In ICMC 2000, Berlin (Germany), ICMA.

Rhythms as emerging structures

François Pachet, pachet@csl.sony.fr

SONY CSL, 6 rue Amyot, 75005 Paris, France

Introduction

Rhythm has traditionally been considered a fundamental dimension of music. In the context of musical content feature extraction, in particular for music catalogues, the rhythmic dimension has up to now been curiously under studied. In particular, no operational taxonomy of musical rhythms is available, not to mention measures of similarity between rhythms. One reason is probably because that most of the approaches to rhythm modeling (e.g. Laine, 1999) are based on a view of rhythm as first class, objective data. These approaches are well suited to a score-based analysis, but are less useable for understanding the organic nature of rhythms.

We propose an evolutionary approach for modelling musical rhythm. In this model, rhythm is seen as a musical form, emerging from repeated interaction between several rhythmic agents. These agents engage into a dynamic game which simulates a group of human players playing, in real time, percussive instruments together, without any prior knowledge or information about the music to play, but the goal to produce coherent music together. Instead of specifying rhythms as first class objects, we shift the focus of attention and propose to specify the rhythmic agents, and their individual properties. The model allows to define various sorts of rhythmic agents, run simulations, and observe rhythms as the emerging forms for particular configurations of agents. In this paper, we describe the overall framework and some preliminary experiments.

Definition of rhythmic agents

The agents, once initialised, play together in real time in loop, in a synchronised fashion, i.e. with the same tempo, and the same cycle length. Each agent plays a different rhythm, and is endowed with a perception and an action module. A "rhythm" here is to be understood as a temporal sequence of midi notes, which can be mapped to arbitrary sounds (percussive or pitched).
The perception module takes as input the whole music played by all agents, in the form of the global sore of all agents in a symbolic form. The perception module applies basic parsing functions on this score, in order to extract information needed for the generation module, such as beat structure, or emphasis on beats.

The production module has the role of modifying the agent's rhythm, based on the perception module. This modification is specified by a set of transformation rules. The rules are generic, i.e. are not agent-dependent, and are organised in a rule library. Each agent is given a set of rules at the beginning of the play. The global rhythm is then the result of the ongoing play between these co-evoluting agents.
Besides, each agent holds a set of parameters defining its state, such as:

- A midi program change
- A midi channel and a midi port, to allow a maximum flexibility in Midi rendering.
- A tempo
- *Possible_pitches*: A set of pitches the agent "knows". This set is used e.g. to generate new notes.
- *Initial_rhythm*: an initial rhythm, expressed as a sequence of midi pitches. This is the initial rhythm the agent will use at the beginning of the session. The empty sequence is a possible starting sequence.
- A set of transformation rules (see below).

As a working example, let us consider a very simple popular rhythm: rock. Rock can be seen as a rhythm made up of two main agents: a drum bass agent and a snare agent. In the cannonical version of Rock, each agent plays the same rhythm: a periodic sequence of notes, with a period T, and each agent is desynchronized by T/2. It is the simplest binary rhythm on can think of, and is illustrated in Figure 1.
However, when drummers play – even bad ones - they never play exactly this "ideal" rhythm: they introduce errors (notes not exactly played on the beat), and variations, such as doubling the bass drum, or playing the snare a bit syncopated. The specification of these variations is precisely the subject of this work, and is achieved through transformation rules.

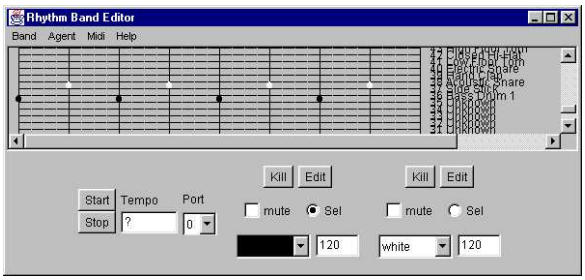


Figure 1. A simple two agent rhythm: a basic rock.

In ICMC 2000, Berlin (Germany), ICMA.

Transformation rules

The transformation rules are the core of our model. These rules are generic operators which apply on rhythmic melodies to produce variations of these melodies. Instead of using very specific rhythmic operators as proposed, e.g., in (Assayag and al., 1999), we have attempted to reduce the number of rules to generic rhythmic operators.
Rules are basically conditional procedures. More precisely, they take the following parameters:

- a condition, expressed as a predicate holding on the parameters of the agent, as well as the global score produced by all agents in the preceding iteration,
- an action, expressed as a modification of the agent's own rhythm.

These rules are fired at each cycle, once for each agent, and in anticipation, i.e. during the cycle. The effect of the rules is to create a new version of the agent's rhythm. The agent's rhythm is effectively updated with this new rhythm at the end of a cycle. All the agents perform this reasoning task in parallel.
Our experiments consist in identifying which rules may lead to which classes of rhythms. We give here several examples of situations and the corresponding rules defined for dealing with them.

Creating rhythms from scratch

The approach can be used to create rhythm from scratch, i.e. with agents having initially empty rhythms. Here are two examples. First, let us introduce metric related rules:
Emphasize_strong_beat
If (there is a strong beat during which agent does not play) Then add a pitch chosen at random among *possible_pitches*, at this strong beat.
Emphasize_weak_beat
If (there is a weak beat during which agent does not play) Then add a pitch chosen at random among *possible_pitches*, at this strong beat.

A possible representation of a basic rock created from scratch can be the following:

- a bass agent. This agent has one rule: **Emphasize_strong_beat**.
- a snare agent. This agent has one rule: **Emphasize_weak_beat**.

If agents have initially empty rhythms, and no other rules, the execution of the model converges quickly after 4 cycles (in the case of a 4 measure score) to the basic rock of Figure 1, and then remains obviously unchanged since no rule is applicable.

Another class of important rhythmic patterns is obtained through so-called "syncopations". A syncopation is basically the occurrence of a note "just before" a beat. This notion can be implemented as a rule as follows:

Syncopation
If (there is a beat during which an agent plays) Then add a pitch chosen at random among *possible_pitches*, just before this beat. "Just before" in our case may be fixed to some small division of a beat.

This rule may be illustrated with one agent having two rules: **Emphasize_strong_beat**, and **syncopation**. After 4 iterations, the following score (Figure 2) is obtained:

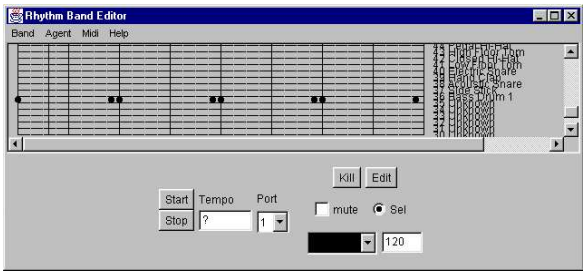


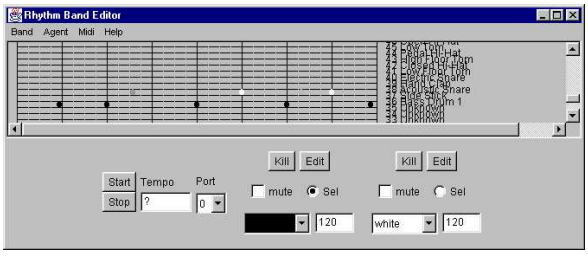
Figure 2. Creation of a syncopated bass drum.

Variation of existing rhythms

Variation can then be introduced by adding random rules to agents. Several kinds of randomness can be defined. Here are simple ones:

Add_Random:
If (true) Then add a pitch chosen at random among possible_pitches, at a time chosen at random within the cycle length.
Remove_Random:
If (rhythm not empty) Then remove a pitch chosen at random in rhythm.
Move_pitch:
If (rhythm not empty) Then translate in time a pitch chosen at random in rhythm.

If the two agents in the Rock example have now also rules *add_random* and *remove_random*, the following scores are generated after a few iterations:



In ICMC 2000, Berlin (Germany), ICMA.

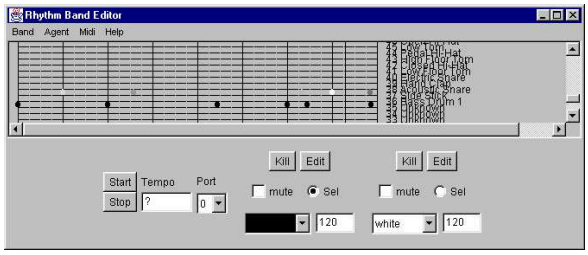


Figure 3. Several evolutions of the basic rock rhythm.

These scores sound more or less "rocky" in the sense that there is always a bass drum contribution on most of the strong beats, and a contribution of snares on weak beats.

Harmonic agents

The framework can also be used for producing harmonic agents, i.e. agents which evolve mostly vertically instead of horizontally. Experiments were done starting with a very simple "chord" agent, which uses a 4-note chord as a starting "rhythm", and one simple evolution rule, which consists in moving 1 or 2 half step higher or lower one of its four notes. Several such agents can then be put in the system to create harmonic progressions.

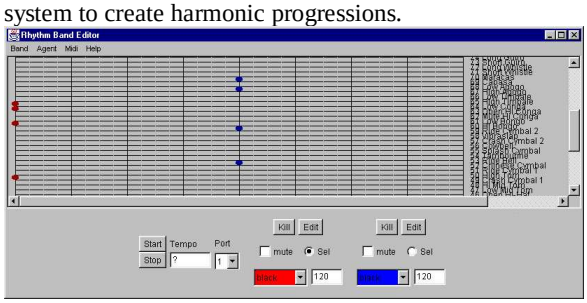


Figure 4. A chord sequence created by two harmonic agents.

Although the use of one simple context independent rule already produces interesting sequences, one can try to create more "constrained" harmonic progressions. To do so we further introduce rules that determine pitch changes according to the "next chord" in the global score. For instance, rules for attraction/repulsion can be defined as follows:

Attraction

If (distance between rhythm and the rhythm of the "next" agent in the band is < Threshold) then choose a pitch in rhythm and move it toward the corresponding pitch in the next rhythm.
Repulsion is defined similarly, with > instead of <.

By defining agents with attraction and repulsion rules, one can create infinite chord sequences which *never* converge, while still giving the impression of turning around some strange attractor.

By extension, any parameter of the chord can be used to be constrained with parameters of other chords or musical construct. For instance, the density, chord type, harmonic function, etc. can be used to create sequences having more natural musical structures, such as avoiding chords with too many dissonant intervals, avoiding paralell fifth, etc.

More complex structures

Finally, the whole set of possibilities can be used to create richer rhythms: chord agents mixed with rhythm agents. Figure 5 shows a set up with two harmonic agents and two rhythmic agents (one is set to play drum sounds, the other a Kalimba pitched melody).
For these configurations, it is interesting to introduce rules that actually connect chords to melodies. First, a simple rule of this kind detects an agglomeration of notes at a certain temporal position (with a possible approximation given by a threshold) and decides to emphasize it by playing a note (possibly a bass drum sound) at that time. This rule allows to let chord structures emerge from different melodic agents.
Conversely, a rule allows chord agents to adapt to melodic agents by changing one of their pitches to a pitch which is the most common in their immediate surrounding.

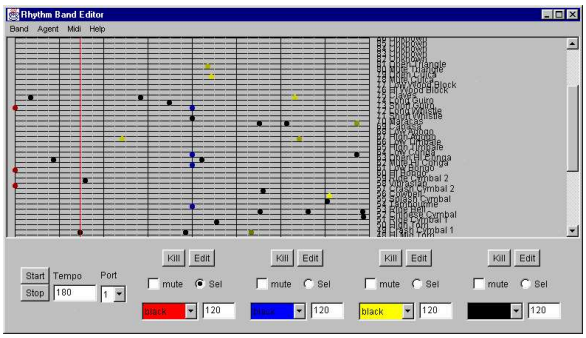


Figure 5 A setup with a mix of rhythmic and harmonic agents

Back to Rock

One can also start from an existing rhythm, e.g. in Midi format, and load it in the system. Each voice is then associated to an agent with initially no evolution rule. Rules can then be added to agents to let the rhythm evolve in various ways. Figure 6 shows two popular rhythms: a rock rhythm complete with crash and cymbal; and a Brazilian Batucada rhythm with 7 percussions. In these cases, the goal is to make the rhythm evolve without departing too much from the initial structure, and by avoiding to define ad hoc rules for this purpose. This is done by:

- Introducing various sorts of noise in agents (e.g. with rules add and remove random),

In ICMC 2000, Berlin (Germany), ICMA.

- Introducing variations through syncompations. Typically in the Batucada, where some instruments should never play on the beats, but always slightly ahead.
- Introducing attractors for rhythmic agents to strong or weak beats (as shown with the basic rock example)

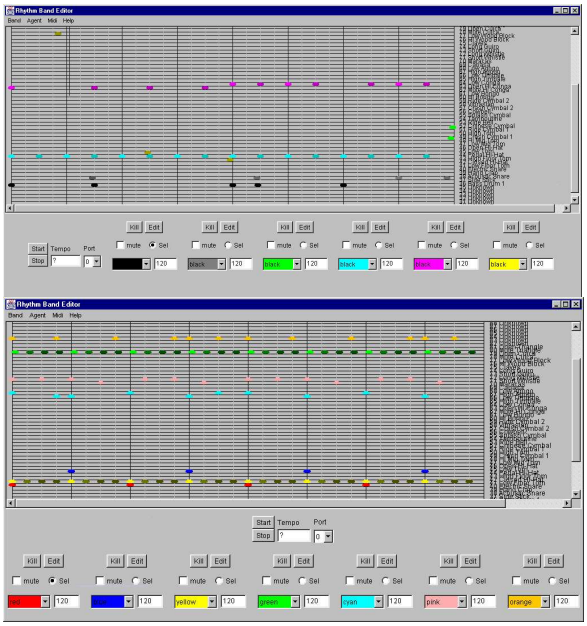


Figure 6. Two examples of complex popular rhythms taken as starting points for rhythmic evolution.

Implementation

The model has been implemented under the form of an editor for specifying agents, and their various perception and action parameters, including the initial rhythm of agents (possibly empty), their instruments (using General Midi convention), and their rules of behaviour. The real time multi tasking of agents is implemented with the Thread facilities in Java; the scheduling of notes is performed by MidiShare (Orlarey and al. 1997). The editor allows to define initial agent's melodies or chords and change them by clicking on the global score in real time (as in a standard drum box), as well as changing an agent's rules interactively.

The actual operators are implemented using the MusES music description language (Pachet and al. 1996). This language is a Java library including classes to define basic ingredients for music construction: melodies, chords, notes and intervals. Currently each rule is defined directly in Java and therefore requires a compilation process.

Discussion, future work

Current work include doing more experimentations to determine which basic operators are needed to produce a variety of different rhythms and variations.

Based on our experiments, we now look for a generalization of operators in two main categories: 1) Operators tending to emphasise strong / weak beats, by introducing in the rhythm a note on a strong/weak beat, 2) operators trying to produce syncopations, i.e. notes slightly ahead (or after) strong/weak beats. Each of these two operators are parameterised by the type of beat to apply to (strong or weak), and the nature of the emphasis (positive or negative). In the case of a negative emphasis, the agent withdraws any note it is possibly playing from his melody. These operators should be general enough to allow many variations, but adapted to rhythm specification, to avoid having to input a lot of parameters.

The formalization of this model should eventually lead to a constructive model of rhythm.

Finally, we also use the editor to produce actual music or soundtracks (documentary) as well as for live performance, where one of the agent is replaced by an actual humand Midi input.

References

Laine, Pauli (1999) Motor Neuron Based Virtual Drummer, ICMC, 1999.

Gerard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue, (1999) "Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic", CMJ, December.

Pachet, F. Ramalho , J. Carrive, J. (1996) Representing temporal musical objects and reasoning in the MusES system. *Journal of New Music Research*, Vol. 25, n. 3, pp. 252-275.