

ANALYZING INTENTION IN
MUSICAL PERFORMANCE


by

KEVIN THOMAS LOONEY

A THESIS

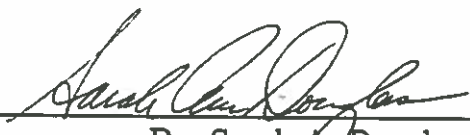
Presented to Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Master of Science

June 1988

Approved: 
Dr. Sarah A. Douglas

An Abstract of the Thesis of
Kevin Thomas Looney for the degree of Master of Science
in the Department of Computer and Information Science
to be taken JUNE 1988

Title: ANALYZING INTENTION IN MUSICAL PERFORMANCES

Approved: 
Dr. Sarah A. Douglas

In this thesis, I exploit the use of intentionality in music, particularly in performance systems. I have created a model for an ensemble of performers. This model stresses intention and negotiation to cooperatively add to the musical performance of multiple agents. My thesis project is a limited version of this ENSEMBLE model, focusing on three basic intentionalities: changing tempo, changing dynamics, and negotiating for a solo performance. The problem being addressed is the lack of a cognitive basis for current representations of music. My system can produce musical output, or modify its planned events on the basis of interpretations of other performers intentions. As a result of this research, I have formalized criteria for interpreting clues that result in belief. These clues represent gestures and musical events. The system is implemented in ORBS, an object-oriented, rule-based programming language developed for artificial intelligence applications. A performance is more than just the production of notes. This methodology defines a performance as the visual and musical effort, and the score as only a blueprint.

VITA

NAME OF AUTHOR: Kevin Thomas Looney

PLACE OF BIRTH: Hartford, Connecticut, USA

DATE OF BIRTH: 29 August 1962

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, Oregon, USA

State University of New York at Buffalo, Buffalo, New York

DEGREE AWARDED:

Master of Science, June 1988, University of Oregon

Bachelor of Science, May 1985, State University of New York at Buffalo

AREAS OF SPECIAL INTEREST:

Artificial Intelligence

Computer Music Applications

PERSONAL EXPERIENCE:

Teaching Assistant, Department of Computer and Information Sciences
University of Oregon, Eugene Oregon, 1986-1988

Programmer, General Electric Company
Syracuse, New York, 1985-1986

Programmer, Hansford Data Systems
Rochester, New York, 1985

Teaching Assistantship, Department of Computer Science
University of Buffalo, Buffalo, New York, 1984-1985

PUBLICATIONS:

Looney, K. (1988, February). Artificial intelligence and other buzzwords,
Keyboard, pp. 12

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to Professor Sarah Douglas, my advisor. Her encouragement and guidance throughout my study has been invaluable. Special thanks are also due to Professor Kent Stevens and Professor Stephen Fickas for their support, and insightful comments. I also express my gratitude to Michael Hennessey for his patience and comments during some stages of this work. To my colleague and partner in thesis formatting crime, P. Nagarajan, thanks for the support. To Jeff Harper and Craig Thornley, I express kudos for their ORBS help; (Yes Jeff, ORBS is useful.) Finally, the faculty, staff and colleagues in the CIS department, have all contributed in making my stay here at the University an educative and pleasant experience.

DEDICATION

To my parents and friends, keep listening to the music.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION	1
Performance Versus Compositional Systems	2
Problem Basis	3
Problem Statement	5
Proposal	6
Solution Constraints	11
Summary	12
Introduced Concepts/Definitions	13
2. ENSEMBLE SPECIFICATION	15
Processes of a Musical Agent	15
Analysis	16
Planning	16
Performing	17
Intermediate Structures	17
Intentionality of a Musical Agent	18
An Example of Musical Agent Processing	19
Summary	21
3. LIMITED ENSEMBLE IMPLEMENTATION	23
The Choice of Language	23
Representations of the Limited Ensemble	25
Representation of an Agent	26
Overall Construction	27
Control Flow	29
Implementation of the Processors	29
Intermediate Structures	30
Internal State Variables	32
Heuristics	41
Configuration	42
Input/Output Information	43

Gestures	43
Musical Events	47
Tying It All Together	52
Summary	55
4. CONCLUSIONS	56
Results	56
Related Systems	57
Compositional Aids	57
Performance Systems	58
Further Work	59
Intentionality in Note Selection	59
Issues in Synchronization	59
Event Synchronization	60
Conflict in Interpretation of Intentionality	61
Listening to the Future	61
 APPENDIX	
 A. SPECIFICATION OF INPUT/OUTPUT ITEMS	63
 B. RULES	69
 BIBLIOGRAPHY	77

LIST OF FIGURES

Figure	Page
1. A Sample ENSEMBLE	8
2. A Musical Agent	9
3. Processers and Structures in a Musical Agent	16
4. An Example of a Musical Agent Processing	20
5. A Limited ENSEMBLE	26
6. Internal Structure of a Musical Agent	28
7. Beliefs	31
8. Intermediate State Variable Changes	33
9. Tempo, Location, and Meter	34
10. Tempo State Transition and Intermediate Tempo State	37
11. Dynamics State Transition and Intermediate Dynamics State	38
12. Modes of the MA	40
13. Input/Output Information (Gestures and Metrics)	44
14. Input/Output Information (Musical Events)	45
15. An Example Gesture	47
16. An Example Musical Event (A Note)	50
17. Another Example Musical Event (A Chord)	51

LIST OF TABLES

Table	Page
1. Intentionality in a Musical Agent (Belief Table)	18
2. Sketches and Their Values	41
3. A Sample Musical Agent Heuristic	42
4. Description of Metrics Values	43
5. A Description of Gesture Values	46
6. A Description of Values of Musical Events	48
7. A Description of Values of Musical Events	49

CHAPTER 1

INTRODUCTION

The operating mechanism [of the Analytical Engine] ... might act upon other things besides numbers, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the Engine. Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the Engine might compose and elaborate scientific pieces of music of any degree of complexity or extent.

– Ada Augusta, Countess of Lovelace,

(Kassler & Howe, 1980)

Composition, improvisation, and performance of music are extremely complex tasks in which the grammar of the languages are poorly specified. We know of definite characteristics of sound. The elements we deal with in describing sound tend to describe notions of pitch, timbre, rhythm, amplitude, and harmony. We can also formalize stochastic and deterministic reasoning to the analysis of these characteristics in a musical context, however we can only scratch the surface at realizing the underlying representation of the musical content, and we have barely taken any advantage of contextual information in the generation of composition, nor general compositional methods. The equivalent analogy is the belief that we could produce well composed text given rules of English grammar, and words categorized by items in the grammar to apply to it.

In the performance of music itself, we deal with many “unseen” activities –

activities hinting at what is upcoming in a performance. In the realization of music performance in systems to date,¹ we have dealt with “surface level” features of the music: the discrete mapping of timed musical events. But there is so much more to music than notes, their harmonies, and rhythms. If this were not the case, there would be no difference between any two individual interpretations of some written score.

Performance Versus Compositional Systems

Most systems being developed in computer music research address representations for automated composition. Traditionally, composition systems are handled as “single agent” systems: since only one person writes the composition, only one source of input is required – the general body of knowledge of the one composer. I tend to believe that generative processes of music require multiple agents. This is a belief that is highly motivated by considering systems whose purpose is directed more toward the performance rather than the composition of music.

The view of separating composition and performance systems allows us to specify that the creation and progression of music is in fact based on many inputs, internal knowledge as well as environmental contexts. Performance systems by definition imply a complete creation (system outputs) including the notes, how they are articulated, dynamics, and all associated visual and audible contexts of a performance.

When a performance is established by a single performer (even when there are no other people listening), the performer plays the roles of music producer, and music listener/evaluator. If the currently produced music is aesthetically pleasing (in some abstract manner) to the listener/evaluator based on compositional criteria

¹ Here I stress music performance systems. I believe many compositional systems are the same way. I base this statement on the systems I have reviewed in preparation for this thesis: MUSICOMP, PLA, FORMES, COMPUTER IMPROVISATION, and FLAVORS BAND.

and emotionally pleasing feedback, a given probability for the repetition of that pattern will be increased. This is information that is transferred back to the music producer. Compositional systems do not provide this viewpoint.

Compositional systems do not provide descriptions for the intentionality of a given division of music (i.e. a song, phrase, bar, etc.). This becomes necessary in a multi-agent system in which it is time-critical to evaluate a grouping of notes to determine what it means, and how we can add to it (see the next section).

Problem Basis

The basis for this thesis is that when we are given any form of communication, there is at least one very specific "intention" behind it. This intention is the most important part of the communication. It is the meaning of the message we are trying to get across to our audience, whether the communication is in the form of speech or music. This theory of intentionality in music is similar to the intentional basis of conversations discussed in (Searle, 1969, Allen & Perault, 1980). Curtis Roads explains in his survey of artificial intelligence research in music (Roads, 1985, pg 168), "A computer system that could track tempo or listen for other musical cues could bring flexibility to the performance of music."

The basis does not stop here however, as a communication act is a two process function. The "generator" represents the intention within the framework of some language (once again where the language may be musical, or some spoken language) and the "interpreter" uses skills of reasoning to understand the message on many different levels - literal formation, visual gestures, tonal gestures, etc. In some processes, the generator may actually be the listener (which is what I believe happens in self-inspired composition of music). My point is that the notes or words are inadequate for describing the intentionality behind it, an intentionality that only may be derived by a listener drawing upon clues. The clues that I refer to may be our knowledge of the vocabulary, or inflections in speech. This is analogous to dynamic

or rhythmic emphasis in music, visual gestures such as nodding to indicate a soloist, or even a common emotional background. The intentionality arrived upon by the listener may only be a near approximation of the intention the initiator expressed, but this is the one thing that makes different musicians perform differently.

To exhibit my point, let's examine the following situation in English. If I describe a situation by saying "That's just great," it could ambiguously imply at least three specific meanings. This shows us the fundamental problem with exploiting intention from a static language, too many contextual clues have been stripped in the mapping of the performance to a static language. If I give you a visual clue such as a surprised look on my face, and a somewhat light and happy sounding intonation in my voice, you now can more plausibly resolve the ambiguity of intention of the description as a sincere communication of acceptance and acknowledgement. If the clues had been different, such as lowered eyebrows, a not so cheery but more cynical smile, and a gradual lowering of pitched sounds as the sentence progresses, the intention becomes one of sarcasm in which the meaning actually is that "that is not so great." A look of puzzlement and a lowering followed by raised pitch as the sentence progresses including stress on the word "just" may indicate a questioning of whether "great" is the correct word to describe the situation.

In a more abstract way, music performance is accomplishing the same means by relating individualized intentions to the listeners. In effect, the act of the performance is the function which creates a state within the world of the listener, the performer, and their shared worlds (Appelt, 1985). The musical tensions created by Ravel's "Bolero" are contextual musical clues in the form of a snare drum's hypnotic and repeating ostinato, a musical tension in that the pattern is constant and persistent. More dramatically, in the context of a jazz ensemble playing, clues to initiate activities of other performers may be "hidden" within the context of the produced music. For example, a drummer may signal the ensemble that a shift in time will occur on the next upcoming bar division, a very significant intention

hidden within the context of an "agreed" signal - such as an unusual grouping of bass drum kicks.

Agreed signals present other problems in that very often they are misinterpreted or ignored - a hard problem in that this leads to communication breakdown. Communication breakdown may not even be noticed until the intended activity has already started, and discrepancies from normal patterns are noticed by the other performer. In my theories, I shall avoid misinterpretation of clues as much as possible as it may be too difficult to deal with at this point, and will only detract from my focus.

In another perspective of these intentions, we may have very different groupings of some communication that may mean exactly the same thing. In English, three sentences may very well have the same intention: "Let's shoot some hoops." or "How 'bout a game of round ball?" or "Want to play a game of basketball?" Similarly, in music, we have very different literal forms of musical groupings with exactly the same intention, whether the phrases variate to allow balancing of syllable stress in a lyrical song, or just variate on a given theme.

Problem Statement

Previous forms of automated musical systems take the perspective of describing "language" formalisms to justify the "literal output" in the form of scored or performed music. My claim is that these systems can never attain any resemblance of the creative process without justifying the intentional basis for those processes. If we do this, we can open up new possibilities for creation of automated composition and performance system. To do this, the first job would be to provide a language for the intentions (not an easy job).

Intentions may be a mapping of an emotional belief or theme, a mapping onto a given musical grouping. The emotional theme may map a musical relation or pattern exactly the same as, or similar to something that we've heard before. We often tend

to lead up to these patterns through some connective mechanism, usually in the form of established musical rules. On another level of cognition, after collecting these abstract notions, we may quickly parse our activities to validate their syntactic form. This happens certainly from the standpoint of composition, but in a “real-time” improvisation situation there often is not enough time for this (my own basis for why there is often musical “error” during improvisation, inspired by the beliefs of Bradley Goodman (1986) in reference identifications and failures in English).

If this could be done, we could then relate our intentions to characteristics that indicate these intentions. On another level, these characteristics may be thought of as intentions also as they are giving commands to lead to a musical event – an entirely different meaning than the actual musical message but one that provokes interpretation of the musical intention nonetheless. Previous musical compositional systems ignore any concept of intentionality, giving mechanical representations of musical knowledge and static descriptions of temporal relationships between notes. In my theory, there is still room for these previous types of systems as their knowledge collection would thus provide the framework for mapping down the creation of a performance from an intentional language, as well as the mapping up process of the listener in order to understand some abstract version of the intent of the performer.

Proposal

The significance of the problem is shown in the previous discussion. In this thesis I will exhibit some principles of this theory by constructing a “toy world problem” to display some of these intentional and communication properties. My intention is not to discover new forms of creativity (nor assume mastery of modeling the creative process), nor is it to produce a new product in the form of an intelligent sequencer. The purpose here is to establish a richer musical representation for the performance of music through the interpretation of intentional content.

ENSEMBLE, is a group of modeled musicians, each playing a different instrument and adding a different characteristic to the performance. The ENSEMBLE will ultimately produce a performance of limited improvised music, based upon intentions interpreted from the clues generated by the other musicians. Figure 1 shows us a four piece ENSEMBLE of musical "agents."

The entire ENSEMBLE is driven by musical data that is added or removed by each agent. Data in this definition refers to visual or musical clues produced by each agent. This system is what I envision as a complete system. The actual programming portion of this thesis will finesse many issues, including the activities of three of these agents. The thesis will concentrate on the creation of one agent, driven by a dumb agent that takes the place of the other three.

Figure 2 shows us an individual agent. What is represented in the figure as a single knowledge source is actually two components of knowledge that comprise the entire inference heuristics of the structure. Interacting musicians derive their knowledge of what to do literally in a performance from two areas: Common Musical Knowledge (CMK) - which consists of general musical context knowledge, and given knowledge about what the other musicians do; and Specific Musical Knowledge (SMK) - knowledge that is specific to the performers own individual performance and instrument.

The CMK is knowledge that lets us perceive changes in the perceptual flow of a performance. This could be as simple as increasing the dynamics in our own instrument to match the levels of other instruments, or as complex as determining a solo negotiation through visual clues.

SMK is the knowledge that allows us to determine the more "literal" aspects of our performance - the selection of notes and/or rhythm patterns, variations on these patterns, etc. In SMK, we draw upon recent musical information in order to reproduce our version of the same intent, as well as our prior knowledge of "patterns" and "processes."

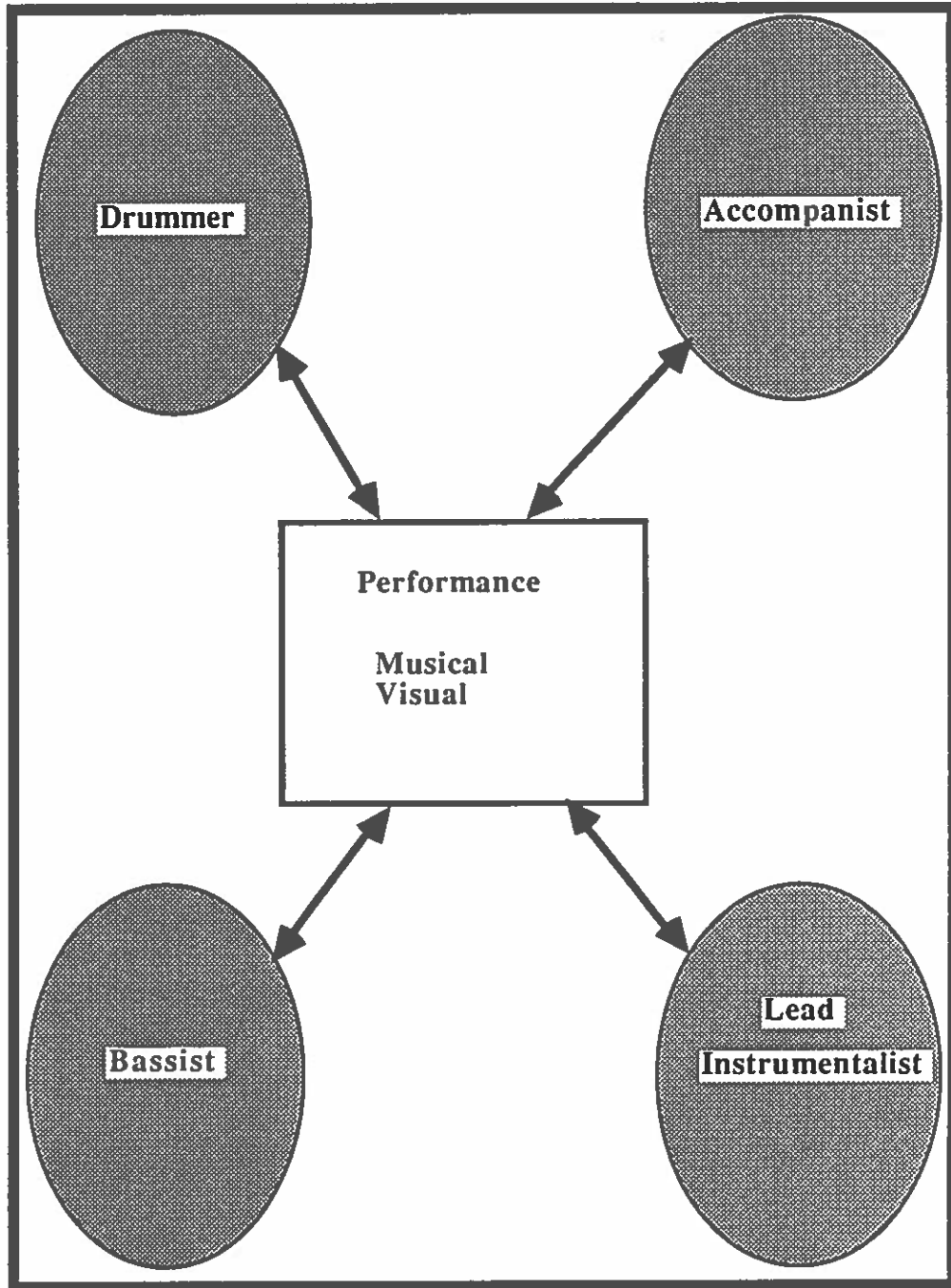


Figure 1: A Sample ENSEMBLE

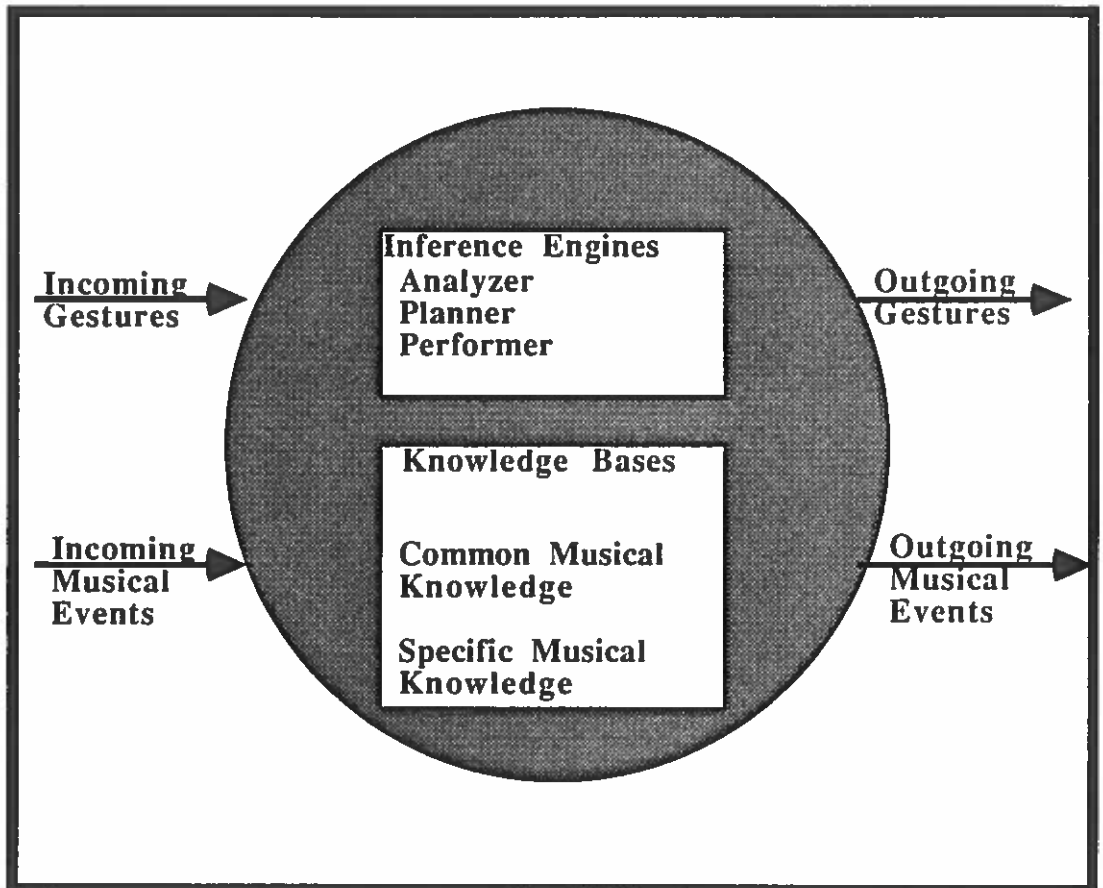


Figure 2: A Musical Agent

The causal relationship that I am trying to specify here is that we are given musical and visual clues, as well as information sources of musical patterns, to produce inferences of intention as an intermediary state. The intentional beliefs that performers infer are both intention of a compositional nature, and intention of action to change a flow of musical events. These inferences are further used along with characteristics about the performing agent and performance itself to drive tasks. These tasks are viewed as the intention the performer is trying to communicate to the other performers. This intention will ultimately be used in the production of the agents musical flow, as well as possibly producing some kind of clue for other agents either in the musical flow or visually.

I would like to define "intention" as I use it in this thesis. Intention comes in two forms: Thematic Intention - the intention behind a phrase in terms of its melody, rhythm, or harmony, and Progressive Intention - the intention that we wish another performer to perceive in order to cause an action of a particular manner, thus affecting the progressive nature of the performance. Though it is mostly the thematic intention that is the most difficult to realize and exploit, it is also the most significant. For brevity and a more concise scope, I will finesse issues of thematic intention as much as possible. The progressive intention is also significant, and the area I wish to concentrate on. Progressive intention strictly refers to the information modifying the performance alone, and is an essential feature for two or more performers to produce a coherent performance. The mechanism that determines these intentions is the inference.

Inferences determining intentionality are implemented by rules that act upon a flow of information into the agent. An interpreter may match recognized patterns to affect a state of a performer, which is later evaluated in order to produce the expected action. For example, a given piano part may be scanned for the low register notes. These are matched against a chord fact base to determine which note is the root. The analysis for roots over a given time will let us determine

the sequence of roots. A sequence of roots may be a given relationship of notes progressing to a certain pattern. This progression can be matched to known bass progressions for a steady state pattern of bass events. We may further vary this progression within the confines of the primary downbeats to produce some dynamic bass pattern capable of being used in real time until a break of steady-state activity is required. At the same time, we can match our dynamic levels and speed to the piano's overall performance. We can also determine if anyone is commanding us to plan for varying musical intention in the near future, either through visual clues or an unexpected steady-state pattern of performance. Thus rules may infer states from which a musical action may be accomplished (described in more detail in Chapter 2), where an action may also be to ignore a given intention and proceed with steady state events.

Solution Constraints

The implementation of an ENSEMBLE is extremely complex. Issues in parallelism and multi-agent negotiation must be considered in the ultimate system. I wish to specify my project portion of this thesis as the limited ENSEMBLE. The limited ENSEMBLE is concerned with the specification and creation of an agent, and isolating intentions in performance. The performance of the other agents will be partially simulated by a pseudo-agent (mainly a driver for the explicit purpose of running the model agent). The model agent will be referred to as the MA (musical agent), while the driver will be referred to as the PA (pseudo agent). The MA will react to "planted" musical and visual clues from the PA, as well as producing clues of its own. The agents will not be blind of possible scores, as they will know chord progressions of the events. I feel this is warranted, as performers in a real ensemble often agree ahead of time on a standard progression of chords upon which to build their improvisation. In this limited model, the MA will also have a melody pre-chosen for it. A further stage of development would be the planning of

improvised material within a MA. I will create the structures to make this possible, but will not attempt an improvisation scheme as part of the thesis project. The information that an agent processes is incomplete information, but enough for us to take advantage of for interpreting intention, and modifying our own planned steady state of events.

Summary

In this introduction, I have provided a cognitive framework, and a brief proposal of a project to exploit some theories of intentionality in performance. My goals are to:

1. Isolate and exploit intentionality in music performance by (a) developing a language of clues and the intentions they are associated with, and (b) developing a language which maps intentionality into the associated action upon planned musical events, and eventually the performance.
2. Establish a framework for musical events and gestures that will support goal 1, and provide a basis in this model for improvisation.
3. Develop the concept of a musical agent such that it may be used as the groundwork for a system of interacting musical agents.

The ENSEMBLE model is the group of musical agents that interact by interpreting musical events and gestural clues in the performance, and add to it with their own musical events and gestural clues. My contribution to this theory is the establishment of a limited model of ENSEMBLE with the following refinements:

1. The concentration on creating a single musical agent (MA), driven by a pseudo agent (PA).
2. The reduction of ENSEMBLE to a two agent system, finessing the problem of group consensus and intentional conflict.

3. The limitation of this exercise to the exploitation of intentional beliefs, finessing the issues of improvisation, but providing a basis for it.

I have also specified types of knowledge in the performance domain (CMK and SMK). A separation of intentionality in the domain of musical performance was specified. Thematic intentions specify the intentions that are the basis for the compositional nature of music (the selection of the notes). Progressive intentions specify the intentions that motivate changes in the performance of music. In the next chapter, I will develop these concepts into the specification of the limited ENSEMBLE.

Introduced Concepts/Definitions

Common Musical Knowledge (CMK): A body of knowledge that describes all of what an agent knows about general musical rules of musical performance, and rules of how other individual agents behave in certain contexts. These rules are most commonly used in the determination of musical intention.

Composition System: A system whose primary focus is to generate musical events, events considered from a single point of view in a methodical fashion.

Musical Action: The combination of musical events and visual clues produced as a singular description.

Musical Agent (MA): A generic reference to one object-oriented model of a performer.

Musical State: A view of current music and gestural events, as well as a description of an agents view of itself and belief in other agents (environment) at any discreet time during a performance.

Musical Communication Breakdown: A missed or misinterpreted musical intention from a progression of musical actions.

Musical Event: The static description of a note, chord, or rest, with descriptions of duration, dynamics, and any applicable articulations to the specific instrument of the agent.

Performance System: A system whose primary focus is the presentation of events that compromise a musical action, including the production of the music and all other actions that are considered for that production.

Performance System: A system considering stimulus from external sources to produce musical and gestural events in a performance.

Progressive Intention: A description of a performance objective derived from analysis of our own intention, or the analysis of performance of other agents in the derivation of intention.

Specific Musical Knowledge (SMK): Knowledge that allows us to map an intention within a certain musical state into a new musical state via the production of a musical action specific to the individual performing agent. These rules are also commonly used in the production of progressive intention.

Thematic Intention: A description of a musical objective derived from progressions through musical states.

CHAPTER 2

ENSEMBLE SPECIFICATION

In the previous chapter, I discussed the primary goal of this thesis, the exploitation of intentionality in the performance of music. I discussed the evolution of a multi-agent system called ENSEMBLE, and my limitations to the model, including the implementation of only a single agent and a driver agent, and finessing the issues of improvisation in the agent. In this chapter, I will develop these concepts into the specification of a MA (musical agent). This will be done by giving a high level description of the processes of a MA, the main structures of the MA, and a brief example of the control flow of a MA.

Processes of a Musical Agent

The MA is an agent that takes information in the form of musical events and gestures. Three distinct phases of processing occur. These are “analysis” of the events and gestures, “planning” of actions to occur, and “performing” of the actions when they are due to be produced. In the MA, this specifies a processor for handling each of the processes. They are subsequently called the analyzer, planner, and performer (see Figure 3). Each of these processes are sequential, and provide information for the subsequent process. Internal state information to the agent (i.e. belief in tempo, dynamics, etc) is used in all of the processes.

In the following discussion I will be referring to the term clues. A clue is one of the data items in the domain, either a musical event or a gesture.

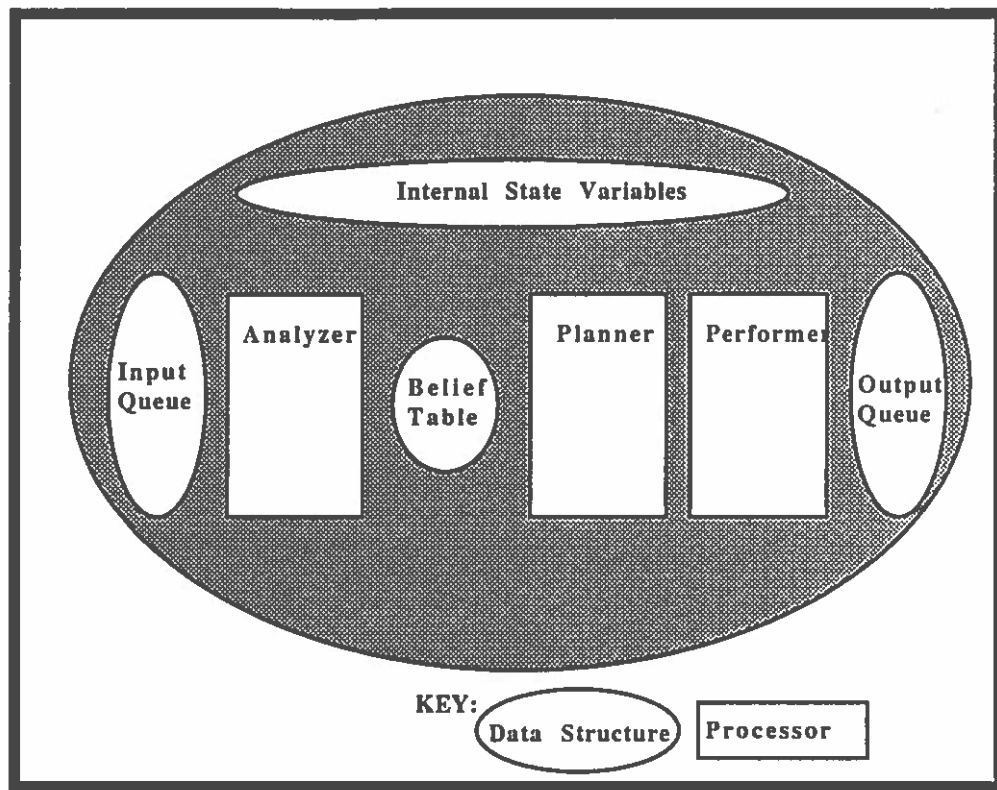


Figure 3: Processors and Structures in a Musical Agent

Analysis

Analysis is the process of interpreting clues to either change the state of the agent, or to change levels of belief in intention. Analysis occurs as the first stage of processing, as seen in figure 3. If a clue does not have any significance to the analyzer, it is simply discarded.

Planning

Planning is the process of interpreting significant beliefs, and reacting to them by changing internal state, initiating an activity upon currently planned events in the performer, or adding a gesture to the planned events in the performer to promote the MAs intention in the performance. Planning follows analysis, as it is driven by the beliefs propagated in analysis.

Performing

Performing is the process by which we modify our planned events according to direction from the planner. The performance process is relatively “dumb” in that it either does what it is told (i.e. change the timing of all the events you currently know), or waits until a time occurs in which one of the clues that is planned may be performed. Performance follows planning in most cases, but an action inside the performer process may be triggered as a direct result of planning.

Intermediate Structures

The processes described above represent the major functionality of the MA. These processes use information internal to the MA, and stored in various intermediate structures. These structures may be divided into internal state variables, and shared structures. Internal state variables describe state information that the MA believes, and is accessible to all processes. This includes state beliefs like location metrics (what bar, measure, beat, and tick (beat subdivision) the MA is in), tempo, dynamics level, and state of play (in relation to lead or normal play). Shared structures contain information used by specified processors, but not by all processors. Figure 3 displays these data structures in a MA. Shared structures include the input and output queues, and the belief table.

Information from the outside world, or performance outside of the MA created by other agents, is queued into the agent where it remains until the agent is signaled to start processing. The input queue is just an intermediary structure between the outside world and the analysis process. The action of analysis concerning beliefs is to modify our beliefs in an intention based upon the previous input events. These intentions are subsequently interpreted by the planner, so the planner must have shared access to this information. This motivates the necessity for a structure shared between the analysis processor and the planning processor for the belief of intentionality. This structure is known as the belief table (table 1). The performer

INTENTION	MEANING
Adjust Dynamics	The dynamics of my planned events should be modified to the level dynamics of the incoming events
Adjust Tempo and Location	The dynamics of my planned events should be modified to the temp of the incoming events
Receive My-Lead Request	Recognize somebody wants me to take a solo
Receive My-Lead Acknowledge	Recognize an acknowledge to my request for a lead
Receive My-Lead Nack	Recognize a denial of my lead request
Request My-Lead	Request for a solo
Return My-Lead Acknowledge	Return an acceptance of a solo request
Return My-Lead Nack	Deny a request for a solo

Table 1: Intentionality in a Musical Agent (Belief Table)

produces events and gestures used in the performance. These are queued into an output queue, which serves as intermediary storage until the performer process is complete. At this time the contents of the output queue, in this case the musical event that will be performed, is added to the performance.

Intentionality of a Musical Agent

The MA is an automaton that can change various features of its performance according to externally interpreted clues. The function of the performer process is quite simple in that it only needs to react under the circumstances that an event has reached performance time, or the planner has initiated a task within the performer. The functioning of the analyzer and planner are more data driven, and the key piece of information to their functioning is the intentionality. The intentions are literally beliefs that a task should be initiated. Table 1 describes the intentions the MA understands. There are two types of intentions that a MA understands, absolute belief intentions and incremental belief intentions. "Adjust tempo and location" is an example of an absolute belief because we either believe that we are on time, or

we don't. Negotiations such as "receive my-lead request" are incremental because the more clues we interpret supporting this intention, the greater our belief is. All beliefs have a threshold associated with them. The planner reacts to intentions only when they have been incremented to surpass their thresholds. The planner may increment beliefs as an activity triggered from the interpretation of another belief that has surpassed a threshold. An example of this is when we have surpassed the threshold for the request of a lead, the action the planner specifies is to increment the belief in returning an acknowledge (ACK) or a no-acknowledge (NACK). This belief in turn triggers the planner to initiate an activity of placing a gesture reflecting the ACK or NACK within the planned events of the performer.

This scheme of belief management is an additive model, and completely a hack for the limited ENSEMBLE model. A better model of belief management would be an assumption based truth maintenance system (ATMS) as described in (de Kleer, 1986). This would allow the possibility for adding and retracting beliefs with the provision that all assertions that pose inconsistencies in the beliefs cause the retraction of those inconsistencies.

An Example of Musical Agent Processing

This section displays the control flow of the processes with an example of tempo change. Figure 4 shows a representation of the structure with the gesture downbeat nod. From the performance, MA takes any current events or gestures and places them in the input queue when they are created. In this example, a gesture is placed into the input queue of the MA. The gesture is a data item characterized by its implementor (head), its motion (nod), its emphasis (strong), and its timing information (which is a relic of the performer process of the agent that produced the gesture). A control function within the MA waits to be signaled by an external synchronizer when the agent is to start execution.

When the MA control function begins to process its events, it looks in the input

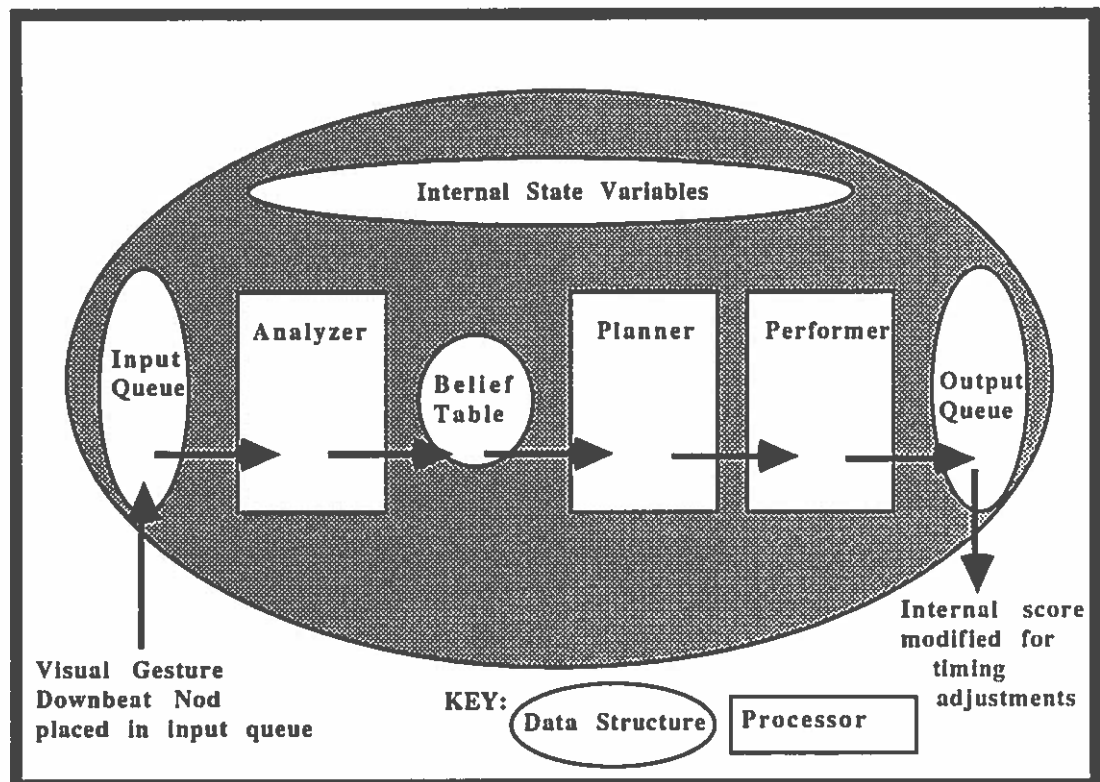


Figure 4: An Example of a Musical Agent Processing

queue to find any information to give to the analyzer. The analyzer then takes the information and matches it with what it knows about nod gestures and the internal state of the MA. In this particular example, the nod is reflective of a beat in the other agent, and it is earlier than when our state variables describe a beat should occur. This causes the analyzer to look up in a state table (MA internal state variable information) what the MAs current tempo state will be when it is given a BEAT-EARLY for its current tempo transition. Some of the states described in the state table are intermediary, and can result in no action other than the change of internal tempo state and registration of beat discrepancies. Other states are terminal and cause the analyzer to react by increasing the belief in the intention of “adjusting the tempo and location.”

The planner takes control when the analyzer has finished processing its inputs (in this example only the gesture). The planner notices that a belief has surpassed its

threshold for “adjusting the tempo and location.” From the internal state variables where previous timing discrepancies were registered, a new tempo is calculated, as well as when the next beat should occur. The planner invokes the performer to return the internal state variables for tempo and the beliefs in tempo adjustment back to a normal level. The performer then adjusts the events for a newly calculated tempo and location. When the performer is done with this task, the control is returned to the planner which reacts to any other beliefs that may have surpassed their threshold.

When this is done, the control is passed to the performer whose primary task at this point is to take any of the planned events that are to be performed at the current location in time, and place them in the output queue. The performer is finished, and returns control to the MA control function, which then takes whatever is in the output queue and places it in the performance. Control is returned to the external synchronizer that initiated the processing.

Summary

The MA is comprised of three processes: analysis, planning, and performing. Each process has an associated processor with it, referred to as the analyzer, planner, or performer. The analyzer reacts to input gestures and musical events, producing a change in internal state, or belief. The planner reacts to beliefs that have surpassed their thresholds, and produces either a change in state, a change in belief, a change in planned events (through the initiation of a task in the performer), or combinations of any of these. The performer reacts to initiated tasks, and performs clues which are time-active (clues that have a planned time that is the same as the MAs currently believed time).

Beliefs are the measure of an interpreted intentionality. Some are considered absolute, while some are considered incremental. All beliefs have an associated threshold that the planner may react to. Beliefs are stored in a shared structure

between the analyzer and the producer called a belief table.

In the next chapter, I will describe my implementation of the limited ENSEMBLE and the translation of this specification into the ORBS rule-based programming language.

CHAPTER 3

LIMITED ENSEMBLE IMPLEMENTATION

In the previous chapter, I discussed the functionality of a MA, describing three distinct processors: the analyzer, planner, and producer. These processors use facts from internal state variables (i.e. tempo and dynamics), and intermediate structures (i.e. belief table, input and output queues). The processing in the limited model involves interpreting the belief in an intentionality from the clues (musical events and gestures) in a performance. The belief is then analyzed to determine if it is strong enough to warrant action. This action may change beliefs, internal state variables, or planned events that will be performed. The outcome of this processing is the addition of clues to the performance.

This chapter develops representational issues of my limited ENSEMBLE model. The choice of the programming language ORBS is discussed. The implementational issues of mapping the previously discussed processes, structures, and clues will also be discussed.

The Choice of Language

Music is a highly symbolic language, therefore a symbolic processing language is needed for the specification of operations on symbolic items. The object-oriented paradigm carries this viewpoint one step further by allowing the direct translation of domain concepts into classes of objects, described by their attributes, and methods that act upon them. A rule-based system features the ability to make "inferences" based on the state of a system.

The limited ENSEMBLE is a system that must have a symbolic language in order to represent the clues descriptively. A rule-based interpreter is also necessary

to allow the inference of state and planning described previously. ORBS II¹ conveniently provided these features, with the addition of structures similar to objects.

The ORBS LISP interpreter is the basis of the system, and features a subset of COMMON LISP. The syntax allows operations on atoms, integers, floating point numbers, pairs, lists, objects (in the ORBS definition of objects), and rules. The object-oriented environment allows a more meaningful conceptual basis for programming, as it allows the direct translation of objects in an ontology to their respective code parts. The idea is that everything that we believe to exist in our conceptual world is represented by a separate entity. These entities gain attributes and functionality from classes that the entity belongs to (inheritance). These entities are instances of objects, inheriting methods and attributes. Methods (the functions private to an object) are invoked by messages sent to an object.

The rule-based system is a logic interpreter that allows the insertion of facts

¹ The Oregon Rule Based System, or ORBS was developed under the direction of Stephen Fickas at the University of Oregon as a new programming language. Its design was based on a multi-paradigm language model that incorporated features of LISP style syntax, object-oriented programming, and rule-based interpretation. The first generation of the language was written in LISP and mainly featured rule based programming similar to PROLOG. ORBS II was developed over the summer of 1987 by Jeff Harper, and incorporated the aforementioned features, improving basic design, speed, and portability through its implementation in C. Part of my programming included modifying ORBS II to include hooks into the MACINTOSH II toolbox to the Sound Manager routines. This seemed to be the most promising choice, as I could preserve my ORBS representations, and only modify a channel-constant to choose an output stream of either internal sound, or MIDI (International MIDI Association, 1983) data. The MAC II also featured Sound Modifiers, routines that could be linked to channels to intercept the flow of data, and modify it before passing it on. This seemed useful for modifying changes in dynamics, pitch, or duration of a stream, logically isolating these modifications. In a performance system, this seemed to be a necessary abstraction. The majority of development was done on a Sun workstation. Object inheritance was added near the end of the limited ENSEMBLE development. ORBS II was still relatively new during ENSEMBLE development. It had been tested only once before on a larger scale project (a bridge playing model).

into a database. These facts are unified with rules in a rule base. The rules provide "IF-THEN" styled condition-reaction specification of heuristics. A rule fired in ORBS is equivalently known as an inference. The inference engine (interpreter) unifies all known facts in a factbase associated with the interpreter to all known rules in a rulebase also associated with the interpreter. The set of rules pending to fire form the "conflict set" from which any or all rules may be chosen to fire. The test for a rule firing may be any boolean expression, and the action of a rule firing is any group of expressions.

Before proceeding to the representations of my system, I wish to clarify the term object. An object refers to a data structure in ORBS that has a set of instance variables, and methods associated with it. Objects in ORBS (at the time of project development) do not have inheritance, so there exists no hierarchy of classes from which attributes and methods are common to groups of objects.

Representations of the Limited Ensemble

This section briefly discusses the highest level of implementation in the limited ensemble model. This is significant because it displays how synchronization of the overall system works.

Figure 5 shows a limited ENSEMBLE, featuring a MA, a PA, and a conductor. The conductor is an object whose sole purpose is to send ticks to each agent in the ENSEMBLE in order of their execution. This means that one unit of real time is not complete until all agents have received their tick, and completed an execution cycle. In this context, I define an execution cycle of a MA as the complete processing of the MA control function (described in the next section under MA control). The conductor is shown in figure 5 sending ticks to each agent. This is occurring synchronously in an order determined by the conductor.

The PA is built from the same definition as the MA, but only has enough rules to allow it to produce predefined clues that are placed within the PA at system

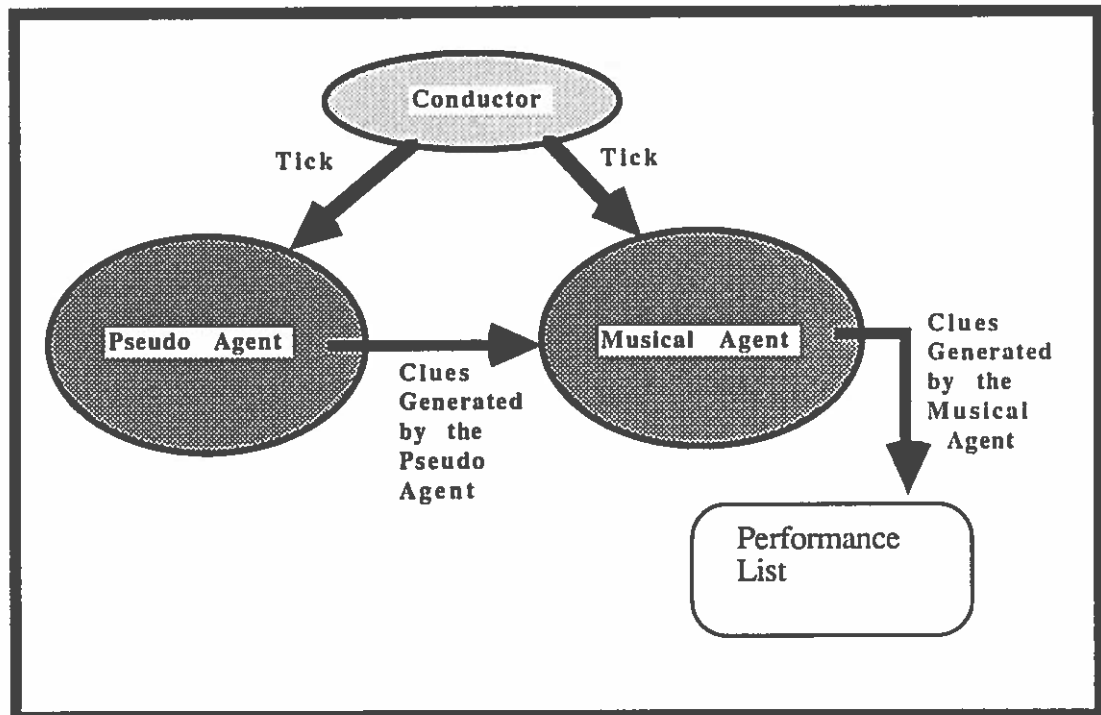


Figure 5: A Limited ENSEMBLE

initialization time. The conductor will thus send a tick to the PA, who in turn may produce some output that is considered the existing performance that the MA will process as input. Upon completion of the execution cycle of the PA, the conductor will send a tick to the MA. This sequence of ticks to each agent repeats for a predefined number of cycles. The PAs performance is placed into the input queue of the MA. The MA starts processing the queued clues only when it receives the tick from the conductor. When its execution cycle is finished, the MA puts its output clues into a performance list. This is just an ordinary list accumulating clues.

Representation of an Agent

The following section details the implementation of the MA. First, the overall construction will be discussed. This will be followed by a description of the control flow, the processors, intermediate structures, and internal state variables.

The agent is an automaton with three purposes in relation to its performance

of clues:

1. Noticing and adjusting tempo of performed events according to tempo variations from what the MA expects. These variations are encoded in gestures.
2. Noticing and adjusting dynamics of performed events according to dynamics variations from what the MA expects. These variations are encoded in musical events.
3. Negotiation of Lead play according to interpreted and initiated gestures.

These purposes motivate the implementation of rules and internal state variables.

Overall Construction

The MA is literally an ORBS object. As discussed earlier, an ORBS object is defined by its instance variables and associated methods. Figure 6 shows a MA, comprised of its internal state variables, intermediate structures (input queue, output queue, and belief table), and the three processors. Each internal variable has its own designated slot within the MA. The intermediate data structures and processors also reside in slots of their own. This allows us to directly access them by reference to a slot name. For example, it is possible to perform a function on a belief by sending a message to members of the BELIEF-TABLE directly.

The MA has a few methods defined internally. These include methods for taking clues from the performance and placing them in the input-queue, putting clues from the output queue into the performance, initially configuring the MA, and the control function.

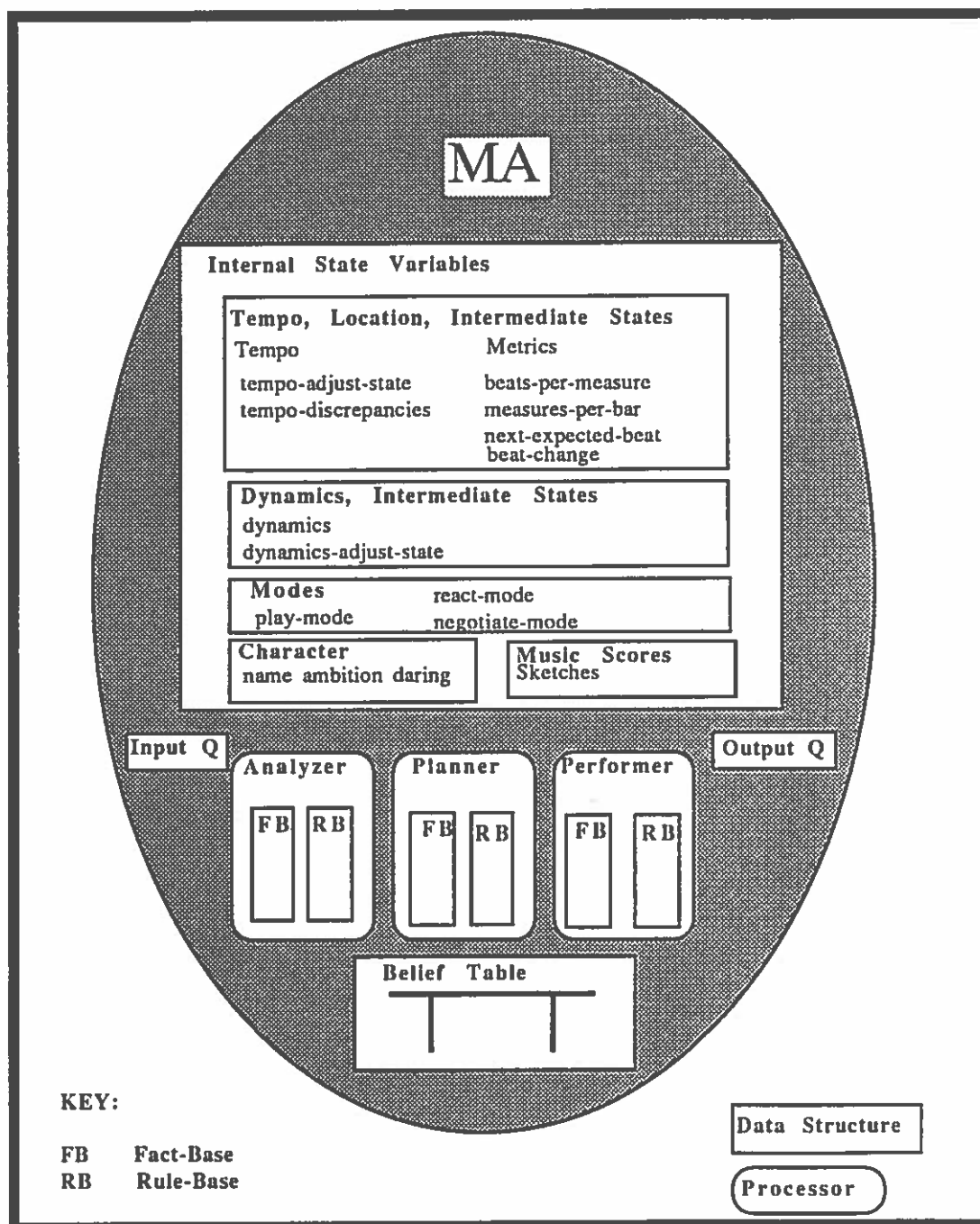


Figure 6: Internal Structure of a Musical Agent

Control Flow

Control of execution in the MA is determined by the control function. Literally, this function is implemented by a method named "tick" such that the MA becomes active when receiving a tick from the conductor. The method that puts clues into the input queue is activated by a message sent from the PA, but this processing is not considered part of the execution cycle of the MA.

The control function starts by taking all clues in the input queue and imports them into the analyzer (importing into a processor discussed in the processor implementation section). The control function also increments its own time and location internal states. The control function then tells the analyzer processor to start, and waits for the analyzer to relinquish control (this is a synchronous activity, the processors are not truly asynchronous processors). The analyzer then removes all clues from its fact base. When the control function regains control, it starts up the planner in the same manner. Once again, when control returns from the planner, the performer is started. The difference with starting the performer process is that some other control information needs to be specified for its start. This information tells the performer to use only the rules pertaining to moving clues that are currently active into the output queue. The output queue is finally emptied into the performance, ending the control function and the execution cycle of the MA.

Implementation of the Processors

The processors are implemented as ORBS rule-based interpreters. A rule based interpreter in ORBS is its own object, composed of several objects. Figure 6 shows the two most pertinent objects inside a processor object: a fact base, and a rule base. The fact base is the storage area for information upon which all inferences are made. The rule bases contain the active heuristics of each processor. These bases get their information via an "import" message containing the rules or facts

to be added to the base. Likewise, they may be removed via a "delete" message. At system configuration, each MA is given a set of heuristics from a storage area known as user-preferences (see system configuration section below). This set is divided by rules from CMK and SMK. Each one of these sections is subdivided into groupings relevant for each processor (ie a group for the analyzer, the planner, and the performer).

As well as having clues imported into the fact base of each processor, all internal state variables, and the intermediate structures that are relevant to the processor are also imported. This is the equivalent of passing a reference pointer to these items.

The processors are given a message to start, at which time all possible rule firings occur. In the planner there is a noticeable difference in that some of the heuristics specify that the action is in fact a task that is to be accomplished by the performer. This is a case where the performer is told to turn off all heuristics except for the one specified by the planner action. The planner then starts the performer, and receives its control just as if a function was called.

Intermediate Structures

The input and output queues as previously mentioned are stored in slots of the MA. Their structure is quite simply a list. The list can have musical events or gestures added or removed just as simply as any other list operation.

The belief table is a list of objects of type belief. These beliefs (see figure 7) contain an intentionality name, its current weight of belief, and the threshold that makes the belief significant. The belief has methods associated with it that set and reset the weight of the belief, as well as tell us whether the threshold has been surpassed.

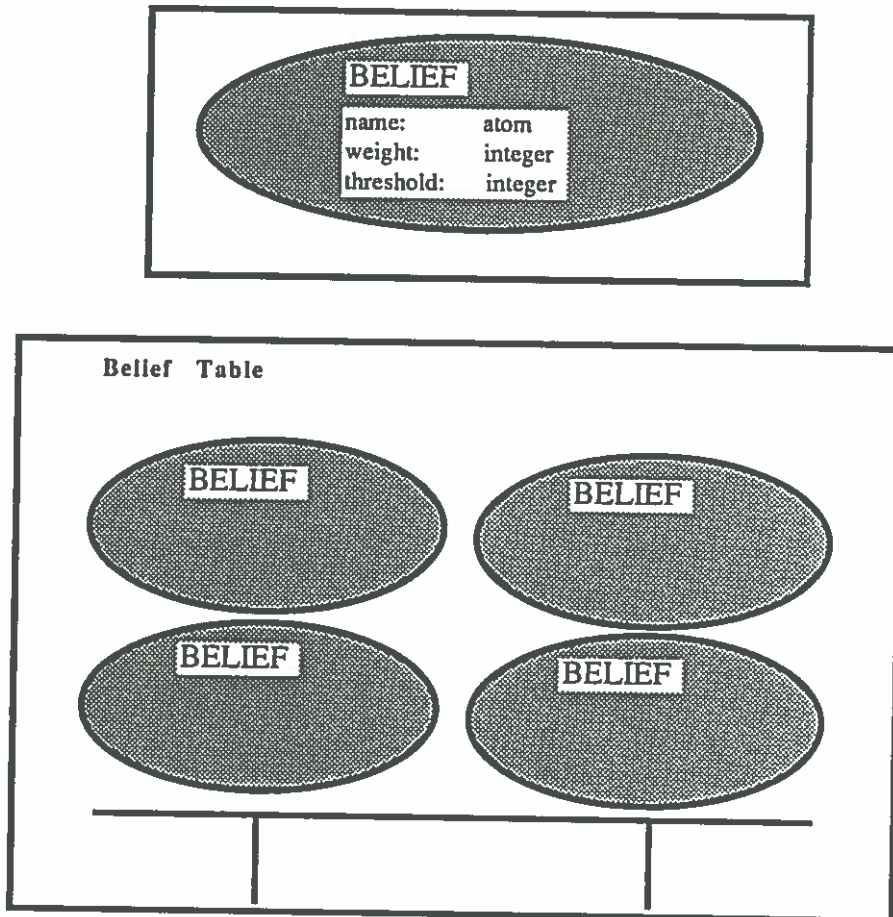


Figure 7: Beliefs

Internal State Variables

Internal state variables are attributes of the MA that describe what the MA “knows” at any given time. Figure 6 shows all of the current attributes of the MA. The limited ENSEMBLE model has internal state variables chosen to facilitate the three purposes of the MA described in the beginning of this section. Location, time, tempo, and meter instance variables represent what the MA believes is correct tempo information. Intermediate tempo state variables determine how the input performance varies from the MAs expectations at any time. Likewise, dynamics information represents what the MA believes is the dynamics level of performance at any time, and intermediate dynamics state variables determine how the performance varies from this. Figure 8 gives a high level view of how the intermediate state variables are affected by the actions of rule. The MAs third purpose of negotiating a lead has its own associated intermediate state variables in the form of modes. Internal state variables also describe MA character descriptors. Finally, sketches of pre-determined events are stored in internal state variables.

Location, Time, Tempo, and Meter Information

The most crucial part of MAs functioning is keeping track of where it is. The location, timing, tempo, and meter beliefs are crucial to establishing this function. Figure 9 displays some of the variables and values I discuss in this section.

Location of the MA in a score is determined by the internal state variable “location.” It contains an object of type “metrics” (see the section Input/Output Items below) that contains the state of what MA believes which bar, measure, beat, and tick it is currently in.

Time is just an absolute clock, keeping track of total system ticks. Metrics are adjusted according to differences in absolute time information.

Tempo is the measure of rate in the progression of a score. In terms of common

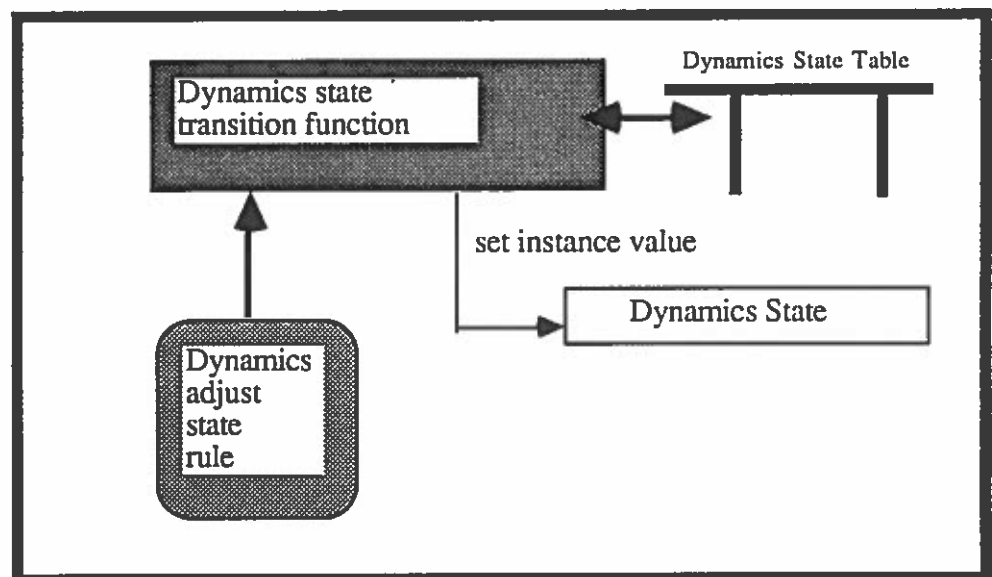
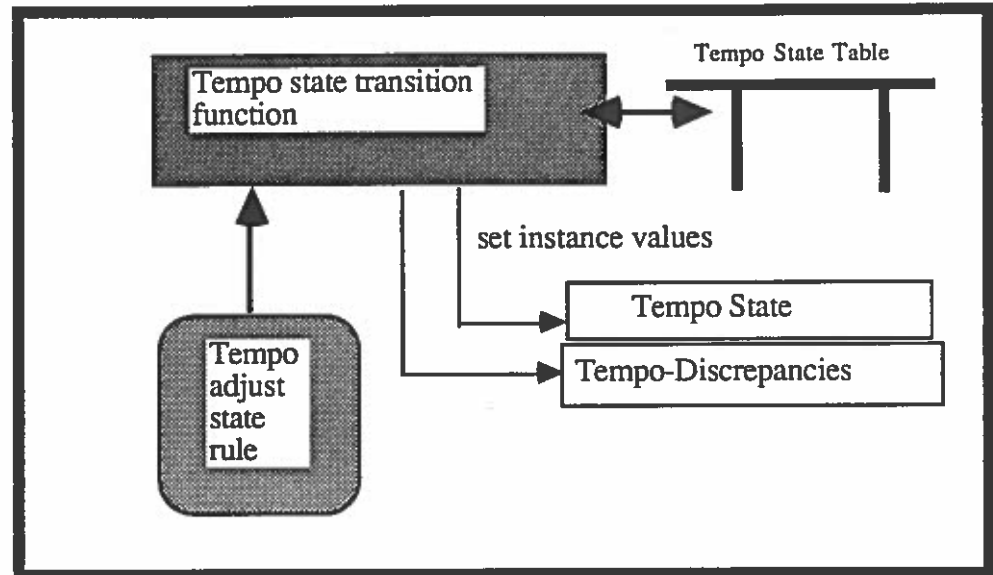


Figure 8: Intermediate State Variable Changes

practice notation, it is represented by a symbolic value, an integer value (beats per minute), or both. I have symbolically qualified Tempo to contain rates ranging from andante to prestissimo. These values act like enumerated types, and their associated rank is used to calculate a multiplier for determining durations according to the absolute time. The multiplier is used to allow the MA to have more than one value for a tempo symbol. In particular, I allow four values for each tempo distinction, so the multiplier is four times the rank of the tempo in the list of tempos.

Beats-per-measure and division-of-beat are instance variables representing time signature. The beats-per-measure variable is just an integer, and is what can be seen in common practice notation as the top number in a time signature. Division-of-beat represents what is seen in the bottom number. I chose an atomic representation of division-of-beat that contains duration-elem (see Musical Event section, Durations). These are values like HALF, QUARTER, or EIGHTH. Like Tempo, these values are also enumerated types. A function exists in the system that calculates the number of ticks in each symbolic duration name. These two pieces of information, along with the tempo are used to determine how long a beat is in terms of the absolute clock. As an example, assume that the division-of-beats is an EIGHTH, the tempo is PRESTISSIMO, and the beats-per-measure is four. PRESTISSIMO is the first tempo in the tempo list, so the multiplier is four (rank 1 times four). The next beat will occur after the duration of an eighth note has expired. To determine the next beat, we multiply the base number of ticks in an eighth note by the multiplier in order to determine the number of ticks for a beat. If the base number of ticks for an eighth note is 12, then the absolute duration of the beat would be 48 (12 times the multiplier 4).

The instance variable next-expected-beat is temporary storage that allows us to keep track of beat changes in terms of absolute time. In the example above, if the MA is currently on an even beat, and the absolute time is 96, then the next expected beat will be at 144 (just an addition of the absolute value of the division

of beat to the current clock).

Measures-per-bar is a significant concept in blues music. In blues, predictable patterns happen on every bar, which is defined by a consistent number of measures. In blues, this happens to be 12. Measures-per-bar is used to determine when a bar counter (inside of the location object) is to be incremented.

Beat change is a flag that is set every time a beat occurs. This is used to facilitate processing of tempo rules.

A MAs belief in tempo is absolute. It either believes it is on time, or it believes it is off. The complexity of the problem is that it takes time and various sequences of beats that are expected or unexpected to determine the state. This means that intermediary states for tempo are and discrepancies are needed. These states are represented by instance variables named tempo-adjust-state, and tempo-discrepancies.

A state transition function determines what the tempo-state value will be after the analyzer determines whether a beat has been determined to be on-time, early, or missing. This is the function that was described earlier in Figure 8. Figure 10 details the mechanics of this function. The discrepancies measured in absolute ticks for abnormal states are recorded in the list tempo-discrepancies. This information is subsequently used when the intentionality of “changing tempo and location” is determined by the planner. The discrepancies are the MAs record of what the new absolute values between beats are. This is used to compute new tempo symbols, and multipliers. If the sequence of beats is incomprehensible, the tempo transition function returns a tempo state which indicates that the MA should hold off on the production of events until it can notice the next downbeat. That will be the time when the next measure of events will start. All planned events between when the MA determined it was confused and the new measure start are purged.

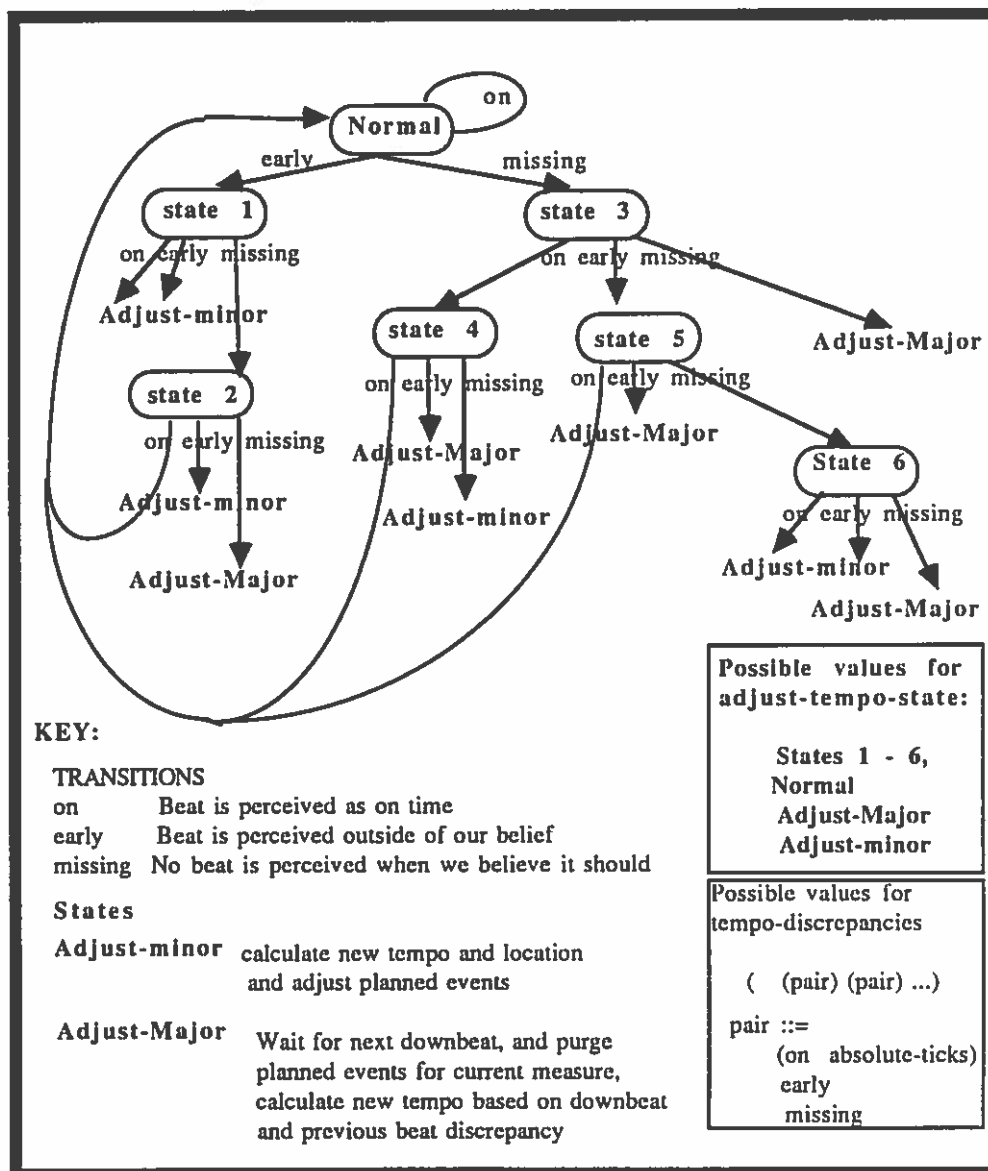


Figure 10: Tempo State Transition and Intermediate Tempo State

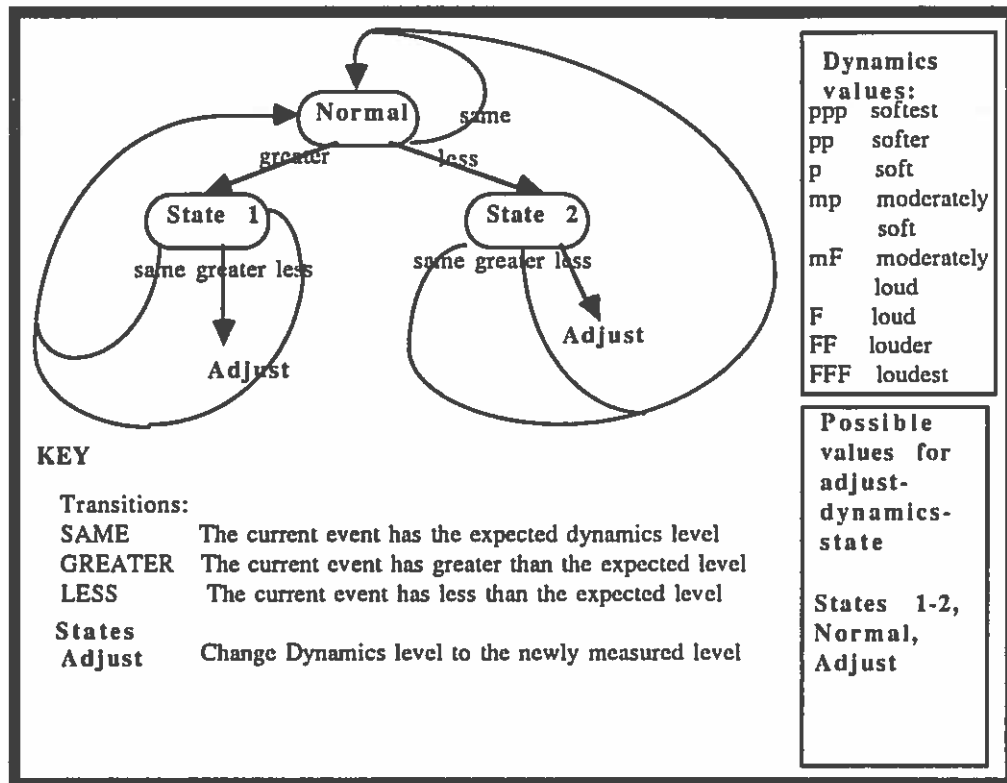


Figure 11: Dynamics State Transition and Intermediate Dynamics State

Dynamics, and Intermediate Dynamics State

Dynamics behavior in the MA is very similar to tempo, as can be seen in figure 8. The MA keeps track of a general dynamics level in the instance variable dynamics. The mechanics for dynamics transitions is described in figure 11. The dynamics values of musical events are directly determined by the value of the MAs dynamics instance variable. This is a symbolically quantified value, much like tempo. Dynamics are symbols being manipulated, and do not have a numerical basis like tempos. This means no discrepancies are stored and no multipliers are calculated. This was a design decision to limit the scope. It could have been handled in exactly the same symbol to number translation scheme that the tempo had. In fact, this must be done if the system will be driving actual sound devices. A state transition table is implemented, so the system still has a tolerance before reacting to dynamic changes. Dynamics changes are accomplished by a mechanism

of analyzer rules calling dynamics state transition functions on a dynamics state table to produce a change in the state variable dynamics-adjust-state.

Modes

Modes are the intermediary state mechanism for the MA regarding playing a preset melody or improvising a lead. As can be seen in figure 12, there are three major modes implemented in the MA. Play-mode is the crucial one for determining where the generated ideas of the performer will come from. Negotiation-mode is used for status of solo negotiations for the MA. React mode is used as a basis for determining whether the MA should modify its performance based on interpreted intentionality, or just go by its own beliefs regardless.

Character

These variables are my ad-hoc collection of internal state. These variables are logically described here for lack of a better grouping. The character variables contain the name of the MA, as well as random variables ambition and daring. The latter variables are used in the determination of self-initiated intentionality.

Sketches

Sketches is the logical grouping that contains "loose collections of ideas" that are used to create the events of the MAs performance. At some point in time before a bar change occurs, events are generated out of melodic ideas stored in either the melodic sketch list, or the improvised melody sketch. The harmonic sketch and rhythm sketches are reserved for the next development of ENSEMBLE which would explore counterpoint on these sketches to improvise a lead. Table 2 gives a description of what the sketch lists look like.

Modes

Play-Mode

Pending-NORMAL

Mode in which MA gets its performance information from melody sketches MA takes these uses the sketch information to generate events, which are placed in the Performer Fact-Base. MA either returns to NORMAL if it previously was in NORMAL, or Stays in Pending-normal until the end of the bar when it returns to NORMAL.

Pending-LEAD

Mode in which MA creates its own performance information in the improvised melody sketch. This information is used to generate events, which are placed in the Performer Fact-Base. MA either returns to LEAD if it previously was in LEAD, or remains in Pending-lead until the end of the bar when it goes into LEAD.

NORMAL

Signifies that MA is just playing normal melody, and intends to keep doing that.

LEAD

Signifies that MA is just playing a lead, and intends to keep doing that.

Negotiation Mode

Normal

MA is not waiting for an answer to any negotiation

Pending-Acknowledge

MA is waiting for an ACK or NACK to a request

React Mode

Internal

Ignore external requests (egotistical MA)

External

React to all external requests (docile MA)

Figure 12: Modes of the MA

<sketch>	::= <melodic sketch> <harmonic sketch> <rhythmic sketch>
<melodic sketch>	::= (<melodic sketch element> +)
<melodic sketch element>	::= <melody element> <duration>
<melody element>	::= <pitch-element> "REST"
<harmonic sketch>	::= (<harmonic sketch element> +)
<harmonic sketch element>	::= <harmony element> <duration>
<harmony element>	::= <harmonic-mode> "REST"
<rhythmic sketch>	::= (<rhythmic sketch element> +)
<rhythmic sketch element>	::= <rhythm element> <duration>
<rhythm element>	::= "EVENT" "REST"

Table 2: Sketches and Their Values

Heuristics

A set of the currently implemented heuristics is described in Appendix B. These rules are all from CMK (no specific musical knowledge significant to the character of the MA is implemented in the limited model). The heuristics are categorized by the processor (analyzer, planner, or performer) to which they belong. Table 3 shows us an example of a rule from the analyzer rule base in pidgin form (taken from Appendix B). This rule matches a gesture that resides in the analyzer. The gesture has the qualified values "Head" for the slot implementor, "Nod" for the slot motion, and "Intent" for the slot emphasis. The MA's internal state variable negotiate-mode must be set to "NORMAL" as the last condition that must be satisfied for the rule to fire. The action specified is to get the belief "receive-my-lead-request" in the belief table, and add one to it. So in English, the rule "interpret-my-lead-request" says if the MA currently has a negotiate-mode of normal play, and has received a head nod gesture with intent emphasis, increment the belief in request for the MA to take a lead.

```

RULE interpret-my-lead-request
  IF :   GESTURE_Implementor = "HEAD"
         and
         GESTURE_Motion = "NOD"
         and
         GESTURE_emphasis = "INTENT"
         and
         Negotiate-Mode = "NORMAL"
  THEN: <add-belief>
        /get (BELIEF-TABLE) receive-my-lead-request/ 1

```

Table 3: A Sample Musical Agent Heuristic

Configuration

At system startup time, the conductor is configured, and given a set of user preferences for each MA to configure itself. The conductor is also configured with the names of each MA instance created (in this model, the name of the MA and the name of the PA). The conductor sends messages to each MA instance to configure. The configuration method of the MA receives control, at which time it initializes all processors and structures according to a base set of initializations described in the function. It then looks for user preference configurations from the conductor to further distinguish its character.

This is the way that future ENSEMBLE models will be able to give each agent a different character. The user preferences include information such as the base musical sketches of information to be performed (see the section Input/Output Information), and timing information. Theoretically, we could substitute sketches and improvisation rules for each particular musical motif to get a different musical style performance (i.e. blues or jazz).

<metrics>	::= <bar> <measure> <beat> <tick>
<bar>	::= integer
<measure>	::= integer
<beat>	::= integer
<tick>	::= integer
<timestamp>	::= integer

Table 4: Description of Metrics Values

Input/Output Information

The clues which the agents in an ensemble interpret and produce are divided into two categories: gestures and musical events. The complete descriptions of input/output information is compiled in Appendix A. Figure 13 and Figure 14 show the hierarchy of these clues.

All gestures and musical events are tagged with timing and location information. Locations are specified by an object called “metrics”, which contains location information relative to a score (bar, measure, beat, and ticks (beat subdivisions)). Figure 13 shows the hierarchy of metrics as a single hierarchical object. Table 4 describes the slot values of metrics. The values are integer, but their ranges are specified by the values of the internal state variables beats-per-measure, division-of-beat, measures-per-bar, next-expected-beat, and tempo. The timestamp is an absolute measure, while metrics is a relative measure of location in a score.

Gestures

Gestures are implemented as ORBS objects that contain an implementor (the thing that makes the gesture), a motion (the type of gesture), an emphasis, and location information. Figure 13 shows the hierarchy of gestures, and table 5 describes the slot values. Gestures are currently the most utilized clues of the limited ENSEMBLE model. Figure 15 gives an example of a gesture. In English, this gesture

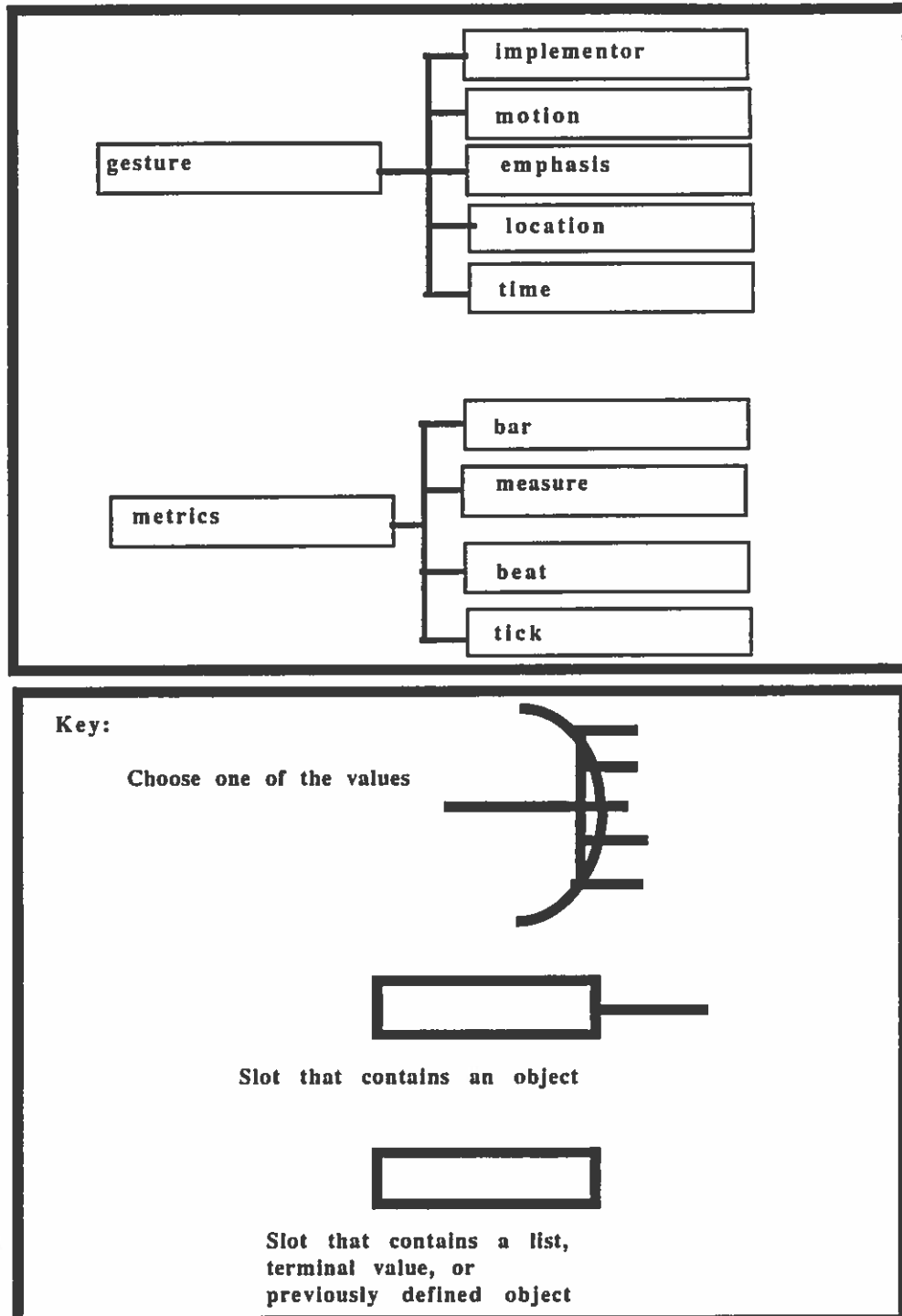


Figure 13: Input/Output Information (Gestures and Metrics)

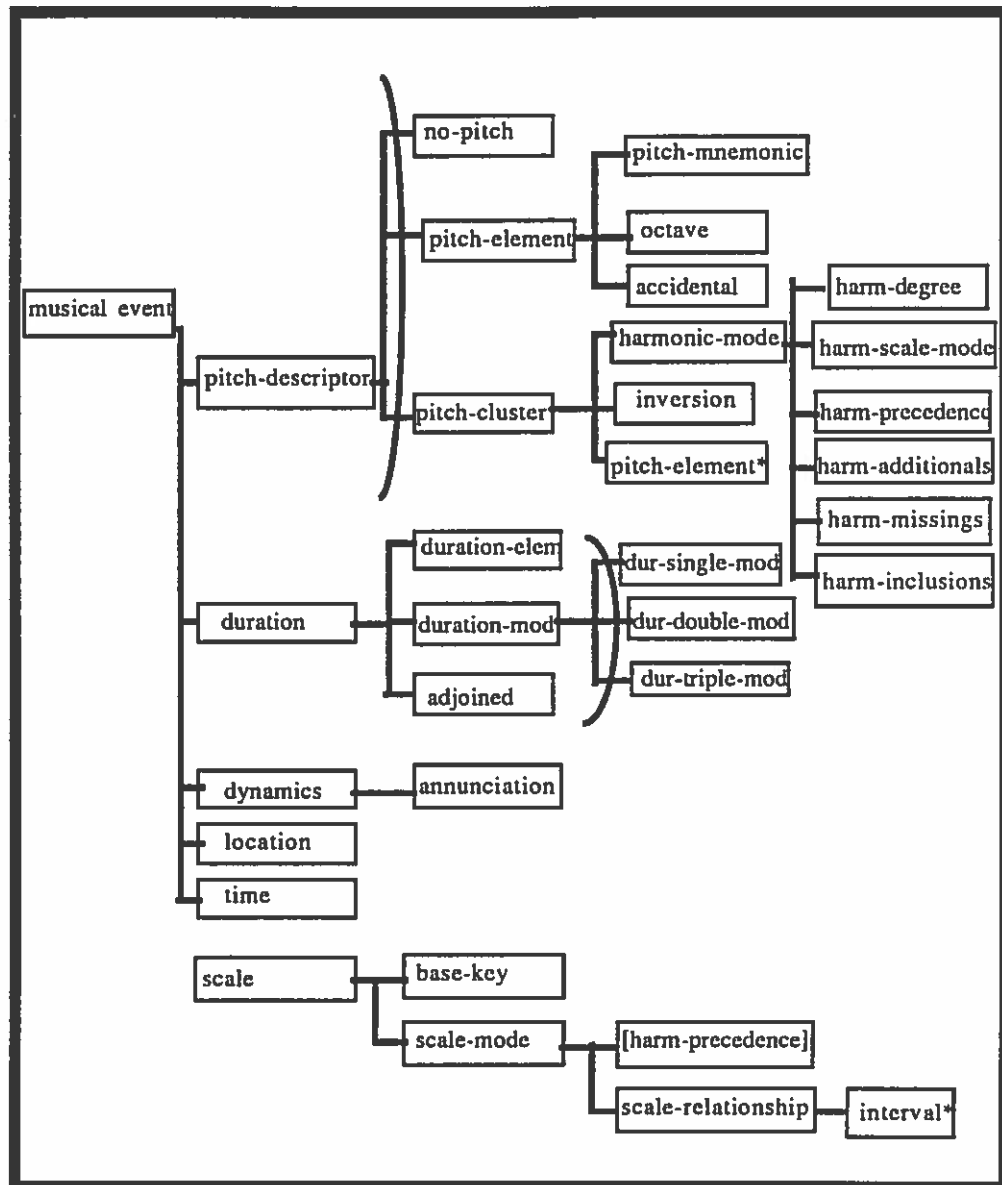


Figure 14: Input/Output Information (Musical Events)

<gesture>	::= <implementor> <motion> <emphasis> <metrics> <timestamp>
<implementor>	::= FACE HEAD HAND FOOT
<motion>	::= <face-motion> <head-motion> <hand-motion> <foot-motion>
<face-motion>	::= SMILE GRIMACE STARE
<head-motion>	::= NOD SHAKE
<hand-motion>	::= POINT
<foot-motion>	::= TAP
<emphasis>	::= <face-emphasis> <head-emphasis> <hand-emphasis> <foot-emphasis>
<face-emphasis>	::= <smile-emph> <grimace-emph> <stare-emph>
<smile-emph>	::= SMIRK CLOSED-SMILE OPEN-SMILE
<two-level-emph>	::= CASUAL INTENT
<grimace-emph>	::= <two-level-emph>
<stare-emph>	::= <two-level-emph>
<head-emphasis>	::= <two-level-emph>
<hand-emphasis>	::= <two-level-emph>
<foot-emphasis>	::= <two-level-emph>

Table 5: A Description of Gesture Values

Gesture GE-1	
Implementor	FOOT
Motion	TAP
Emphasis	CASUAL
Location	LC-1
Timestamp	0000000012
METRICS	LC-1
Bar	1
Measure	1
Beat	1
Ticks	1

Figure 15: An Example Gesture

translates to a casual foot tap that happens on the first beat of a song.

Musical Events

Musical events are ORBS objects with more of a hierarchy to adhere to than gestures. A musical event contains pitch information, duration information, dynamics information, and timing and location information. Excepting the timing information, all the above descriptions are ORBS objects filling instance variables of the event. The hierarchy follows the description of musical events in figure 14, and the values of the terminals in slots in table 6 and table 7.

Figure 16 and figure 17 give examples of musical events.

The note of Figure 16 is a dotted quarter note, being the C sharp just below middle C, played with a mezzo-forte dynamics, and played on the first beat of a song.

<event>	::= <pitch descriptor> <duration> <dynamics> <location> <time>
<location>	::= <metrics>
<pitch descriptor>	::= <pitch element> <pitch cluster> <no pitch>
<pitch element>	::= <pitch mnemonic> [<octave>] <accidental>
<pitch mnemonic>	::= C D E F G A B
<accidental>	::= SHARP FLAT NATURAL
<pitch cluster>	::= <harmonic mode> <inversion> (<pitch element>*)
<inversion>	::= integer (0 .. (1- number of pitch-elements))
<harmonic mode>	::= <harm degree> <harm scale mode> <harm precedence> <harm-additionals> <harm-missings> <harm-inclusions>
<harm degree>	::= I II III IV V VI VII
<harm scale>	::= <scale mode>
<harm precedent>	::= MAJOR MINOR
<harm position>	::= <harm pos> <harm alterations>
<harm pos>	::= 1 2 3 4 5 6 7 8 9 11 13
<harm alterations>	::= AUGMENTED MAJOR MINOR DIMINISHED
<harm additionals>	::= (<harm position> +)
<harm missings>	::= (<harm position> +)
<harm inclusions>	::= (<harm position> +)

Table 6: A Description of Values of Musical Events

<duration>	::= <duration elem><duration mod><adjoined>
<duration elem>	::= WHOLE HALF QUARTER EIGHTH SIXTEENTH
<duration mod>	::= <dur-single-mod> <dur-double-mod> <dur-triple-mod>
<dur-single-mod>	::= DOTTED STACCATO NONE
<dur-double-mod>	::= TIED NONE
<dur-triple-mod>	::= TRIPLET NONE
<adjoined>	::= (<duration> *)
<dynamics>	::= <annunciation>
<annunciation>	::= PPP PP P MP MF F FF FFF
<scale>	::= <base key> <scale mode>
<base key>	::= <pitch element>
<scale mode>	::= [<harm precedence>] <scale relationships>
<scale relationships>	::= (<interval> ⁺)
<interval>	::= HALF WHOLE WHOLE-HALF

Table 7: A Description of Values of Musical Events

MUSICAL EVENT ME-1			
Pitch-Descriptor	PD-1		
Duration	DR-1		
Dynamics	DY-1		
Location	LC-1		
Time	00000000012		
PITCH-ELEMENT	PD-1	DYNAMICS	DY-1
Pitch-Mnemonic	C	Annunciation	mF
Octave	3		
Accidental	SHARP		
DURATION	DR-1	METRICS	LC-1
Duration-Elem	QUARTER	Bar	1
Duration-mod	Dotted	Measure	1
Adjoined	()	Beat	1
		Ticks	1

Figure 16: An Example Musical Event (A Note)

MUSICAL EVENT ME-2			
Pitch-Descriptor PD-2			
Duration	DR-2		
Dynamics	DY-2		
Location	LC-2		
Time	00001328488		
PITCH-CLUSTER	PD-2	DYNAMICS	DY-2
Harmonic-Mode	HM-2	Annunciation	FF
Inversion	1		
Pitches	(PE-1 PE-2 PE-3)		
DURATION	DR-2	METRICS	LC-2
Duration-Elem	HALF	Bar	2
Duration-mod	NONE	Measure	3
Adjoined	()	Beat	2
		Ticks	1
HARMONIC-MODE	HM-2		
Harm-Degree	V		
Harm-Scale-Mode	SM-1		
Harm-Precedence	MAJOR		
Harm-Additional	()		
Harm-Missings	()		
Harm-Inclusions	((1 MAJOR) (3 MAJOR) (5 AUGMENTED))		
SCALE-MODE	SM-1		
Precedence	MAJOR		
Relationships	(whole whole half whole whole whole half)		

Figure 17: Another Example Musical Event (A Chord)

The chord of Figure 17, assuming in the base-key of C, is an augmented major fifth, played for a half note duration, played with very loud dynamics (*forte forte*), and located at bar two, measure three, beat two. The chord is played in the first inversion, so the pitches played would be B, D sharp, G. The pithes slot of the pitch cluster can be filled at time of creation of the chord, as all the necessary information is there.

This representation is not an inheritance type of class system. Inheritance was unavailable in ORBS at the time of specification. The domain should not be too difficult to alter for representations utilizing a true class system.

In the implementation of the limited ENSEMBLE, the dynamics of the musical events are the only salient feature that the rules presently take advantage of. Those values are used in determining change in dynamic levels. The framework should be well set by the ENSEMBLE model to increase the level of intentionality derived from the music.

Tying It All Together

This section ties all of the previous sections together with an example of an execution cycle from the perspective of the discussed implementation.

Refer back to Figure 6, and assume the following internal state variables have been assigned:

1. Tempo is PRETISSIMO
2. Tempo-adjust-state is STATE-2
3. Tempo-discrepancies are ((early 0033) (missing 0036))
4. Metrics are (a) bar 1, (b) measure 1, (c) beat 3, and (d) tick 5
5. Time is set to 0041
6. Beats-per-measure is 4
7. Measures-per-bar is 12

8. Next-expected-beat is 0048
9. Beat-change is FALSE
10. Dynamics and Dynamics-adjust-state are incidental
11. Play-mode is NORMAL
12. React-mode is EXTERNAL
13. Negotiate-mode is NORMAL
14. Character variables are incidental
15. Sketch variables are incidental
16. Input and Output Queues are initially empty
17. No beliefs in the Belief table have been set

A foot-tap gesture (as in figure 15) is placed in the MAs input queue containing all the information with the exceptions that the location is the same as the metrics above, and the time of the gesture is 0042. The MA has received a "tick" from the conductor. The control function takes over, and increments the time and metrics. The control function subsequently imports all clues from the input queue into the analyzer factbase. The control function gives a "start" message to the analyzer.

The analyzer puts the rule `adjust-internal-state-for-beat-tempo` into the conflict set as it is matched with the gesture. No other rules in the analyzer rule base can unify at this time, so the rule fires. The action is a call to the tempo transition function with the state `BEAT-EARLY`. As figure 10 shows, when the MA is in `STATE 2` with a `BEAT-EARLY`, the state that is placed in the tempo-adjust-state variable is `ADJUST-MINOR`. The analyzer can now unify the tempo-adjust-state with the rule `interpret-tempo-and-location-change`. This rule fires, producing an action of setting the belief `adjust-tempo-and-location` past its threshold. The analyzer can no longer unify any rules, and it relinquishes control to the MA control function.

The MA control function sends a start message to the planner. The planner unifies the rule `adjust-my-tempo-and-location`, because the belief in the intentionality of adjusting tempo and location has surpassed its threshold and become active. The rule has also unified the `tempo-adjust-state` and the `react-mode`. The planner fires the rule. The first functional action is the calculation of new metrics, and a new tempo multiplier. The difference of minus three absolute ticks per beat translates to a tempo multiplier of one less than the four ticks of the original multiplier. The next-expected beat is calculated with the new multiplier, and the metrics is set to indicate synchronization of the new beat. The next function of the rule action is invoking the performer.

The performer has all rules turned off with the exception of `adjust-timing-of-events`. This rule unifies all planned events in the factbase of the performer, and fires the rule whose action modifies the time of the unified event for the newly calculated tempo and location. When all events have been modified, the control is returned to the planner.

The planner continues its action of the current rule by resetting the `tempo-adjust-state` to normal. At this point, no more rules may unify, so the planner returns control to the MA control function.

The control function starts the performer. The performer will unify any events that have an equivalent time to the MAs, or any gestures that have an equivalent metrics to the MAs. The performer deletes these from its own factbase, and places them in the output queue. Any other clues whose time has been bypassed by the tempo change are removed from the factbase according to the rule `remove-surpassed-events`.

The control function then resumes control by emptying all clues in the output queue into the performance list. The MA has completed its execution cycle.

Summary

The MA is implemented in ORBS II based on its provision of multiple instantiations of rule-based interpreters, and object-oriented LISP-like style.

The MA is implemented as a group of three rule based systems, one for each processor. Each rule-based system executes synchronously in order to complete an execution cycle of the MA. The MA is an ORBS object containing instance variables that store all processors, intermediary data structures, and internal state variables. Each processor is implemented as an ORBS rule-based interpreter, containing its own rules in a rule base, and references to internal state, intermediate structures, and clues in its fact base. The rules are divided into CMK and SMK rules, which are further divided into a group specifying which processor they belong to. Domain items consist of gestures, and musical events, and are represented as ORBS objects.

This completes the description of the limited ENSEMBLE system. In the next chapter, I discuss the results of this research, where the implementation of the limited ENSEMBLE resides in the area of computer music, limitations of the ENSEMBLE model, and related research.

CHAPTER 4

CONCLUSIONS

This thesis describes my efforts in understanding current problems in computer music. I isolated the topic of intentionality in performance after drawing the connection of work in the area of intentionality in English phrases (Appelt, 1985). The purpose then became to find a way to exploit the interpretation and production of intentionality in a musical context, namely performance. One implication of this was to develop a language that would serve as a way to infer intention from visual and musical clues. The indulgence into musical intentionality lead to the specification of a large scale system, ENSEMBLE, that would exhibit cooperative agent behavior in the production of a performance through the interpretation of intentionality. This model was too large for the scope of this masters thesis, and many stages of refinements had to be made. One major refinement made was the specification and implementation of a single agent, driven by a partially functional agent. This limited the amount of synchronization and multi-agent negotiations, but still provided much work in establishing the basic model of a musical agent. This musical agent is a paradigm for future work in multiple cooperating musical agent systems, and exhibits the intentional basis for determining the content of a performance.

Results

My thesis has presented an initial formalization of a multi-agent performance system in music. This approach seems to be useful, but because of the work necessary to build a system that captures the complete functionality of a cooperative multi-agent ensemble, it remains to be seen how accurately this methodology is in

comparison to the real-world equivalent.

At the time of this writing, the limited model was not complete. All major structures were defined, all domain items were defined, and tested with their associated operators. A basic set of CMK rules was developed. System integration and general debugging was left at the time development was suspended for writing this document.

Related Systems

In the field of computer music, two areas of research have inspired the research direction of this thesis. These are automated composition, and performance systems. Roads (1985) surveys the field, concentrating on artificial intelligence applications.

Compositional Aids

The majority of work in the field addresses compositional aids. These systems specified various levels of music representation, but left the task of composition either for random note-generating functions, or user ordering of rules and functions provided by the systems. In essence these systems were languages for music specification. The work of (Hiller, 1969, Hiller, 1981) formalized a system of generation and analysis functions that a user could choose and order on a set of note generating functions based on random and probabilistic functions. Similar to this was the POD system of (Truax, 1976) which used poisson distributions as a basis for note generation.

Systems that followed these earlier attempts emphasized the hierarchical nature of music representation. The PLA system (Schottstaedt, 1983) was organized as a formal ALGOL-like programming language for music, providing a form of parallel processing with a message passing system for task communication between processes. A user could specify a number of musical tasks that could pass information

to each other (i.e. a generator task could provide notes to a trill task that would turn each note into a trill). The FORMES system (Rodet & Cointe, 1984) provided many of these features, but based its representations on an object-oriented model of the domain. Flavors Band (Fry, 1980, Fry, 1984) provided a higher level of language functions than the other systems, and like FORMES have its representations formed on an object-oriented model.

Compositional systems were intended to be compositional aids. These assistants provided tools and language support for the composer to work with in formalizing musical knowledge to produce a composition. Automated composition systems only provided random ideas from which a composer could choose. Compositional systems never address the entire creation of the music. They also never address a sound cognitive basis in their representations. The things that are addressed are musical syntax and some high level rules of composition.

Performance Systems

Performance systems address a distinguished set of computer music research. These systems deal with real-time issues, and in general aiding or complementing a performance. They can vary in complexity from off the shelf commercially available sequencers and compositional aid systems, to academic research oriented systems such as the automated accompanist systems of Vercoe and Dannenberg, described in (Roads, 1985). The Dannenberg system tracked a lead instrument in order to produce an accompaniment of a stored score while adjusting for performance clues such as tempo and dynamics changes (this was one of the major inspirations of the ENSEMBLE system). Performance systems that address automated improvisation try to do in realtime and with external influence what automated composition systems do. Automated compositional systems are not forced into a multi-agent approach, and often do not make attempts at allowing human intervention. This limits intentionality in composition to be based on what the composer chooses it to

be.

The conceptual and cognizant processes of composer and listener alike can not be complete until emotional behavior is considered. Intelligent systems of the future will provide some type of planning based on this information. As described by Otto Laske (Laske, 1983), "these music planning systems can be 'opportunistic'," (an event driven control structure made up of clusters of planning actions), "or 'non-opportunistic'" (script-based, hierarchical, and linear approaches). ENSEMBLE has taken a step in both directions.

Further Work

This section analyzes the ENSEMBLE project critically from a conceptual view. There were many areas that were finessed, and some where an answer was not verifiable, such that a judgement call had to be made. ENSEMBLE is a first attempt at many different issues in computer music systems, and should provide a basis for other computer music problems in performance systems.

Intentionality in Note Selection

The basis of this thesis was the exploitation of intentionality in music. I have researched intentionality in performance interpretation (progressive intention), but have finessed issues of compositional intentionality ("what is the composer/improviser trying to thematically say through this selection of notes"). Thematic intention is a research area wide open, and must be realized for any future accomplishments in performance and compositional systems.

Issues in Synchronization

In my specification, two large assumptions are made dealing with synchronization. The first is the synchronous agent problem (ordering which performer goes when, as opposed to asynchronous agents adding to a performance). The second

is the internal synchronization problem (having analysis, planning, and execution ordered and synchronous).

The Synchronous Agent Problem

It is assumed that an agent believes that a performance is complete with the exception of the information it will add, and thus completes the performance with its own addition. A multi-agent system must truly be multi-tasking such that we can only look back at previous time slices, and not at the current time slice (we can only add to that). The MA in its current form assumes events in the performance have all been added except for its own, and all the information it will need will be in the performance for the next time it is active.

Internal Synchronization Problem

The second major assumption is that the processes within each agent itself are synchronous. Evidence from psychology suggests that humans focus attention on one task at a time, work on these tasks in small amounts, and in order of opportunity to accomplish relatively simultaneous events. A blackboard approach to modeling an agent with cooperating processes in analysis, planning, and performing would be better than the synchronous control structure presently in the MA.

Event Synchronization

In the limited ENSEMBLE, I have made some intentional inferences for the adjustment of timing. This is handling the smaller, and most frequent problem of tempo drift, where timing references variate by small incremental amounts. The problem of finding location in a score when confused is more of an AI pattern search problem, and was one of the main features the Dannenberg (Roads, 1985) automated accompanist addressed. This was the problem of noticing when the

expected performance was not the same as the performance encountered by the agent that is monitored. The accompanist would search through a score until it found a suitable candidate position for where the lead currently was, and move its performance to that spot. My ENSEMBLE assumes that no agent will stray by major portions.

Conflict in Interpretation of Intentionality

Conflict in the interpretation of intentionality is when two clues can possibly promote different beliefs. My system has managed to alleviate this problem partially by drawing upon contextual information within the internal state variables of the MA. An example where this happens is when a gesture that is a head shake could either be a "receive my lead acknowledge", or a "receive my-lead request." When the system becomes more complex in terms of multiple interpretations (i.e. determining beat from musical and gestural clues, etc.), this will unfortunately be reflected in the rules and the state structure of the MA.

Listening to the Future

This thesis also serves as an inspiration to apply theories of intentionality to other forms of communication besides language. Music is an area of communication that is not researched to the same extent as language, and may not ever be due to the strong connections it has to emotion. Emotion crucially needs to be formalized before intentionality can be verifiably tied to the inspiration of music. For now, we can view intentionality functionally in terms of modifying a performance.

This project is just the tip of a very large iceberg. My personal views are that AI programming techniques are quite valid in music applications, in particular performance and compositional systems. My contribution is only a small glimmer of hope that many tough problems in computed music may be accomplished, and yet it has also identified many other problems along the way. I hope someday to

continue this work, and in general hope that my pursuits are followed by others. There are many parallels between music and speech, and I feel that anyone who follows my path of research will find it beneficial to review work in natural language processing.

Some previous researchers have taken the path of describing music in terms of descriptive languages. Some researchers have provided systems that choose notes with very little cognitive basis, and random behavior. Others have provided systems that interact with their users to produce notes, but have kept the human still as the primary decision element in the composition process. The view of choosing notes based on intentionality seems to have a sound cognitive basis, and challenges researchers to formalize a connection between emotion and intention. The use of intentionality as a method to modify performance also has a sound basis. Intentionality can bring us one step closer to the creative process, and one step closer to truly inspired music.

APPENDIX A

SPECIFICATION OF INPUT/OUTPUT ITEMS

This appendix describes the input/output items of the MA. The language I use here looks like BNF descriptions of language grammars. It is only used here to qualify values in the record like fields (slots) of my orbs implementation of the input/output items. These descriptions do not imply order.

The descriptions I give contain a field-value description of the items, followed by a table describing the major relations (implemented as functions or methods) of the items.

KEY:

<label>	Decomposable Item
UPPERCASE ITEM	Atomic Symbol (qualified value)
lowercase item	Specified Type
::=	Is translated to mean
	OR
<element>*	0 or more of the element
<element>+	1 or more of the element
(<element>)	list of an element
[<element>]	optional element (one element in a list of elements)
{ <element> }	mandatory element (one element in a list of elements)

Temporal Specifiers

The ranges for bar, measure, beat, and ticks are quantified according to user preferences.

<metrics>	::= <bar> <measure> <beat> <tick>
<bar>	::= integer
<measure>	::= integer
<beat>	::= integer

<tick> ::= integer
 <timestamp> ::= integer

functions

METRICS
same-metrics
greater-metrics
lesser-metrics
same-bar (strong)
same-bar (weak)
greater-bar
lesser-bar
same-measure (strong)
same-measure (weak)
greater-measure
lesser-measure
same-beat (strong)
same-beat (weak)
greater-beat
lesser-beat
same-ticks (strong)
same-ticks (weak)
greater-ticks
lesser-ticks

Musical Events

<event> ::= <pitch descriptor> <duration>
 <dynamics> <location> <time>

<location>	::= <metrics>
<pitch descriptor>	::= <pitch element> <pitch cluster> <no pitch>
<pitch element>	::= <pitch mnemonic> [<octave>] <accidental>
<pitch mnemonic>	::= C D E F G A B
<accidental>	::= SHARP FLAT NATURAL
<pitch cluster>	::= <harmonic mode> <inversion> (<pitch element>*)
<inversion>	::= integer (0 .. (1- number of pitch-elements))
<harmonic mode>	::= <harm degree> <harm scale mode> <harm precedence> <harm additional> <harm missings> <harm inclusions>
<harm degree>	::= I II III IV V VI VII
<harm scale>	::= <scale mode>
<harm precedent>	::= MAJOR MINOR
<harm position>	::= <harm pos> <harm alterations>
<harm pos>	::= 1 2 3 4 5 6 7 8 9 11 13
<harm alterations>	::= AUGMENTED MAJOR MINOR DIMINISHED
<harm additional>	::= (<harm position> +)
<harm missings>	::= (<harm position> +)
<harm inclusions>	::= (<harm position> +)
<duration>	::= <duration elem><duration mod><adjoined>
<duration elem>	::= WHOLE HALF QUARTER EIGHTH SIXTEENTH
<duration mod>	::= <dur-single-mod> <dur-double-mod> <dur-triple-mod>
<dur-single-mod>	::= DOTTED STACCATO NONE
<dur-double-mod>	::= TIED NONE
<dur-triple-mod>	::= TRIPLET NONE
<adjoined>	::= (<duration> *)

<dynamics>	::= <annunciation>
<annunciation>	::= PPP PP P MP MF F FF FFF
<scale>	::= <base key> <scale mode>
<base key>	::= <pitch element>
<scale mode>	::= [<harm precedence>] <scale relationships>
<scale relationships>	::= (<interval>+)
<interval>	::= HALF WHOLE WHOLE-HALF
<sketch>	::= <melodic sketch> <harmonic sketch> <rhythmic sketch>
<melodic sketch>	::= (<melodic sketch element> +)
<melodic sketch element>	::= <melody element> <duration>
<melody element>	::= <pitch-element> "REST"
<harmonic sketch>	::= (<harmonic sketch element> +)
<harmonic sketch element>	::= <harmony element> <duration>
<harmony element>	::= <harmonic-mode> "REST"
<rhythmic sketch>	::= (<rhythmic sketch element> +)
<rhythmic sketch element>	::= <rhythm element> <duration>
<rhythm element>	::= "EVENT" "REST"

functions

DURATION	DYNAMICS	PITCH	PITCH-CLUSTER	SCALE
same -duration (strong)	same -dynamics (strong)	same -pitch (strong)	same-chord (strong)	same-scale
same -duration (weak)	same -dynamics (weak)	same -pitch (weak)	same-chord (weak)	scale -member
greater- duration	greater- dynamics	greater- pitch	greater -harm-rank	scale -union
lesser- duration	lesser- dynamics	lesser-pitch	lesser -harm-rank	scale -intersection
		find-relative -pitch		scale-difference
		find-pitch- distance		make-mode- from-scale-list

Gestures

<gesture>	::= <implementor> <motion> <emphasis> <location> <time>
<location>	::= <metrics>
<implementor>	::= FACE HEAD HAND FOOT
<motion>	::= <face-motion> <head-motion> <hand-motion> <foot-motion>
<face-motion>	::= SMILE GRIMACE STARE
<head-motion>	::= NOD SHAKE
<hand-motion>	::= POINT
<foot-motion>	::= TAP
<emphasis>	::= <face-emphasis> <head-emphasis>

```

<face-emphasis>          <hand-emphasis> | <foot-emphasis>
                        ::= <smile-emph> | <grimace-emph> |
                           <stare-emph>
<smile-emph>             ::= SMIRK | CLOSED-SMILE | OPEN-SMILE
<two-level-emph>        ::= CASUAL | INTENT
<grimace-emph>          ::= <two-level-emph>
<stare-emph>            ::= <two-level-emph>
<head-emphasis>         ::= <two-level-emph>
<hand-emphasis>         ::= <two-level-emph>
<foot-emphasis>         ::= <two-level-emph>

```

In this implementation, the relations are implemented as functions, since they all require operations on external objects (some possibly of different types). Internal operations are implemented by methods. Other than creators, methods were only useful in this implementation for “sets” and “gets” of object attributes on the domain objects (provided by ORBS).

APPENDIX B

RULES

The following appendix describes the heuristics of the system. They are divided into three groupings; analyzer rules, planner rules, and performer rules. All of these rules are considered to have come from CMK, as they are common knowledge, not specific to any individual character (i.e. drummer, bassist, etc.). The rules are described in a pidgin form, adhering to the following rules:

KEY:

OBJECT: Refers to a specific type of object denoted by uppercase word.

Attribute: Refers to an instance variable of an object, denoted by capitalized word. When not preceded by an **OBJECT**, it refers specifically to an instance variable of the MA.

OBJECT_Attribute: An instance variable in an object is denoted by an underscore separating the slot name and the object name.

productionssystem: An ORBS production system residing in an instance variable of the MA with the name in lowercase.

productionssystem*Attribute: An ORBS production system with a given feature (such as rulebase, factbase, etc) specified as an attribute.

(STRUCTURE): An intermediary structure within the MA.

RULE rulename: A pidgin description of a heuristic implemented in ORBS with a name of rulename, and with an IF-THEN condition-action specifier.

"VALUE": a literal value.

All other symbols are syntactic sugar for describing actions

remove	<function name>	param1
add	>	if .. then .. else .. fi
get	<	send
set	=	<=
+	and	>=
-	or	not
/ perform surrounded by slashes /		

Analyzer Rules

The analyzer rules are the interpretation rules that allow the MA to interpret belief. These heuristics promote two types of intentionality additions, absolute belief and incremental belief.

RULE adjust-internal-state-for-dynamics

IF : MUSICAL-EVENT

THEN: <find-dynamic-state> MUSICAL-EVENT

RULE interpret-dynamics-change

IF : Dynamics-Adjust-State = "ADJUST"

THEN: <set-past-threshold>

/get (BELIEF-TABLE) adjust-dynamics/

RULE adjust-internal-state-for-db-tempo

IF : GESTURE_Implementor = "HEAD"

and

GESTURE_Motion = "NOD"

and

GESTURE_emphasis = "CASUAL"

THEN: if Beat-Change then

<find-tempo-state> "BEAT-ON-TIME"

else

<find-tempo-state> "BEAT-EARLY"

else

fi

RULE adjust-internal-state-for-beat-tempo

IF : GESTURE_Implementor = "FOOT"

and

GESTURE_Motion = "TAP"

```

                                and
    GESTURE_emphasis = "CASUAL"
  THEN: if Beat-Change then
    <find-tempo-state> "BEAT-ON-TIME"
    else
    <find-tempo-state> "BEAT-EARLY"
    else
    fi

```

RULE interpret-tempo-and-location-change

```

  IF :   Tempo-Adjust-State = "MINOR-ADJUST"
                                or
        Tempo-Adjust-State = "MAJOR-ADJUST"
  THEN: <set-past-threshold>
        /get (BELIEF-TABLE) adjust-tempo-and-location/

```

RULE interpret-my-lead-request

```

  IF :   GESTURE_Implementor = "HEAD"
                                and
        GESTURE_Motion = "NOD"
                                and
        GESTURE_emphasis = "INTENT"
                                and
        Negotiation-Mode = "NORMAL"
  THEN: <add-belief>
        /get (BELIEF-TABLE) recieve-my-lead-request/ 1

```

RULE interpret-my-lead-request-acknowledge

```

  IF :   GESTURE_Implementor = "HEAD"
                                and
        GESTURE_Motion = "NOD"
                                and
        GESTURE_emphasis = "INTENT"

```

and
 Negotiation-Mode = "PENDING-ACKNOWLEDGE"
 THEN: <add-belief>
 /get (BELIEF-TABLE) recieve-my-lead-acknowledge/ 1

RULE interpret-my-lead-request-acknowledge
 IF : GESTURE_Implementor = "HEAD"
 and
 GESTURE_Motion = "SHAKE"
 and
 GESTURE_emphasis = "INTENT"
 and
 Negotiation-Mode = "PENDING-ACKNOWLEDGE"
 THEN: <add-belief>
 /get (BELIEF-TABLE) recieve-my-lead-nack/ 1

Planner Rules

RULE adjust-my-dynamics
 IF : <surpassed-threshold>
 /get (BELIEF-TABLE) adjust-dynamics/
 and
 React-Mode = "EXTERNAL"
 THEN: <invoke-performer> adjust-event-dynamics
 <reset-belief> /get (BELIEF-TABLE) adjust-dynamics/

RULE adjust-my-tempo-and-location
 IF : <surpassed-threshold>
 /get (BELIEF-TABLE) adjust-tempo-and-location/
 and
 Tempo-Adjust-State = "ADJUST-MINOR"
 and
 React-Mode = "EXTERNAL"
 THEN: <determine-new-time-and-location>

```

<invoke-performer> adjust-timing-of-events
set Tempo-Adjust-State "NORMAL"

```

RULE recover-tempo-and-location

```

IF :   <surpassed-threshold>
        /get (BELIEF-TABLE) adjust-tempo-and-location/
        and
        Tempo-Adjust-State = "ADJUST-MAJOR"
        and
        React-Mode = "EXTERNAL"
THEN: send <Metrics> "ADJUST-TO-NEXT-MEASURE"
        set Tempo-Adjust-State "WAIT-FOR-NEXT-DOWNBEAT"

```

RULE notice-my-lead-request

```

IF :   <surpassed-threshold>
        /get (BELIEF-TABLE) receive-my-lead-request/
THEN: if Ambition >Ambition-Threshold then
        <set-past-threshold>
        /get (BELIEF-TABLE) return-my-lead-acknowledge/
    else
        <set-past-threshold>
        /get (BELIEF-TABLE) return-my-lead-no-acknowledge/
    fi
    <reset-belief>
    /get (BELIEF-TABLE) receive-my-lead-request/

```

RULE send-my-lead-acknowledge

```

IF :   <surpassed-threshold>
        /get (BELIEF-TABLE) return-my-lead-acknowledge/
THEN: <add-performance> <make-lead-acknowledge>
        set Play-Mode "PENDING-LEAD"
        <reset-belief>
        /get (BELIEF-TABLE) return-my-lead-acknowledge/

```

RULE send-my-lead-no-acknowledge

IF : <surpassed-threshold>
 /get (BELIEF-TABLE) return-my-lead-no-acknowledge/
 THEN: <add-performance> <make-lead-no-acknowledge>
 <reset-belief>
 /get (BELIEF-TABLE) return-my-lead-no-acknowledge/

RULE add-evidence-request-my-lead

IF : Ambition * Daring * Location >Lead-Threshold
 THEN: <add-belief> /get (BELIEF-TABLE) request-my-lead/ 1

RULE request-my-lead

IF : <surpassed-threshold>
 /get (BELIEF-TABLE) request-my-lead/
 THEN: <add-performance> <make-lead-request>
 set Negotiation-Mode "PENDING-ACKNOWLEDGE"
 <reset-belief>
 /get (BELIEF-TABLE) request-my-lead/

RULE receive-my-lead-acknowledge

IF : <surpassed-threshold>
 /get (BELIEF-TABLE) receive-my-lead-acknowledge/
 THEN: set Play-Mode "PENDING-LEAD"
 set Negotiation-Mode "NORMAL"
 <reset-belief>
 /get (BELIEF-TABLE) receive-my-lead-acknowledge/

RULE receive-my-lead-no-acknowledge
 IF : <surpassed-threshold>
 /get (BELIEF-TABLE) receive-my-lead-no-acknowledge/
 THEN: <reset-belief>
 /get (BELIEF-TABLE) return-my-lead-no-acknowledge/
 set Negotiation-Mode "NORMAL"

Performer Rules

The performer rules may be divided into two categories; task rules, and event timing rules. Task rules are heuristics that are invoked on demand by the planner. Timing rules are heuristics that find current events and gestures, putting them into the output queue.

Task Rules

RULE adjust-timing-of-events
 IF : MUSICAL-EVENT
 THEN: <adjust-timestamp> MUSICAL-EVENT Timing-Discrepancies
 <adjust-duration> MUSICAL-EVENT Tempo

RULE adjust-timing-of-gestures
 IF : GESTURE
 THEN: <adjust-timestamp> GESTURE Timing-Discrepancies

RULE adjust-event-dynamics
 IF : MUSICAL-EVENT
 THEN: <adjust-dynamics>
 MUSICAL-EVENT Dynamics-Discrepancies

Timing Rules

RULE remove-surpassed-events

IF : MUSICAL-EVENT.TIMESTAMP < MA.TIMESTAMP
THEN: remove MUSICAL-EVENT from performer*factbase

RULE perform-event

IF : MUSICAL-EVENT.TIMESTAMP = MA.TIMESTAMP
THEN: remove MUSICAL-EVENT from performer*factbase
add MUSICAL-EVENT to (Output-Queue)

RULE perform-gesture

IF : GESTURE.TIMESTAMP <= MA.TIMESTAMP
THEN: remove GESTURE from performer*factbase
add GESTURE to (Output-Queue)

BIBLIOGRAPHY

- Allen, J. F., & Perrault, C. F. (1980). Analyzing intention in utterances, Artificial Intelligence, 18, 143-178.
- Appelt, D. (1985). Planning english sentences, Cambridge, UK: Cambridge University Press.
- de Kleer, J. (1986). An assumption-based TMS Artificial Intelligence, 28, 127-162.
- Fry, C. (1980). Computer improvisation, Computer Music Journal, 4(3), 48-58.
- Fry, C. (1984). Flavors band, Computer Music Journal, 8(3), 273-305.
- Goodman, B. A. (1986). Reference identification and reference failure, Computational Linguistics, 12(4), 273-305.
- Hiller, L. (1969). Some compositional techniques involving the use of computers . In Von Foerster, H., Beauchamp, J. W. (Ed.), Music by computers (pp. 71-83). New York, NY: John Wiley And Sons.
- Hiller, L. (1981). Composing with computers: A progress report, Computer Music Journal, 5(4), 7-81.
- International MIDI Association. (1983). MIDI 1.0 specification. North Hollywood, CA: International MIDI Association.
- Kassler, M., & Howe, H. S. (Eds.). (1980). Computer music. In S. Sladie (Ed.), Grove's dictionary of music and musicians (pp. 613). London, UK: Macmillan.
- Laske, O. (1983). AI topics in computer-aided composition, Proceedings of the 1983 International Computer Music Conference, Berkley, CA.
- Loy, G. (1981). Notes on the implementation of MUSBOX, Computer Music Journal 5(1), 34-50.
- Roads, C. (1985). Research in music and artificial intelligence. Computing Surveys, Communications of the ACM, 17(2).
- Rodet, X., & Cointe, P. (1984). FORMES: Composition and the scheduling of processes, Computer Music Journal, 8(3), 32-50.
- Schottstaedt, B. (1983). PLA: A composers idea of a language, Computer Music Journal, 7(1), 11-20.

Searle, J. (1969). Speech Acts: An essay in the philosophy of language,
Cambridge, UK: Cambridge University Press.

Truax, B. (1977). The POD system of interactive composition programs,
Computer Music Journal, 1(3), 30-39.