Representing temporal musical objects and reasoning in the MusES system

François Pachet, Geber Ramalho, Jean Carrive, Guillaume Cornic

LAFORIA-IBP, Université Paris 6,
Boîte 169, 4, Place Jussieu, 75252 Paris, France.
E-mail: pachet@laforia.ibp.fr

ABSTRACT. We describe a representation framework for temporal objects and reasoning in the MusES system. The framework is claimed to be use-neutral within the context of tonal music. It is based on the use of an object-oriented programming language, Smalltalk, and makes full use of two main representation mechanisms: class inheritance and delegation. We describe the kernel of the representation framework and explain the main design choices. We show how the kernel can be used and extended to represent important temporal concepts related to tonal music. The framework is validated by the realization of two substantial applications: an automatic analyser for chord sequences, and a simulator of jazz improvizations.

1. Introduction

1.1. OOP and musical knowledge representation

Object-Oriented Programming (OOP) has traditionally been a favorite paradigm to build complex musical systems (see e.g. the special issue of the Computer Music Journal, 13 (2), 1989 on the use of Smalltalk for musical systems). This adequacy of object-oriented programming to musical representation has given birth to an impressive list of famous musical systems: the *Formes* system, to specify interacting concurrent musical processes (Cointe & Rodet 1991), the *MODE* system, a environment for the development of tools for musical score, sound processing and performance (Pope 1991), the *Kyma* system, a graphic environment dedicated to digital audio processing and computer-aided composition (Scaletti 1987), and more recently *Improvisation Builder*, a framework for simulating jazz improvisation, seen as a particular kind of communication between agents (Walker et al., 1992).

Following this movement, we are deeply convinced that the structures and mechanisms of object-oriented programming are particularly well suited for effective musical knowledge representation. We conducted a series of experiments aiming at studying more particularly the representation of knowledge related to tonal music. These experiments are embodied in a system called MusES [Pachet 94]. The MusES system contains a representation of the basic concepts of tonal harmony such as pitch classes, notes, intervals, chords, scales, and melodies.

An important claim underlying the MusES effort is that it is possible to provide a *use-neutral* representation, within the context of tonal music. By use-neutral, or application-independent, we mean that these representations may be used by specific applications as is, with only minor modifications if any. This claim is backed up by the philosophical assumption that there exist some common sense layer of musical knowledge which may be made explicit. We will not discuss the validity of this hypothesis here, and will take it for granted in the context of this paper.

Another requirement of our work is that we want our system to effectively capture consensual musical knowledge[1] in all its complexity. This requires our representation paradigm to be expressive enough for this task.

[1] By "consensual knowledge", we mean the layer of knowledge commonly agreed upon by musicians.

Several applications have been built using MusES that address specific problems of tonal music: a system that performs automatic analysis of jazz chord sequences (Mouton & Pachet 95; Pachet 91), a system that performs automatic harmonization of monophonic chorales satisfying the rules of harmony and counterpoint (Pachet & Roy 95), a system that generates jazz improvisation in real time using a case-based model of memory (Ramalho & Ganascia 94). Other applications are in progress that use the same MusES kernel, such as an interface based on the Harmony Space concept of (Holland 94), and a pattern induction system.

In this paper we describe the design choices made for representing *temporal* musical knowledge. This representation is claimed to be use-neutral, like the rest of the MusES system, and hence effectively reusable. We will first introduce our temporal model and the design choices we made, that fulfill our requirements. These design choices are validated by the realization of two substantial applications. For each of them, we will show how the basic classes were reused, and how the particularities of each temporal model could be easily represented and implemented using this framework.

1.2. Object-oriented Design: the fury of wild reification

Although it is extremely difficult to give a precise account of the modeling process necessary to produce a "good" object-oriented design of a real word problem, there are certain important guidelines that rule - more or less explicitly - the modeling activity. One of them is that only *interesting* objects should be reified (i.e. made explicit and given a first class status). By interesting we mean objects on which particular things have to be said, assertions stated, properties described, which are significantly different from other existing entities of the ontology. Although this principle could sound trivial, it is far from being the case in practice. We will give here some examples of which concepts of tonal music have been deemed interesting to reify, and which concepts have not. Since the quality of a good design comes precisely from these design choices, we will give some of the arguments which led to these decisions.

2. Our Temporal model

2.1. Temporal models for music representation

Time plays an important role in music, and hence in musical knowledge representation. Several theories of time have been developed in Artificial Intelligence, using different temporal primitives: *intervals* (Allen 84), *points* (McDermott 82) or *events* (Kowalski & Sergot 86). From a purely theoretical standpoint, the choice of primitives emphasis the dimension of time considered as fundamental in the problem solving context. However, the three approaches entertain strong relationships, and have been proved equivalent in terms of expressiveness (Tsang 87).

The model we propose here uses interval primitives. In this model, time is represented by intervals, having a start time and a duration. Allen (1983) showed that only 13 binary relations may be stated between two time intervals (meets, finishes, overlaps, etc.) Our model may be seen as a particular implementation of Allen's model, with important extensions as we will see below.

2.2. Temporal versus non temporal musical objects

The MusES system contains an object-oriented representation of consensual musical objects, with their most common properties and operations. The system contains around 90 classes (in the sense of object-oriented programming) and 1400 methods, representing properties and operations associated with these classes.

In the process of identifying the consensual musical objects, we realized that there is a lot of musical knowledge related to concepts having no temporal extension : pitch classes, octave-dependent notes, intervals, scales, chords, etc. Of course, some of these concepts also have temporal extensions. For instance, "actual" notes (in a melody) can be seen as octave-dependent note "plus" a temporal interval. These temporal concepts have specific properties that their non temporal counterpart do not have, such as precedence relations.

Therefore, we distinguish between two categories of objects: objects with no temporal extensions hereafter called non-temporal objects, and objects having a temporal extension, called temporal objects. As we will see, our representation framework for temporal objects will provide a certain kind of relation between these two categories of objects.

In the first category, we include non-temporal concepts such as:

- *pitch-classes* (e.g. A, Bb, F##). These objects entertain a non trivial algebra of alterations. Typically, enharmonic spelling is taken into account here (Pachet 94b),
- *octave-dependent notes* (e.g. A4, B*bb*3, etc.) They represent "materializations" of pitch-class in a given octave. These notes entertain special relationships with regards to intervals, since the scale is no longer circular,
- *intervals* (e.g. minor third, augmented fourth). They are also represented as first class objects. Methods allow the computation of the extremities of interval given a note, or the computation of an interval given a couple of notes,
- *chords*. We distinguish between *abstract chords* (such as [C min 7]) with no reference to particular octaves, and *octave-dependent chords* having an explicit pitch-list (a list of octave-dependent notes). Methods allow to compute chords from chord names or lists of notes, and conversely lists of notes from names.
- other objects such as *scales* (e.g. C# harmonic minor) and *signatures* (akin to scales), and more abstract objects such as *tonalities* and *analysis* (e.g. {II of E*b* major}).

The second category of objects includes objects having a temporal extension such as:

- *temporal notes*. These are the temporal equivalent of `OctaveDependentNote`. They are actually called "Playable notes" since they contain also various information related to performance such as amplitude, midi channel, and so forth.
- *temporal chords*. These are the temporal equivalents of octave-dependent chords,
- *melodies* (monophonic and polyphonic) and more generally objects representing *collections* of temporal objects.

It is important to note here that some sort of symmetry exists between temporal and non-temporal objects. `PlayableNote` is the temporal equivalent of `OctaveDependentNote`, and `PlayableChord` the temporal equivalent of `OctaveDependentChord`, and so forth. However, this symmetry is not systematic. All non-temporal objects do not have necessarily a temporal equivalent. For instance, `PitchClass` objects exist only as non-temporal entities; the concept of `TemporalPitchClass` has not been introduced (though it could have been), because no specific consensual properties of temporal pitch-classes have been found so far. Similarly, scales have no temporal equivalent. Conversely, some temporal objects do not have non-temporal equivalent. For instance, the notion of `Measure` (a part of a chord sequence or of a melody) has no non-temporal equivalent. An extreme case is the notion of `MusicalSilence` which, by definition, is useless in a non-temporal context: although we could indeed build a representation of silences (and therefore e.g. distinguish between "fourth-note values" and "eighth-note values") there is nothing more to declare about these concepts, that is not already stated in class `Lapse`.

2.3. The temporal kernel in MusES

The temporal kernel in MusES is organized around three basic concepts, represented by classes:

- `Lapse`

defines a time interval. Lapses are represented as explicit objects in our model for more flexibility. The class defines two attributes : `startTime` and `duration`.
The methods implemented in class `Lapse` are of three types:
1) basic access methods (temporal primitives), such as `endBeat`, and methods for comparing the temporal primitives between two lapses.
2) the 13 Allen primitives for the sake of completeness (`meets`: `overlaps`:, and so forth)
3) a set of useful methods and specific vocabulary. Although these methods are redundant with the methods defined above, it is important to offer the maximum flexibility when

dealing with temporal relations. Here a set of standard "useful" methods have been implemented (such as `intersectsOrEqualTo:`, `finishesBeforeOrEqualTo:` and so forth). In specific cases, subclasses of `Lapse` may be introduced (see section 3.4).

- `TemporalObject`
Defines a temporal object in general. The basic temporal object defines an attribute `lapse` holding an instance of the class `Lapse`. Concrete subclasses will specialize this class as seen below.

Note that this representation is to be opposed to the design choice made in the MODE system (Pope 1991); where lapses are represented independently from musical objects, through "association objects" held in temporal collections. From our point of view, the main difference between the two approach is the following. In a performance-oriented context (which is the context of the MODE system), the main role played by temporal object classes is as object generators: temporal objects are reified mainly to be created, and MODE provides a wealth of tools to create collection of notes yielding certain characteristics. Lapses are better represented "outside" notes, which facilitates editing operations (such as copy/cut/paste), and improves the modularity of the code. In a analysis-oriented context (which is the context of the MusES system), temporal object classes have to support a variety of analysis tasks, which are most of the time based on *local manipulation* and local pattern recognition sub-tasks (see below for a description of the analysis application, and the pact system, section 4.2.1). Therefore, notes have to somehow "know" their lapse directly, to facilitate the representation of these tasks.

- `TemporalCollection`
Represents a collection of temporal objects. Class `TemporalCollection` is defined as a subclass of `SortedCollection`, and ensures that its elements are sorted temporally, i.e. according to their lapse. This class defines all standard access protocol, such as getting all the temporal objects within a given lapse, or computing the intersection between two temporal collections, etc.

2.4. Defining temporal objects with the kernel

All meaningful temporal objects are defined using one or both of the two main construction mechanisms of object-oriented programming:

- Inheritance, i.e. defining a subclass of one of these classes. A simple example of the use of class inheritance is class `MusicalSilence`, defined as a subclass of `TemporalObject`, thereby inheriting attribute lapse, and adding specific behavior (mainly overriding methods related to pitch). Similarly, class `PlayableNote` is defined as a subclass of `TemporalObject` (see Figure 1).

- Delegation
  The mechanism of simple inheritance, however, is not enough to define all the temporal concepts we need : simple inheritance forces the designer to chose one and only one superclass for each class. For instance, the class `PlayableNote` will be defined as a subclass of `TemporalObject`, thereby inheriting from attribute 'lapse'. However, all the information pertaining to `OctaveDependentNote` will not be inherited. The use of delegation is an interesting alternative, as discussed in (Lieberman 86). It provides an alternative to inheritance that is often both more natural, and easier to implement than multiple inheritance. The delegation mechanism consists in defining a class with an attribute holding an instance of another class, to which a significant part of incoming messages are delegated. The main interest in using delegation is to allow a clear separation of behavior between classes.
  In MusES, this mechanism is mainly used to define collections of temporal objects (Cf. Figure 2). For instance, class `MonophonicMelody` is defined with an attribute "elements" holding an instance of `TemporalCollection`, and representing its actual notes. All messages pertaining to the list of notes are systematically delegated to this object (Cf. section 2.5),

In order to facilitate the use of these mechanisms, we introduce two intermediary concepts, that will be particularly useful to define new types of temporal concepts: `TemporalObjectWrapper`, and `TemporalCollectionWrapper`.

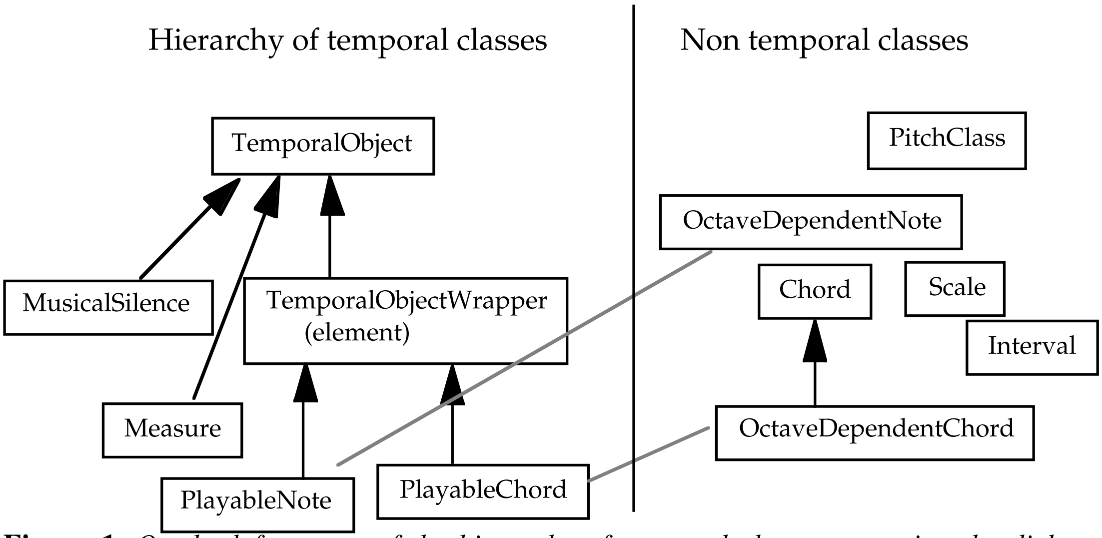

Figure 1. *On the left, a part of the hierarchy of temporal classes, stressing the dichotomy between temporal and non temporal objects. Attributes are between parenthesis. On the right, some non-temporal classes. Vertical arrows represent the class/subclass relationship. Dashed lines represent the delegation relationship.*

`TemporalObjectWrapper` is a subclass of `TemporalObject` that defines an attribute (`element`) holding an instance of a `TemporalObject` subclass. The main characteristic of this class is that it delegates all messages not pertaining to time to its `element` attribute.

`TemporalCollectionWrapper` follows the same pattern, though for collections. It defines an attributes `elements`, holding an instance of `TemporalCollection`. The main characteristic of this class is that it delegates all messages pertaining to temporal collection to its `elements` attribute. As we saw, a typical example is class `MonophonicMelody`, defined simply as a subclass of `TemporalCollectionWrapper`, thereby inheriting the delegation behavior to its list of notes. Moreover, a primary assumption of class `MonophonicMelody` is that silences are not represented explicitly.

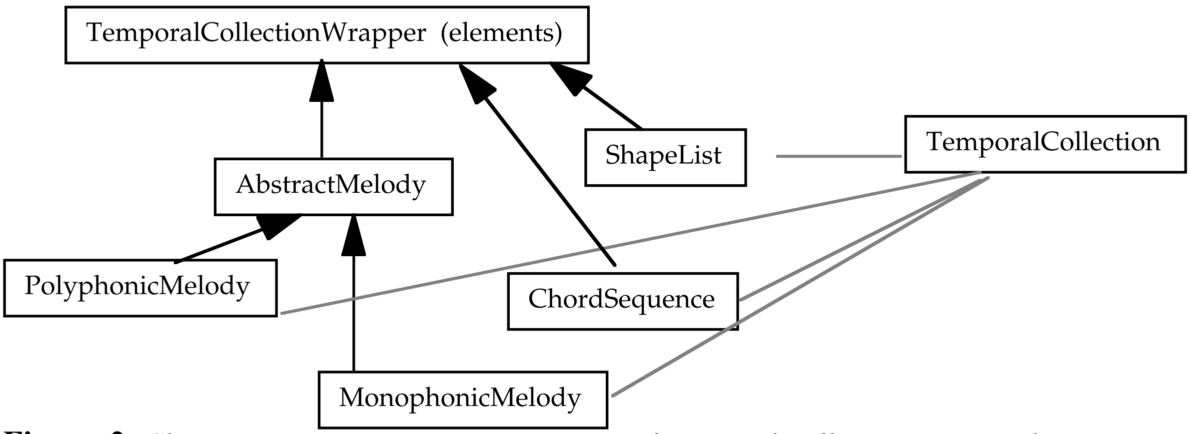

Figure 2. *Classes representing various concepts of temporal collections. Vertical arrows represent the class/subclass relationship. Horizontal lines represent the delegation relationship.*

2.5. Automatic delegation versus multiple inheritance

In order to minimize the representation and programming effort, we propose to automate the definition (and compilation) of delegating methods. The idea is to exploit reflective facilities of the underlying language (Foote & Johnson, 1989) to modify the interpretation of messages, dynamically, through the redefinition of the interpretation of error messages, coupled to a dynamic compilation of delegating methods. In this scheme, for instance, each time a playable note object receives a message pertaining to its delegates (octave-dependent note) it will: 1) provoke an error because the class PlayableNote does not understand the message, 2) compile the delegating method, 3) re-send the message to itself, which will eventually result in the delegation to the octave-dependent note object.

For instance, the first time an instance of `TemporalNote` receives, say, message `fourth`, our mechanism will compile the following method in class `TemporalNote` :

```
fourth
    ^note fourth
```

The message `fourth` is then sent again to the same instance which will now delegate it to the `OctaveDependentNote` object (attribute `note`), and yield the expected result. Note that the compilation of this method is performed automatically, the first time an instance of class `TemporalNote` receives message `fourth`. Subsequent messages will be directly interpreted by this automatically compiled method. The same mechanism is used for each subclass of `TemporalObject` having some non-temporal equivalent.

Note that multiple inheritance could be an alternative here, but it would not solve the problem: it provides another formulation which does not lead to a simple design. Instead of choosing the right simple inheritance tree, the problem becomes the choice of the right conflict resolution strategy. This is not the approach we followed here.

We will now describe briefly two applications of the MusES system which both use intensively the temporal model described here. Each application has specific needs concerning the temporal model, which are not taken into account on the kernel. We show how these specific needs are represented simply using inheritance and delegation.

3. Application 1: analysis of jazz chord sequences

3.1. Background

The aim of the MusES analysis system is to build up a model for the analysis of jazz chord sequences, as found in the standard corpus of (Real 81), or (Fake 83; 91). Our goal is to build a fully operational model that account for most of the regularities found in this corpus.

The problem of jazz chord sequence analysis consists in computing, for a given chord sequence the underlying tonality of each of its chords. The main characteristics of this analysis is that it is hierarchical: a tune may be globally in C major, but some parts of it may be in F (modulations), and so on. Generally speaking, harmonic analysis produces a tree with which each chord of the sequence may be analyzed, at several levels of abstractions. Lastly, the aim of the analysis is usually to provide, for each chord of the sequence, indications to the musicians for improvisation. These indications are the underlying tonalities (at all levels of abstractions), as well as identifications of well-known "patterns" that make sense for the improviser, because he will be able to use pre-defined licks well adapted to these patterns.

Several attempts have been made to provide computational models for automatic harmonic analysis of tonal pieces, using various techniques: procedural (Ulrich 77), (Smoliar 1980), grammar-based (Steedman 84; Winograd 68), rule-based (Maxwell 92) or constraint-based (Steels 79). No system however have proposed a fully operational model that accounts for the specificity of Jazz chord sequences.

3.2. The theory behind, revisited

Like classical harmony, tonal jazz harmony is a well studied domain, as one can see by browsing at the numerous books written on this subject (Coker 64). However, to our knowledge, no book attempts at providing a model for the analytic process per se. The situation is actually comparable to the situation in linguistics : if lots of works have attempted to find grammars for natural languages, only few operational models of language understanding have been developed.

Before describing our model for analysis, we propose to formalize the problem around three major points, as follows:

A) Basic principles

The theory is based on two major principles:

1) A "legality" principle

This principle says that each chord, out of any context, can be analyzed in a fixed set of possible tonalities. A tonality is faithfully represented as a scale (a list of notes) and a degree. For instance, a C major chord may be analyzed as: I st degree of C major scale, IVth degree of G major, Vth of F major, VI of E harmonic minor, and so forth. The computation of this "legal set" is entirely deterministic.

2) A minimization principle

In a context, the choice of the "good" tonality for a chord will of course depend on its location, and its relation with adjacent chords. The main idea here is that the best tonality will be the one that minimizes modulations, i.e. that is common to the greatest number of adjacent chords. For instance, the sequence (C / F / E min / A min) has only one tonality that is common to all chords: C major.

B) Perturbations

This nice and simple theory is complicated by phenomena that escape rigorous formalization, but which are essential to capture the essence of the process: substitutions and idioms.

First, some chords may be substituted by others, and the substitute often violates the legality principle. For instance, a seventh chord that resolves may be substituted by its tritone seventh (C7 -> F#7). Second, there are a number of well-known idiomatic "musical shapes" that bear particular harmonic meaning in themselves. This is the case of "two-fives", turnarounds, and other similar shapes. These shapes are remarkable in that they may be analyzed out of their context. Thus, the sequence "Cmaj7/A 7/Dmin7/Db7" is in itself a turnaround in C major, regardless of the fact that C major does not belong to the legal set of Db7. In other terms, Db7 *in abstracto* may not be analyzed in C major, and can only be within such a musical shape.

C) Recursion

Lastly, the process is *recursive*. This means that any recognized shape may itself be considered as atomic for a higher level of analysis. This recursive nature accounts for the hierarchical nature of the analysis. For instance, resolving seventh chords may be considered as preparations, and therefore integrated to their resolving chord. Typically, the sequence: "A7 / D7 / G7 / C" may be entirely analyzed in C major, thanks to a recursive reasoning (see Figure 3).

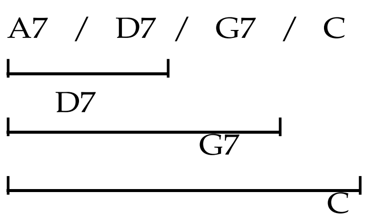

Figure 3. *The hierarchical nature of the analysis of a group of chords.*

At a highest level, global macro forms are introduced in a similar fashion. Thus, a Blues is identified by a succession of 3 musical shapes, covering 12 bars, and such that the fourth of the root of the middle one's tonality is equal to the root of the first and of the last shape. Structures such as AABA or ABAB may be described similarly.

3.3. The analysis reasoning in MusES

The reasoning process is represented by a series of rule bases that perform two kinds of tasks:

- a "pattern recognition" task, in which higher level shapes are identified from configurations of lower level shapes,
- a "forgetting" task, in which irrelevant or redundant shapes are destroyed.

Other rules describe shapes such as resolutions (A7 / D), turnarounds, and substitutions. More abstract rules describe more complex phenomena such as "modal borrowing": a local modulation may be considered as non significant in certain cases, when it comes in between two shapes analyzable in the same tonality (Cf. Figure 4).

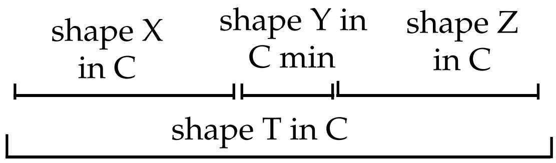

Figure 4. *Modal borrowing configuration.*

For instance, here is the rule (in a Smalltalk-like pseudo syntax) that recognizes a two-five in major (such as "Dmin7/G7"):

```
majorTwoFive
FOR a1 a2 instances of AnalysableObject
IF
    c1 isMinor.
    (c1 hasA: #flatFifth) not.
    c2 isAfter: c1.
    c2 isMajor.
    c2 hasA: #minorSeventh.
    c2 root pitchEqual: c1 root fourth.
THEN
Create a TwoFive, x.
    x beginBeat: c1 beginBeat; endBeat: c2 endBeat.
    x tonality: (c2 root fourth majorScale);
Establish composition link between x and c1 and c2 .
```

At the end of the reasoning, the complete analysis tree is produced. The system is now in the evaluation phase, and already proved capable of analyzing blues chord sequences deemed difficult by (Steedman 84).

3.4. Needs for management of circular time

Shape objects are naturally represented as particular temporal objects. The analysis system was therefore built using the temporal model of MusES. However, we need to distinguish between several time models for chord sequences, depending on the task to perform on them : the time model most adapted to analysis is not necessarily the same than the model to be used for performance. Indeed, there is an important specificity of the temporal model underlying jazz chord sequences. Instead of a linear model of time, a circular model of time is more appropriate for analysing jazz chord sequences, since the end of a tune usually turns back to its beginning.

This difference is not only superficial. As far as analysis is concerned, our experiments showed that the circular characteristic of the corpus was extremely important, and allowed to solve ambiguities inherent to tonal harmony. For instance, in blues tunes such as "Blues for

Alice" (Charlie Parker), the underlying tonality of the starting chord (F major) can only be ascertained by linking it with the unresolving seventh chord (C 7) of the end of the tune: the unresolving end of the tune ensures the tonal stability of the beginning of the tune (Cf. Figure 5).

D-7 (II)	G 7 (V)	C (I)	

C (I)			
		D-7 (II)	G 7 (V)

Figure 5. *A normal 2-5-1 on the left. A 2-5-1 that wraps around the end/beginning of the song on the right.*

More generally, we frequently need to manipulate abstract temporal shapes that can wrap around the beginning of a song.

Although we could use a purely linear model of time (such as Allen's), this would imply a systematic test for each form to be manipulated. For instance, testing that there is indeed a 2-5-1 structure beginning at the end of a tune and ending at its beginning requires a special treatment. With a circular model of time, only one rule is necessary. The same holds for all aggregation rules. Note that the circularity of time has to be introduced in the model itself.

This led us to introduce a circular representation of time in our model. The initial model was simply extended by introducing a subclass of the original `Lapse` class, called `CircularLapse`. We will now describe this new class in more details.

3.5. Circular lapses

3.5.1. Purely circular lapses

Allen enumerates thirteen possible relations (or configurations) between two lapses of time in a linear representation. To switch from this linear representation to a circular representation amounts to doubling the number of possible relations: for each relation in a linear time correspond two relations in a circular time, depending on whether the beginning of the first lapse coincides or not with the end of the second one (see figure 6). Thus we get twenty-six distinct relations for circular lapses. Note that we do not consider here lapses which last more than one period, since time would not be circular anymore, and would be better seen as a kind of spiral. Moreover, following Allen, we do not consider lapses reduced to an empty interval.

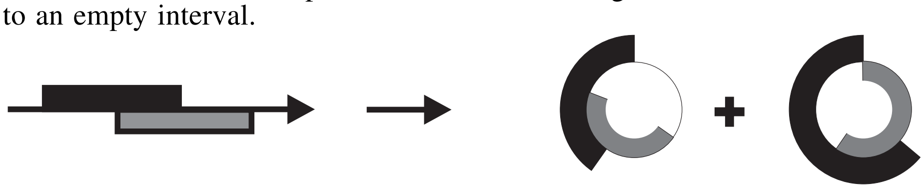

Figure 6. *linear and circular representations.*

3.5.2. Circular Lapses with an origin date

In both representation (linear and purely circular) an *origin date* can be introduced. The number of relations between two lapses increases greatly, because the origin date can be placed at any position within the topology of the binary relation. In the example illustrated by figure 7, the origin date may be located at seven different positions !

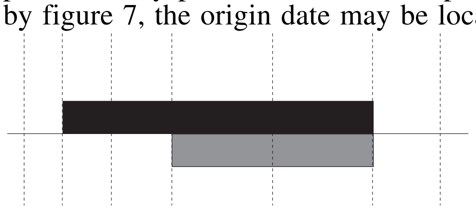

Figure 7. *Adding an origin date to a linear representation of time*

In the linear case, we get 101 different binary relations; and 150 in the circular case. Figure 8 indicates all circular relations with origin dates, and with phase information. This is the model we use in the analysis system.

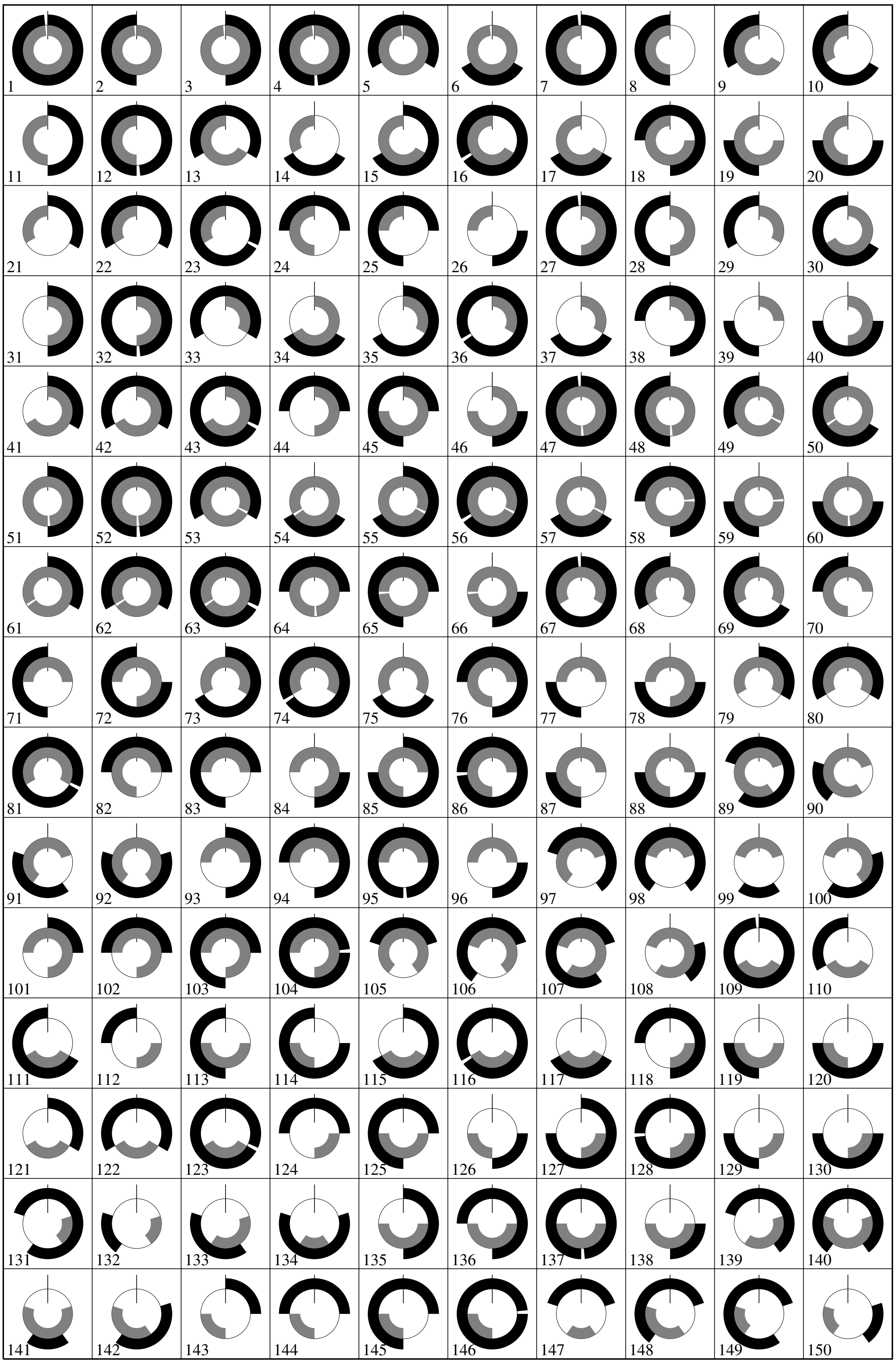

Figure 8. *All possible binary relations for circular lapses with original date and with phase information.*

3.5.3. Lapses that cover a period

There is a third distinction that could be made, with regards to intervals covering the entire time span: this is the case for time intervals covering the entire period of time. It can be interesting to say that all intervals covering the entire time span are equivalent. This amounts to withdrawing phase information from the representation of intervals. In this case, the total number of binary relations is diminished, since some of the previous binary relations become equivalent (e.g. relations 1 and 4 in Figure 8). Since this second model is subsumed by the preceding one, we did not implement it (see (Carrive, 1995) for details).

3.6. Representation

As we saw above, the basic class `Lapse` represents lapses in a linear time. We introduce class `CircularLapse` which inherits from `Lapse`, and adds an instance variable that represents the *duration* of a period. Similarly, class `CircularLapseWithOrigin` class inherits from `CircularLapse`. These circular lapses are then used as default values for the lapse attribute of shape objects (themselves defined as particular temporal objects).

All the possible binary relations are not implemented as explicit methods in these classes, and only primitive access methods are implemented. A current work in progress is to find a reasonable classification of these relations together with a meaningful vocabulary.

3.7. Usefulness of the model

The most complete temporal model is the model illustrated in figure 8, with origin dates and with phase. This is the one we used for the analysis system. The introduction of our circular model allows to reduce drastically the number of rules in the expertise for analysis. Moreover, certain properties between shapes are easier to state with this model, than with the linear one, such as properties involving union or intersections of shapes.

The comparison between the two models is straightforward: there has to be somewhere in the knowledge base "tests" about the relation between a temporal shape and the end of the tune. In the linear model, these tests have to be written in the rule making up the expertise. In our circular model, the tests are buried in the temporal primitives of the model, thereby reducing the overall complexity of the system.

4. Application 2: Simulation of improvisation

4.1. Brief overview of the model

The MusES system knowledge representation framework as well as the result of the Harmonic Analysis system discussed previously are used in another system (Ramalho & Ganascia 94; Ramalho & Pachet 94), which generates a bass line given a jazz chord grid as found in Fake/Real books (Fake 1983; 1991). In order to represent faithfully the environment of the bass player, we enrich the system's input by the introduction of a scenario. This scenario contains symbolic events related to the other musicians actions (e.g. “soloist using Dorian mode” or "drummer is playing louder and louder") and audience reactions (e.g. "applause" or "boo").

Departing from statistic-based (Ames & Domino 92) or grammar-based (Lerdhal & Jackendoff 83) attempts to formalizing musical knowledge, we use an explicit representation of musical actions, through the notion of PACTs (from Potential ACTions). These PACTs, initially introduced by Pachet (1991b) typically include actions such as "play diatonic scale in the ascending direction during this measure”, “play this lick transposed one step higher”, “play more and more loud until the end of the improvisation section” and so on.

With respect to the basic reasoning mechanisms, the model uses a "hybrid" problem solving method involving production rules and case-based reasoning. To our knowledge, this second component has been less explored so far despite the central role that a long-term memory seems to play in jazz learning process (Ramalho & Ganascia 94b). In this perspective, we introduce the notion of Musical Memory which accumulates bass player's experience acquired by musicians in terms of fragments of actual jazz performance transcriptions. Since PACTs provide us with a flexible knowledge representation framework, they constitute the building block of our model, unifying the representation of both the cases in the memory and the abstracts concepts (e.g. syncopation, consonance or pitch contour) manipulated by the rules.

PACTs stored in the Musical Memory are fully specialized, i.e. have all or almost all attributes specified. However, PACTs activated during the improvisation process are often under specialized. It is by their assembly that we can instantiate the notes to be played. Indeed, a basic property of PACTs is that they may be combined into more “playable” PACTs according to their mutual compatibility. For instance, the PACT “Play ascending notes” may combine with “play triad notes” in a given context (e.g. C major) to yield “play C E G”. This ability to combine is at the heart of the inference cycle. Firstly, PACTs are activated according to the latest scenario events, the chords of the current grid segment and the bass line played so far. Then, the previously *activated* PACTs overlapping such segment are *selected*. Finally, all these PACTs are *assembled* into a single playable PACT by the successive application of conflict resolution and combination operators. Moreover, since there is no guarantee that a set of PACTs contains the necessary information so as to produce a playable PACT, the cases stored in Musical Memory can be retrieved and modified to provide missing information.

```
TemporalObject ('lapse')
    ChordChunk ('shape' 'tonalities' 'resolution' 'chords''section' 'rhythmicStructure'
                'chordGrid' 'positionInForm')
    Edge ('initialValue' 'finalValue' 'slope' 'b')
    Pact ('creationDate' 'playability')
        .....
    RhythmNote ()
    ScenarioEvent ()
        BasicOtherMusicianEvent ('musician' 'value' 'variation' 'type')
                DynamicsEvent ()
                HarmonicEvent ()
                RhythmicEvent ()
        AudienceEvent ()
                PassingObjectEvent ('object' 'direction')
                SomeoneSaysEvent ('someone' 'says')
        FineGrainedOtherMusicianEvent ('motive')
    TonalitySpan ('scale')
```

Figure 9. *The hierarchy of temporal objects specific to the improvisation system.*

4.2. Reusing MusES

In this section we will describe how MusES objects are used as a first knowledge representation layer in the improvisation system. Some basic classes, such as `PitchClass`, `OctaveDependentNote`, `Chord`, `PlayableNote`, `Scale`, `MonophonicMelody` are used directly by the inference mechanisms. However, in complex applications, we need quite often to add some new classes. We will discuss how we take advantage of class inheritance to easily add new classes.

4.2.1. New Temporal Objects

Figure 9 shows some classes added under `TemporalObject`. For sake of readability the hierarchy of PACTs is sketched separately in Figure 10. These new classes take full advantage of the fact that MusES temporal objects know their own lapse. In fact, we deal with temporal objects that may be much more complex than notes and that are not necessarily grouped according to the "monophonic and no gap" constraints of melodies, i.e. there is only one note or rest per moment and this note or rest is immediately followed by another one until melody finishes. For instance, there are gaps between the scenario (symbolic) events since no guarantee exists that a scenario event will occur just after the last one.

```
Pact ('creationDate' 'playability')
    BackgroundPact ('type')
        ToBeContextualizedPact ('relativeValue' 'relativeValueVariation')
        ToBeDiscretizedPact ('absoluteValueVariation')
    DeclarativePact ()
        ComplexPact ('futureExpectations')
                ComposedPact ('pitchPact' 'rhythmPact' 'amplitudePact'
                                'rhythmAndPitchPact' 'melodicPattern')
        SimplePact ('hostPact')
                OneDPact ()
                        AmplitudePact ('loudness' 'amplitudeContour' 'envelope')
                        PitchPact ('pitchContour' 'tessitura' 'scale' 'dissonance')
                        RhythmPact ('syncopation' 'density' 'rhythmStyle' 'rhythmicPattern')
                RhythmAndPitchPact ('leadingTone' 'inversion' 'lineStyle'
                                'pullDown' 'rpSequence')
    ProceduralPact ()
                RepetitivePact ('sourcePact' 'repetition')
                TransformingPact ('playablePact' 'transformation')
```

Figure 10. *The hierarchy of PACTs in the improvisation system.*

PACTs are typical examples of objects that make extensive use of the properties defined in `TemporalObject`. A PACT can be activated to last two beats, one measure, one section or an infinite duration. Diversified types of time intersection may thus be found among the PACTs. In fact, the type of time intersections plays a central role in the production rules that deal with PACTs conflict resolution and combination. For instance, if two PACTs do not intersect, then they are incompatible, regardless of their contents. When two PACTs (e.g. "play loud" and "play quiet and consonant") are *locally incompatible* (within a given chord grid segment) they may be considered globally incompatible whether the former *is-during* or *finishes* the latter.

Another example concerns the PACTs preference criteria used to choose between two incompatible PACTs. When the playability criterion is not sufficient to indicate which of two PACT is to be preferred, then the next criterion is their lapse and creation date. Sometimes, according the kind of PACT, the "youngest" is chosen in order to guarantee a fast reaction to a given situation. Other times, the PACT having the longer duration is chosen in order to provide a uniqueness or coherence of the performance.

4.2.2. New temporal collection wrappers

As explained in section 2.4, we can introduce new temporal collection wrappers simply by defining subclasses of `TemporalCollectionWrapper`. Using this technique we defined easily new temporal collection wrappers such as: `Scenario` that contains scenario events, `WorkingMemory` that contains all kinds of activated PACTs, `Envelope` that contains temporal edges, and so on.

Two particularly interesting temporal collection wrappers are `RhythmicPattern` and `MelodicPattern`, both subclasses of `TemporalCollectionWrapper`. `RhythmicPattern` represents a rythmical squeleton, with no specified pitch, and contains specific knowledge pertaining to improvized musical rythms, such as transitions, computations of subjective features (density, syncopation). `MelodicPattern` is the melodic equivalent of `RhythmicPattern`.

On the contrary to what was initially provided by MusES in `MonophonicMelody`, we needed to explicit rests (`MusicalSilence`) in melodies. This necessity of explicit silence is raised by the nature of the knowledge handled by rules used during PACTs combination.

Since PACTs are often partially specialized, the absence of notes in the assembly context means "ignorance" rather than actual rests. The rests are intentionally inserted in the temporal holder as a consequence of a reasoning process that actually occurred and not as an absence of it, which is not the case for MusES `MonophonicMelody`.

We also introduced for `RhythmicPattern` the notion of `RhythmNote` (i.e. a note without pitch). As discussed later in this section, it would be difficult to deal with concepts such as syncopation or density without reifying the notions of note with no pitch and rest. Despite the differences between `RhythmicPattern`, `MelodicPattern` and `MonophonicMelody`, no significant effort was needed to introduce silences and rhythm notes since kernel class `TemporalCollection` can manage any sort of temporal object.

4.3. Reasoning on temporal and non-temporal objects: an example

The duality between temporal and non-temporal objects is exploited from the reasoning perspective in many opportunities by the improvisation system. Let us give an example to stress how the system takes advantage of the clear separation between temporal objects and non-temporal ones in the computation of notes in the context of PACTs combination.

When two PACTs are combined, their non conflicting information is added into a new PACT. If there is enough information in this PACT, some demons are triggered to compute values of other attributes. For instance, let us consider two `RhythmPact` during 4 beats: "play with rhythm style = 4-beat (notes' duration is quarter-based)" and "play with medium syncopation and high density". These two PACTs could be combined into a new one, triggering a demon which computes the *rhythmic pattern*. Let us suppose that the resulting rhythmic pattern is in this case "quarter quarter quarter eighth eighth". For this computation only temporal "no pitch objects" such as `RhythmNote` and `MusicalSilence` are manipulated. This task involves knowledge concerning mainly rhythm features (such as tempo, syncopation, density, rhythm style, phrase duration) as well as rhythm & pitch features such as pull down.

The resulting `RhythmPact` could be combined with other PACTs to generate a `ComposedPact` that has enough information to assign pitches to the rhythmic pattern just computed. Let us suppose that the *line style* attribute of this `ComposedPact` is "arpeggio" (chord-based). The process of generating the sequence of `PlayableNote` is four-step. Firstly, taking into account the rhythmic pattern description, we determine that the arpeggio is 4-note based. Then, we calculate these four pitches manipulating only non-temporal objects (`Chord`, `OctaveDependentNote` and so on). This task takes into account knowledge concerning the following features: 'leadingTone', 'inversion', 'lineStyle', 'pullDown', 'pitchContour', 'tessitura', 'scale' and 'dissonance'. Next, the playable notes are generated through assigning the computed pitches to the four rhythm notes whose onset correspond to the beats 1, 2, 3 and 4. Last, for the remaining rhythm notes, pitches are calculated as passing notes and the note sequence is thus completed.

5. Conclusion

We described a framework for representing temporal objects and reasoning in the context of tonal music. The framework is based on a temporal ontology organized around three basic classes, and extended using the two main mechanisms of object-oriented programming: class inheritance and delegation. The resulting framework is claimed to be use-neutral. The claimed is backed up by examples of sophisticated temporal objects taken from two substantial musical knowledge-based systems. Other musical applications are in progress (automatic harmonization, pattern induction), which make full use of the temporal kernel described here, thereby validating our initial design choices.

6. References

Allen J.F. (1983) Maintaining Knowledge About Temporal intervals, *Communications of the ACM*, 16 (11), pp. 832-843.

Allen J.F. (1984) Towards a general theory of action and time, *Artificial Intelligence*, vol. 23, pp. 123-154.

Ames, C. & Domino, M. (1992). Cybernetic Composer: an overview, In M. Balaban, Ebicioglu K. & Laske, O. eds., Understanding Music with AI: Perspectives on Music Cognition, The AAAI Press, California.

Carrive, J. (1995). Analyses de grilles de Jazz dans le système MusES. Master thesis, Laforia-IBP, university of Paris 6, Sept. 1995.

Cointe, P. Rodet, X. (1991) Formes: Composition and Scheduling of Process. In *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology* , S. T. Pope, ed. MIT Press.

Coker J. (1964) *Improvising Jazz.* Computer Music Journal, Prentice-Hall.

Fake, (1983) *The World's Greatest Fake book.* ed. C. Sher., San Franscisco: Sher Music Co.

Fake, (1991) *The New Real Book.* ed. C. Sher. Vol. 2., Petaluma: Sher Music Co.

Foote, B & Johnson, R (1989), Reflective facilities in Smalltalk-80. *Proceedings of OOPSLA'89*, pp. 327-336, New Orleans, Louisiana

Holland, S. (1994). Learning About Harmony Space: An Overview. M. Smith, A. Smaill & G. Wiggins Eds, *Music Education: an Artificial Intelligence Perspective*, Springer-Verlag, London.

Kowalski R.A. & Sergot M.J. (1986). A logic-based calculus of events, vol. 4, pp. 67-95.

Lerdahl, F. and R. Jackendoff (1983) A Generative Theory of Tonal Music, Cambridge: MIT Press.

Lieberman, H (1986), Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems. *Proceedings of OOPSLA '86*, Portland, Oregon, pp. 214-223.

Maxwell, H.J. (1992) An Expert System for Harmonizing Analysis of Tonal Music, in *Understanding Music with AI: Perspectives on Music Cognition*, K.Ebcioglu, O.Laske and M. Balaban, Editors, AAAI Press: p. 335-353.

McDermott D. (1982) A temporal logic for reasoning about processes and plans, Cognitive science, vol. 6, pp. 101-155.

Mouton, R. Pachet, F. (1995) Numeric versus symbolic controversy in automatic analysis of tonal music. IJCAI'95 Workshop on Music and Artificial Intelligence, Montreal (Ca), August 1995.

Pachet, F. (1991) A meta-level architecture for analysing jazz chord sequences. *International Conference on Computer Music*, pp. 266-269, Montréal, Canada.

Pachet, F. (1991b) Representing Knowledge Used by Jazz Musicians. *International Conference on Computer Music*, pp. 285-288, Montréal, Canada.

Pachet, F. (1994) The MusES system : an environment for experimenting with knowledge representation techniques in tonal harmony, in *Proceedings First brazilian symposium on computer music - SBC&M '94*, Caxambu, Minas Gerais, Brazil, pp. 195-201.

Pachet, F. (1994b). An object-oriented representation of pitch-classes, intervals, scales and chords. *Journées d'informatique Musicale*, Bordeaux (France), March 1994, pp. 19-34.

Pachet, F. & Roy, P. (1994). Mixing constraints and objects: a case study in automatic harmonization. Proceedings of *TOOLS Europe '95*, Versailles (France), Prentice-Hall, pp. 119-126.

Pope, S. (1991). Introduction to MODE: The Musical Object Development Environment. In *The Well-Tempered Object: Musical Applications of Object-Oriented Software Technology*, S. T. Pope, ed. MIT Press.

Ramalho, G., Pachet, F. (1994). From real book to real jazz performance. *International Conference on Music Perception and Cognition*, Lièges, Belgium, july 1994.

Ramalho, G., Ganascia, J.-G. (1994). Simulating Creativity in Jazz Performance. *Proc. of 12th AAAI conf.* Seattle.

Ramalho, G., Ganascia, J.-G. (1994b). The Role of Musical Memory in Creativity and Learning: a Study of Jazz Performance, In M. Smith, Smaill A. & Wiggins G. eds., *Music Education: an Artificial Intelligence Perspective*, Springer-Verlag, London.

Real (1981) *The Real Book.*, The Real Book Press.

Roads, C. (1988) Grammars as Representations for Music, in *Foundations of Computer Music,* C. Roads and J. Strawn, Editor., MIT Press: p. 403-442.

Scaletti, C. (1987). Kyma: An Object-oriented Language for Music Composition. in *Proceedings of the International Computer Music Conference*. International Computer Music Association, San Francisco.

Slade, S. (1991). Case-Based Reasoning: a Research Paradigm, AI Magazine, Spring, 42-55.

Smoliar, S.W. (1980) A Computer Aid for Schenkerian Analysis. *Computer Music Journal* 4 (2), pp. 41-59.

Steedman, (1984) M.J., A Generative Grammar for Jazz Chord Sequences. *Music Perception*. 2 (1), pp. 52-77.

Steels, L. (1979) Reasoning modeled as a Society of Communicating Experts, MIT AI Lab., n. AI-TR-542.

Tsang E.P. (1987) Times structures for Artificial Intelligence, in *Proceedings 10th IJCAI*, pp. 456-461.

Ulrich W, (1977) The Analysis and Synthesis of Jazz by Computer, in *Proceedings Fifth International Joint Conference on Artificial Intelligence*, MIT, Cambridge, Ma, pp. 865-872.

Walker, W., Hebel, K., Martirano, S., Scaletti, C. (1992). ImprovisationBuilder: improvisation as conversation, *Proc. of ICMC*, 1992.

Winograd, T. (1968) Linguistic and Computer Analysis of Tonal Harmony, *Journal of Music Theory*, 12, pp. 2-49. Reprinted in *Machines Models of Music*, S.M. Schwanauer and D.A. Levitt, Editor. 1993, MIT Press: p. 113-153.