1. INTRODUCTION

General-purpose programming languages like C and Lisp have several real-time extensions. That is, they provide a way of specifying what is to be done, but not when. Computer music languages, on the other hand, must provide real-time semantics. Some, such as NOISE [Collage 88], provide real-time semantics that allow interactive implementations but have highly limited compositionality constraints. Other languages such as OBJECT [Gottlieb 89] have very rich real-time semantics, but are difficult to implement efficiently.

The paper describes the real-time semantics of FORMULA, a language whose model is both semantically rich and implementable in interactive applications. FORMULA's model of real-time computation has two assumptions: 1) that all real-time actions are in the form of discrete events scheduled at specific times, and 2) that events are scheduled by non-generating processes, of which many can occur concurrently. These processes are able to advance in time within nested procedure calls. They have separate stacks, and context is maintained over time in activation records on the stack. There is no need for "state variables". All processes share a single address space.

For support of efficiency, we divide the model into three parts:

- Virtual time systems in which the non-generating processes specify the times of type events. These systems are mapped to real time by time-scheduling processes.
- Primitives and language structures for event scheduling and time control.
- Process creation, synchronization, and communication.

Three parts are discussed in Sections 2, 3 and respectively. Section 4 describes the implementation of FORMULA, and Section 5 summarizes the model and its significance.

2. PROCESSES, PROCESS GROUPS, AND VIRTUAL TIME

2.1. Process Groups

A process group is a syntactical collection of non-generating processes or other process groups. Under the mutual exclusion relationship, processes and process groups form a tree in which processes are leaves. The root of this tree is called the root group.

2.2. Virtual Time Systems and Value Deformations

A virtual time system is a percentile system for expressing event times. It is not used to have any relationship with real time. The canonical example is a virtual time in music, in which the time values are abstract and can be performed with different durations.

In FORMULA, a time deformation [19] provides a mapping between two virtual time systems, as between a virtual time system and real time. A TD defines a one-to-one mapping from one to the other, which can be controlled by process synchronization (as discussed in Section 4.1). A FORMULA TD allows time advance (positive duration) and can pass exactly the same "deformed" time advances. A simple TD is a
process with concurrent scheduling. Each time a time advance needs to be performed, the TD system is executed from wherever it left off last. It computes the desired interval, and returns both its and the flow of control back to the caller.

A TD could also be represented by a function, by a state data array, or by a state list of iterations. The process representation used in FORMULA is a standard. It provides a powerful and general way of defining TDS, but it imposes the limitation that TD's cannot be backed up. Once a time advance has been processed, no previous values are accessible.

TDS can be composed in two ways. The serial composition of two TDS is the result of using the output of the first as the input to the second. The parallel composition of a set of TDS's is like an input time advance by applying that input to all the TDS, connecting that "complex function" (input/output) forming the product of those, and returning the product times the original input. A TD expression is formed by applying serial and/or parallel composition, in any order, to a set of simple TDS's. Each process and process group is associated with a distinct virtual time system. The virtual time systems of the root group is real time.

Each system is mapped to be parent by a TD expression. The mapping from a process's virtual time system to real time is the serial composition of the TD expressions between the process and the root group.

FORMULA's virtual-time facility is applicable to many natural situations. It could be used, for example, to conveniently represent a performance in which two voices are subject to independent rules, yet the result is subject to a global articulation. TDS are similar to "time maps" [14][13] except that discontinuities are allowed, and the definition mechanism is more general.

2.4. Time Positions

at any instant in the course of program execution, each process or process group X has three positions in both its own time system and that of its parent group X: Y is the current time position of the object in its own system; (X) is the current time position of the object in the group that contains it. These terms are as follows:

- Each process X determines (X, Y) by explicitly calling a primitive to advance it (see section 3.1).
- (X, Y) is generated from (X, Y) by the TD expression between X and its parent group. Advance in (X, Y) is input to the TD expression, and the accumulated result is (X, Y).
- For a group G, (G, Y) is the maximum of (X, Y) over the elements V of G.

Changes in time positions of groups result from the discrete time advances requested by processors, and are therefore themselves discrete.

2.4.1. Deadlock Scheduling

The time positions of processes in different groups are not always directly comparable. However, the current process can always be found by traversing "parent elements" starting from the root group.

The global time position in the internal time of the root group is the real time position of the root group.

The scheduling of even-generating processes is determined by time position. The executing process is always the second position. This policy is called deadlock avoidance. It can be non-deterministic, since all changes in time position occur as a result of calls to system primitives.

The mechanism that combines deadlock scheduling and time information is easily described. When a process advances an interval time the following steps are done:

1. The time advance is input to the process's TD expression.
2. The result is added to the current time of the process.
3. If the internal time of the process's parent group has advanced as a result of (2), the advance is input to its TD expression, and the result is added to its external time.
4. The process repeats recursively up the tree, potentially to the root group. A control switch can then be made to the new external process.

ICMC 86 Proceedings
CAPABILITIES OF EVENT-GENERATING PROCESSES

Using detailed the framework in which event-generating processes exist, we move to the next task: language processing. They are now.

3.1. Events, Time Advance, and Future Action

An event-generating process produces output by performing the output directly, but by advancing the underlying system. This is not a mere performance counter to be called at a particular time (e.g., the next time corresponding to a current internal clock). This is done by calling

```
schedule_event(process, parameters)
```

where `process` denotes the current performance counter and `parameters` are the parameters to be passed to it. An event-generating process can advance the program counter position using

```
time_advance(difference)
```

By doing this, the ability to schedule additional events before the end time position.

An event-generating process can arrange for a procedure to be called in its current at a future-time position using

```
future_time_function(parameters, option)
```

Option is the same as the time in the schedule, as to be called. The procedure may call `schedule_event` or `schedule_time`, but not `time_advance`.

3.2. Events

The rationale for the above principles is as follows:

- **Formula:** a set of event-labeling systems in which the effect components and event performance are approximate. Schedule-event provides the logical progression that makes this possible.

- **Time advance, in conjunction with a stricter scheduling, allows an accurate implementation.** Whenever the next event advance at time position, the time can perform events scheduled within that time period because some events can be scheduled within the interval.

- **Future action allows events for computations that schedule events or that change process state to be scheduled in a consistent manner.** This makes it possible, for example, to have high-level constructs like `schedule-event` that schedule events in the future with time advanced to the times of those events (as required by the keyboard + time-advanced scheduling by analyzing positions).

3.3. Time Control Features

Conventional language has structures like `let` and `continue` to control execution time. These were in earlier versions (some statements) into high-level structures. In the presence of these, and `time`, it is possible to define an analogous set of time control structures. These can be used to build program structures with known time behavior and concerns with memory behavior.

A time control structure is an object of a rule. It states that the amount of normal time consumed by the enclosed program block (i.e., the sum of its time advance) is not limited to upper or lower bound. FORMULA has three core control structures:

- **action():** a method that specifies that the enclosed code is to continue at most `n` units of virtual time. When the block is executed, the upper limit (time position + `n`) is recorded. If there is a block, a call to `time` advances a time that would cause the program to exceed that limit, then `time` advances itself with a smaller argument that advances strictly to the limit, and control is transferred to the statement following `action()`. If the first time advance was called from within another control block, their activation records are removed from the stack. Calling the block because of the recorded limit has no effect on future states, they are not checked. If there is a block, the lower limit (time position + `1`) is executed. If else `action()` is reached, the time position is

37

ICMC 86 Proceedings
When a new process is created by another event-generating process, it joins the process group of the creator, and its initial external time is that of its creator. A new process created by an outside process (i.e., a new input branch) is created in the root group. There are several choices for its initial external time. For normal purposes, it is desirable to use the real time of the input event that caused the process creation (this may be slightly, but significantly, different from the time of the call to the process creation service). Implementation issues related to this are discussed in [Kelly, 1989]. The idle service for event-generating processes advances in time until there are no more future actions to be performed, and then the process is removed. If its parent group is now empty, then it is also removed. This is done recursively, up to but not including the root group.

4.3 Sleep and Wakeup

When a process sleeps (while waiting to read a message, for example) it is detached from the tree. If it has no siblings, then its parent group is also detached; this may propagate up the tree (similarly, a parent group is detached whenever an existing process has only detached siblings). When a process or process group is detached, it ceases to advance in time. The future actions of a process are not executed while it is detached. However, the object may have siblings, and these must be allowed to continue to advance. This requires detaching the object's virtual time system from that of its parent group, i.e., synchronizing a jump discontinuously in the mapping between the two time systems.

If a sleeping process is awakened and its parent group is not detached, it is reattached to that group. It is possible that the parent, and perhaps some of its ancestor groups, are currently detached from the tree. If so, the highest such group is reattached to its parent group. In either case, the reattached object is reestablished with the earliest element of the group, as explained above.
5. IMPLEMENTATION

The section briefly describes how the language features described in the previous sections are implemented in FORMLA.

5.1. Coarse Time Execution

FORMLA can run programs with only modest performance hindrance, assuming careful application in numerical computations of time. At the user level, all time intervals are represented as normal numbers. These are immediately converted to integers by a counter that approximates the current time on a process basis. Time deconstructions outside or large time intervals, and also approximate times, are handled similarly. Finally, attention to the global time precision is subordinated to a final value with error accumulation. Adjustment of the scaling parameter is an efficient means of global performance control.

5.2. Data Structures for Process Scheduling

Each process is represented by a process object containing its start, process state, context, and process attributes. Each process group is represented by a group record. Each element of a group may be considered to implement a finite state machine in which each state has a field describing the delay of the current time beyond that of the previous group element.

Each element of a group contains the previous positions of the relevant processes and the relevant information about the global time precision. The groups are organized in order of increasing priority whenever the object advances in time, providing overall compatibility of all fixed delays. The groups are organized in parallel.

With these data structures, the scheduling algorithms described in section 2 can be implemented very effectively.

5.3. Synchronization with Event Buffers

The event tracking mechanism is integrated with the event buffer model described in [Nicol]. The event buffer model is a process in an event tracking paradigm (e.g., event circle or time slot). In the event buffer model, the event is the deadline, and the buffer is the process. The event buffer model calls the event buffer's deadline function in the event buffer model.
This service may limit the difference between global time position and real time by passing the deadline process to "sleep" (in the sense of the external scheduling paradigm) while it is too far ahead of real time.

5.4 Future Action Implementation
A future action is represented by a record containing the action's procedure address and parameters. Each record has an associated future action queue in which these records are linked in an incremental delay queue. Time advance is modeled as a routine that examines this queue, performing the actions and moving calls to the deadline scheduler's time advance as necessary.

5.5 Time Control Structure Implementation
Each event-generating process has two stacks of records describing the time control structure: a main stack and a mini-stack. Each record contains a time limit, the dynamic testing level, and possibly a branch address and stack level. Because of recursion, several records on top of the stack may refer to a single (terminal) time control structure.

To implement maximization, time advancement is modeled as a routine that checks the top element on the main stack. If the time limit would be exceeded, a "summarized" (low-level) time advance is done, records from the main stack with lower testing levels are popped, the call stack is restored to the correct level, and control is transferred to beyond the mainstack. Mini stack pops the mainstack. Minized pops the mini-stack and does a time advance if necessary. Minized examines the mainstack top and either branches or pops the mini-stack.

6. CONCLUSION

6.1 The Process-Based Approach
The FORMULA model is based on the philosophy that all action components are processes (in the sense of section 4), and all definitions are processes that can potentially be run as processes. Structural components, however, are constrained to be process-oriented.

The advantage of this approach is that since all components are procedural, they can be embedded in control structures, can be parameterized, can dynamically make random decisions, and can dynamically communicate with other systems. We also feel that processes are a naturally appropriate, e.g. that the generation of tempo fluctuations in a musical performance is better described as an active, state-behaving process than as a table of values.

The disadvantages of the approach are that the duration of a program segment, e.g., can't always be known in advance, and that computations can't be backed up. We believe that the advantages greatly outweigh the disadvantages.

6.2 Different Meanings of "Time Advance"
FORMULA's real-time semantics can perhaps be best understood by observing that there are four notions of "time advance", each defined in terms of the previous. Starting from the bottom, they are:

(1) The event-driven time advance, which is the only one that advances in real time, and indicates an advance in global time position.

(2) The deadline scheduling time advance is called by individual event-generating processes, and triggers on violation of some deadlines or a context switch to a new event process.

(3) The future action time advance corresponds to the lower level time advance with calls to the future actions routine that does within the local advance.

(4) The time control structure time advance checks for exceeded maximizations into limits.

The lowest time advance is the one accessible to user-defined processes. It is interesting to note that the last two notions could be defined in either order. The chosen order is more efficient because time advances are evaluated at a lower level.

6.3 Dependencies
The various language features described in this paper are not available. They can potentially be applied, together or separately, to other computer music languages. We now consider the extent of the applicability.

ICMC 86 Proceedings 40
The idea of a set of virtual time systems has been incorporated in
other languages using time maps [Jul 80]. These deictic systems are
more powerful than time maps, but the point of view of extension,
but were limited because they can't be reversed or hooked up. For
real-time systems, they require efficient context switching.

Future actions, time control structures and timed messages are all
dependent on the model of event-generating processes that involve
in virtual time. For example, time control symmetry might exist in
PLA [Kung and Watanabe 82] but not in MOUSE [Collage 84] or
PLAYFIT [Jay 85] where the latter languages can't advance in time
within a procedure.

REFERENCES

Collage, D., The Mouse Users Guide, Internal report, School of
Music, Univ. of Victoria, Victoria, B.C., Canada (1980).

Dunlop, R.B., Arlet b., a functional language for real-time control,
Proceedings of the 1984 ACM Symposium on Lisp and Functional

Jay, D., Eventful timing in computer music, Computer Music

Kerla, R. and D.P. Anderson, Timing accuracy and response time
in interactive systems, These Proceedings.

Lay, G., Planner - Extensive in the C Programming Language for
Parallel Processing and Real-Time Control, Unpublished
manuscript, CENL, University of CA, UCSE 1985.

Sekimoto, K., PLA Reference Manual, CTYRMA, Stanford Univer-
sity (June 1982).