Representing Knowledge Used by Jazz Musicians

François Pachet
LAFORIA, Institut Blaise Pascal,
Université Paris VI, 4, place Jussieu, 75252Paris Cedex 05

Abstract : Our system is a framework for building Expert Systems that simulate jazz musicians 's playing. The music is generated according to a given jazz chord sequence, and the real-time MIDI input of a soloing musician. A MIDI output is generated, which control MIDI synthetisers. The style of music generated is standard "real book" style swing jazz. We outline here the basic architecture of this system, which relies on a three axis representation scheme : objects, rules and strategies.

Introduction. What kind of Knowledge ?

We describe here an architecture that allows representing certain kinds of Knowledge used by accompagnying Jazz musicians, namely swing bass lines or piano comping. The task of writing such knowledge-based systems is made extremely difficult by the diversity of the knowledge which varies from very formal definitions (inherited from Cassical Music) such as the formal theory of chords, modes, scales, to largely heuristic principles ("play the tonic of the chord in the first beat of the measure" for a bassist).

Procedural/Declarative Knowledge. Knowing how to build or analyse a chord, play a scale, find out common pitches for a given set of chords, may be considered procedural in the sense that there are algorithms available that do the job. More complex operations such as analysing a sequence of chords would not be considered procedural [Pachet 91]. Similarily, adding superstructures to a chord would not be considered procedural as it relies heavily on the context, which may be very complex. Moreover, declarative Knowledge in Music seems to be essentially non-monotonous and incomplete. Typically the bassist's leitmotiv *"play the tonic on the first beat"* is incomplete (expliciting the precise contexts in which this is true or false is impossible) and non-monotonous (the truth value of this statement may vary over time, chord changes, context).

Implicit/Explicit Knowledge. Representing Knowledge is limited to representing only explicit knowledge. Our system therefore represents abstractly the ideal "rational agent" for which every action taken has justifications that are coherent for its inference mechanism.

Importance of licks. The activity of live playing is not entirely spontaneous and creative. A tremendous corpus of information acquired through experience is used, retreived, modified along with inference activities. This corpus consists in pre-existing sets of musical phrases, chords, sequences of chords associated with (uncomplete) contexts of validity. This corpus play a central role in live music. It may even constitute a characterization of a player's style (i.e. the licks that recurrently appear in its playing). A representation of this somehow rigid knowledge has also to be set up.

Role of Time . Any musical composent has a time dimension, which has some properties used by the inference : the sequenciality of objects, duration, metrics, are all relevant aspects to be taken into account by the system. But time plays also an important role with regards to expectation/failure. Musicians play with the expectation of the audience by fulfilling anticipations or creating ruptures or breackdowns. To handle this, an history or trace of what has been played must be used and structured so as to be usable by the inference system.

Description of the system

According to the previous requirements, our system is build around a triadic paradigm : objects, rules and strategies. Those three elements form the kernel of our representation language and are articulated so as to provide the expressive power needed. They are handled by an inference engine which produces the music (a Midi output) according to the Midi input, the chord sequence and all the Knowledge in its various forms. Time is not explicitely represented but is handled by the engine in an ad'hoc way. Music in input is transformed into structured objects (notes and chords) having absolute time values. The engine produces music by chunks, and constantly checks its remaining time (i.e. the time left before the expected begining of the chunk). A Midi driver [Boynton86] is in charge of producing Midi output according to their time tags.

Time management. A distinction is made between the *logical* time and the *real* time. The real time is the actual time as given by the processor clock in milliseconds. The logical time is the time as handled by the engine and can be seen as "what the system is currently thinking of". This distinction is crucial, as the engine will constantly check it has enough time to spend on thinking in order to choose between alternatives : costly calculations will be conducted only if it has enough remaining time. Conversely, if the engine is late or "behind the schedule" (i.e. its logical time is greater than the real time) then the engine will chose direct "stimulus-response" like actions (licks for instance) in order to make up.
This criteria is explicit in the system (there is a rule that dispatch actions according to the difference between real and logical time). The tradeoff between the time taken to chose an action and the effective time taken to perform the action is a constant preoccupation in meta Knowledge Representation[Pitrat 90] and is given here an explicit representation.

Objects. Musical objects are structured according to the Object-orientation paradigms (i.e. instanciation, inheritance, message passing). In this scheme, classes are defined which describe both structures and behavior of their instances. For instance, chords are represented as instances of the class `Chord`, having attributes `listOfNotes`, and `harmonicDegree`. This provides an excellent platform to implement various procedural Knowledge such as :

- structural knowleedge. Objects are structured (in terms of attributes) and thus constitute a network : a chord sequence is composed of chords, themselves composed of a list of notes and a duration. A note consists in a pitch and a velocity and so on.
- Procedural Knowledge associated with these Objects. A chord for instance will be able to compute inversions, transpositions, to extract its most important pitches. Those procedures are called methods.
- Handle multiple representations. For instance, a musical phrase may be described as a sequence of notes, themselves described as having a pitch (a number, an initial time, an ending time, a velocity (or pressure)). Such a representation is inadequate at music production time : a representation using durations instead of absolute time will be prefered. Changing the representation will be made via a method in class `MusicalPhrase`.

Moreover, Objects are also used to structure the environment itself : A class `BassPlayer` is defined, having an **hear** (the object in charge of listening to the input data and converting it into a readable form), a **memory** (in charge of managing the various data containing the pre-recorded licks), a **mind** (the actual inference engine and the rule base), an **eye** (which sees and handles the chord sequence).
This structuration provides a natural representation of the Musician itself, and thanks to inheritance, makes specialization of these classes easy. For instance a class `PianoPlayer` will share a lot of common functionalities with the class `BassPlayer`. Only minor changes have to be made, so both classes will inherit from an abstract class `Musician`, and will define only those characteristic behaviors.

Rules. Knowledge related to groups of objects and their interactions is represented by rules. These rules implement the various operations a Musician can perform. These operations are well represented by rules because they include a context part (the definition of the context in which the action may be performed) and an action part (the actual action to be performed). The classical if-then production rule is thus adequate. However, the contexts of activations are not entirely defined : those rules act more like suggestions or advices than orders. Very often, those contexts are empty. For example, "the bass can play an arpeggio starting with the tonic on the first beat" may be represented as a rule in the system, with an emtpy context (a bass player can *always* do this).

Rules are organized in packs, each pack representing a certain chunk of Knowledge. The packs are not necessarily independant and may share objects. The packs may themselves be recursively structured in packs.
For the bass player rule base, a typical rule may thus be written as follows, stating that a possible action is to produce an apreggio from the root of the current chord, in the same tonality :

```
rule    r
                (currentChord ?c ?- ?- ?tonality)
then    (produce (arpeggioFrom ?tonality ?c))
```

This rule would be written in the pack "arpeggios", as well as other rules dealing with the many variations around producing an arpeggio. The conclusion part of the rule adds a new "`produce`" fact in the fact-base, with the necessary informations to actually compute the musical data. This `produce` fact will be handled by a specific pack of rules that will, if this action is actually chosen, eventually produce the corresponding Midi output data.

Strategies. As in any forward-chaining mechanism, rules have to be chosen, according to the actual contexts. Those choices involve an other kind of Knowledge called meta-Knowledge [Pitrat 90]. In order to provide a conceptual framework to represent this knowledge, we introduced the notion of strategy : a strategy is any potential action related to the music to be produced. It is structured as :
a `type` (specifying the nature of the strategy), a `time interval validity` (possibly open), a `name` (the name of the action), a `set of rules` (actually implementing the strategy).
Typical strategies will be : "play louder", "play more notes", "play arpeggio", "don't play", "play atonal", "play lick number 7612", "play last played phrase transposed by a half step" ...
Strategies beeing defined as classes, as for domain objects, subclasses of `Strategy` can be defined, having particular behavior, or adding structures. For instance, strategies of a special kind (say `OrderedStrategies`) may have priorities that will be used at conflict resolution time.
Since several strategies may be valid at a given time, strategies will interact with each other, at conflict resulution time, in order to determine the final decision for output. Conflict resolution will also be handled by rules, which will decide according to the current context which strategy to chose. In particular, this scheme allows the notion of "long-term" plan to be (partially) represented : once a strategy is chosen, the corresponding pack of rule is triggered, and the cycle starts again. At the next selection time, the strategy may still be valid. The choice of keeping the old strategy or choosing a new one will be made by conflict resolution rules.
Strategies are not always exclusive. For instance a strategy like "play louder" may coexist with a strategy like "play an arpeggio". The strategy "play louder" will not actually produce musical output, but will simply modify the musical data produced by the strategy "play an arpeggio", by producing a louder arpeggio. This distinction is represented by the attribute `type`. Strategies that actually produce something will have the type #`produce`, whereas strategies influencing others will have the type #`influence`. Of course type is not restricted to those two values, and other types can be added.

The Engine cycle. The engine cycle is classically threefold : identification of all matching rules, selection of one rule, application of the action part of the rule. The management of strategies is entirely described by rules. Rules are structured in packs, and each pack is triggered by the action specified in the action part of a rule. Rules implement the strategy management in three steps :

- identification of all possible strategies,
- conflict resolution : choice of actual strategies
- music production : according to the various strategies chosen, musical data is generated.

The top level rules for instance looks like :

```
rule    rLate
if
        (currentTime ?t)
        (logicalTime ?l)
        (< ?t ?l)           ; we are late on the schedule
then    (trigger fastDecisionPack)
```

a similar rule will handle the case when there is enough time to think :

```
rule    rAhead
if
        (currentTime ?t)
        (logicalTime ?l)
        (> (?t (+ ?l 1000))     ; we are 1 second ahead on the schedule
then    (trigger slowDecisionPack)
```

The pack `fastDecisionPack` will consist in choosing pre-recorded licks (in the `memory` of the Musician). Strategies themselves are handled by the lower level packs.

Current Implementation. The current representation uses the n-ary predicate syntax. In this scheme, facts of the fact base are described as lists whose first element is the predicate's name, and the remaining are arguments. Arguments may themselves be the result of Lisp functions (notated between parenthesis).
Strategies are therefore represented as : `(strategy aName firstBeat lastBeat aPackOfRules type)`. The chord sequence is represented by chunks : only the current chords are present in the fact-Base. The chord's structure is `(currentChord aChordName firstBeat lastBeat underlyingTonality)`, where `aChordName` is the chord's name according to the chords naming conventions; `firstBeat` and `lastBeat` are the current chord's range of validity in the chord sequence, and unerlyingTonality is the current tonality. An implementation in Smalltalk-80 using the NéOpus [Pachet91] environment is in progress.

Discussion

This representation scheme allows for representing a great deal of what a Musician (actually the author) can say about its own playing. It provides a good framework to differentiate between different types of Knowledge. Objects provide a very good and flexible representation framework. Rules give a good representation for potential actions to be performed, independently of their contexts of activation, and strategies, acting as an abstract representation of those rules, allow to separate between the description of actions, and the decisions related to those actions. However all Knowledge is not well represented. Notions related to time are poorly handled. A representation of rhythm is essential here, and should be described as a component of an other object. Our system does not have the ability of handling rhythm as a first class object. It is accessible only through the time properties of the objects. The level of abstraction is not sufficient now in this concern, and such notions as *swing*, *on the beat*, *slighlty ahead*, are not easily represented (they are accessed only by costly and hazardous computations).

References

[Boyton L] preFORM : an Object-Oriented environment for Music Composition. ICMC '89.
[Boyton L & al.] Midi-Lisp, a Lisp based Music Programming Environment for the Macintosh. ICMC '86.
[Jackendoff R., Legdahl F.] A Generative Theory of Tonal Music. Cambridge, MA. MIT Press, 1983
[Pachet F.] NéOpus User's manual. Rapport LAFORIA, July 91.
[Pachet F.] Towards a expert system that follows human improvisation. Rapport de DEA. Ircam, Paris 1988.
[Pachet F.] A Meta-level Architecture for analysing Jazz chord sequences. Same conference.
[Pitrat J.] Métaconnaissance, Hermes, Paris 1990.