also be able to rebuild a default rhythmic tree from the linear representation at any moment.

- Implementation of hierarchical structures within a score, allowing the user to group elements by name, where an element can be a chord, a note, a marker, or another group.

5. REFERENCES


- Support for import and export of MIDI, MusicXML and SDIF files.
- A solver for constraint satisfaction problems.

Notice that the software development 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.
In this article, we introduce OSC-NETLOGO, a tool that allows the creation of sonic phenomena by taking advantage of NetLogo’s power for designing and building models of complex systems. NetLogo is a multi-agent programming language and modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems that evolve dynamically over time. We provide two examples taken from NetLogo’s library of models. These examples provide evidence for the capabilities and potential of NetLogo as a sound synthesis tool. We hope that this tool could be of aid in future efforts of creating new complex sounds and interesting musical material.

1. INTRODUCTION

NetLogo is a multi-agent programming language and modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems evolving over time [4]. NetLogo comes from the Logo family of programming languages [3] and has expanded the original Logo concept in a number of ways.

NetLogo allows modelers to give instructions to hundreds or thousands of independent agents all operating concurrently, which is something essential for modeling complex systems. This makes it possible to explore connections between micro-level behaviors of individuals and macro-level patterns that emerge from their interactions. NetLogo enables users to open simulations and play with them, exploring their behavior under various conditions by manipulating several graphical objects such as sliders or buttons. NetLogo is also an authoring environment that enables users to create their own models, and try them on the fly.

According to the NetLogo website [8], NetLogo is being used to build an endless variety of simulations. Members of the NetLogo community have turned turtles into neutrons, magnets, planets, shepherds, lovers, ants, molecules, wolves, voters, passengers, metals, bacteria, cars, robots, neutrinos, magnets, planets, shepherds, lovers, ants, muscles, networkers, and more. Patches have been made into molecules, wolves, buyers, sellers, bees, tribespeople, birds, worms, voters, passengers, metals, bacteria, cars, robots, neutrinos, magnets, planets, shepherds, lovers, ants, muscles, networkers, and more. Patches have been made into trees, walls, terrain, waterways, housing, plant cells, cancer cells, farmland, sky, desks, fur, sand, etc. [4].

In the NetLogo interface, one may make it monophonic, retouch it, and finally quantize it. Every parameter is user-modifiable and affects the result in real-time, as in any electroacoustic granulation machine.

Turtle agents and patches are useful to visualize and study mathematical abstractions, specially non-linear dynamical systems, to display behavior in graphical ways, to make art and even play games. NetLogo comes with a very big library of models, which covers topics such as cellular automata, genetic algorithms, positive and negative feedback, evolution and genetic drift, population dynamics, networks, markets, chaos theory, swarming behavior, and molecular physics. All of these models share core concepts such as complex systems, self-organization and emergence.

Although NetLogo is very powerful for modeling and handling complex systems data, and provides some basic audio functionality through MIDI, it lacks more serious digital audio generation and processing capabilities. However, one good thing about NetLogo is that it provides an API for programmers to develop extensions to the program in Java. We took advantage of this feature and developed OSC-NETLOGO, an Open Sound Control (OSC) [9] extension to Netlogo using the JavaOSC library [2]. This extension allows users to directly map any parameter of a NetLogo patch into any sound processing environment such as Max/MSP, Pd or SuperCollider using OSC.

This article is structured as follows. In section 2 we describe the NetLogo application in more detail, including its history, API. In section 3 we describe the netlogo-osc extension, including its installation and usage. Then, in section 4 we provide examples of mappings of two NetLogo models into sound synthesis using Pd. Finally, in section 5 we discuss the main findings and conclusions of our work.

2. NETLOGO

NetLogo is a cross-platform standalone application written in Java. It has been being developed for more than ten years, which assures that NetLogo is a mature product that is stable and fast. It is freeware, anyone can download it for free and build models without restriction. It also comes with extensive documentation and tutorials and a large collection of sample models, created both by NetLogo developers and the general community of users [4].