RAGA: Musical Gantt Charts for Scheduling in Distributed Real Time Systems

Virginia M. Lo and Gurdeep Singh Pall

CIS-TR-89-20
January, 1991

Department of Computer and Information Science
University of Oregon

RAGA: Musical Gantt Charts for Scheduling in Distributed Real Time Systems

Virginia M. Lo and Gurdeep Singh Pall

Dept. of Computer Science
University of Oregon
Eugene, OR 97403-1202
lo@cs.uoregon.edu

October 20, 1989

Abstract

We propose a simple extension to Gantt charts, called RAGA scores, for use in distributed real time scheduling. RAGA scores use a small set of symbols borrowed from musical notation to enrich the expressive power of the Gantt chart. These symbols enable the timing constraints of real time tasks to be displayed along with the schedule itself. Because of the criticality of timing constraints in real time systems, it is important to be able to visualize these constraints and the schedule simultaneously.

The RAGA score is encapsulated as an abstract data type (ADT) that can be used as a tool in the design and visualization of static and dynamic scheduling algorithms, in the display of real time schedules for performance evaluation through simulation, and for historical records of actual schedules for performance evaluation through 'execution traces.'

1 Introduction

The problem of scheduling tasks in real time distributed systems differs significantly from both the classical problem of scheduling in general purpose uniprocessor systems and from the problem of scheduling in distributed and parallel computing systems. Real time scheduling involves new dimensions with respect to goals, degree of difficulty, algorithm design issues, performance analysis, and the notion of correctness. Most notably,**distributed real time scheduling algorithms must meet the correctness criteria of timing constraints such as deadlines, periodicity, and precedence on individual processors as well as across processors.** In addition, these algorithms must address the problems of coherence and consistency, sharing and resource utilization, communication and synchronization across multiple CPUs [7].

The Gantt chart has been an important aid over the past 20 years for visualizing the space (processors) and time dimensions for schedules, most notably in the domain of deterministic scheduling on shared memory multiprocessors. It has been used for scheduling in distributed systems and message-passing multicomputers by modeling communication overhead as elapsed execution time. In the area of real time distributed systems, the use of the Gantt chart continues to be widespread, particularly for static offline scheduling. Indeed, the fundamental format and inherent simplicity of the Gantt chart have insured its successful use across these problem domains.

The limitation of the Gantt chart for real time distributed systems lies in its inability to clearly express the relationships among the scheduled tasks. We can see this with the help of an example. In Fig 1, the schedule of a group of tasks is presented in a Gantt Chart. Because timing constraints are not explicitly represented, many uncertainties about the schedule exist, making it difficult to evaluate the performance of the underlying scheduling algorithm. Has task C been scheduled across the processors coincidently, or was that a requirement? Is there a reason for task D to be scheduled in this manner? Is A a periodic task or is its repeated occurrence due to independent external events?

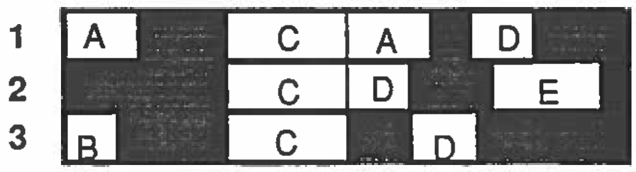


Figure 1: Gantt Chart

We wish to propose a simple extension to Gantt charts, called RAGA [*] scores, for use in distributed real time scheduling. RAGA scores use a small set of symbols borrowed from musical notation to enrich the expressive power of the Gantt chart. These symbols enable the timing constraints of real time tasks to be displayed along with the schedule itself. Because of the criticality of timing constraints in real time systems, it is important to be able to visualize these constraints and the schedule simultaneously.

The RAGA score is encapsulated as an abstract data type (ADT) which includes operations for calibrating, modifying, and displaying the score. The RAGA score and ADT can serve as a useful tool for

- the design and visualization of static and dynamic scheduling algorithms,
- the display of real time schedules for performance evaluation through simulation, and
- historical records of actual schedules for performance evaluation through 'execution traces.'

We wish to stress the fact that the RAGA score is not intended to replace Gantt charts. RAGA is intended to be an elaboration of Gantt charts to be used when timing constraints need to be represented and visualized. When constraint information is not needed, the RAGA display can be filtered so that it looks identical to a Gantt chart.

In Section 2 of this paper, we introduce the components of the RAGA score. In Section 3, we compare the RAGA score and the Gantt chart using an example of an athlete monitoring system. Section 4 introduces the RAGA ADT and discusses its utility in the design of both static and dynamic scheduling algorithms. In Section 5, we illustrate the use of the ADT for two well-known real time scheduling algorithms, rate monotonic [6] and bidding and focused addressing [8]. Section 6 contains conclusions and future plans for RAGA.

2 Components of RAGA score

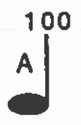
The RAGA score is an extension to the Gantt chart for use in scheduling real time distributed and parallel applications. The RAGA score utilizes a small set of symbols from musical notation to express the timing constraints among tasks in a real time application. Like the Gantt chart, the RAGA score represents time along the X axis and space/processors along the Y axis. Tasks are

[*]RAGA means 'tune or melody' in Hindi and could be an acronym for Real time Applications Gantt Chart Annotation.

represented as 'musical notes' and the relationships among tasks by musical symbols. Below we describe the fundamental components of the RAGA score - tasks, events, and timing constraints. We show the RAGA notation for each of these components and express the semantics of the timing constraints using Real Time Logic (RTL) , a formal semantics for describing timing constraints under the event-action model [3].

2.1 Tasks/Actions

A task (or action in the event-action model) is the unit of work to be scheduled in the real time system. Each individual task is represented as a musical note in the RAGA score. A note is attributed with the name and the duration of the task. In the Gantt chart, each task "blocks out" time corresponding to its duration. In the RAGA score, duration can be blocked out in a similar manner (described later) or it can be represented as an attribute.



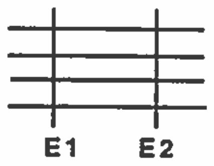
Task A

2.2 Events

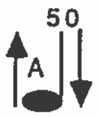
Events are occurrences in time which are of significance to the real time system and typically trigger the execution of one or more tasks. Events are of two types external and internal [3].

External events are generated (caused) by the environment of the real time system. For example, an increase in the temperature of a space shuttle sensor may cause certain monitoring and adjustment tasks to be performed. Or, the space shuttle may need to be refueled from the auxiliary cylinders every hour. In the latter case, hourly clock interrupts are the external events. External events are represented in the RAGA score by a vertical bar through the score. Each event is labeled with a descriptive name for the occurrence it represents.

Internal events are temporal markers within the schedule and may or may not trigger the execution of tasks. In the RAGA score two types of internal events are represented. They correspond to the *Start* and *Finish* events of the tasks. These are of importance in scheduling tasks which have relationships between them such as precedence. Internal events denoting the start event of a task are represented with an upward arrow on the left of the musical note (task). Similarly, finish (internal) events are denoted by a downward arrow to the right of the musical note (task) Internal events are not labeled. Only those internal events which are of significance to the timing analysis (hence, scheduling) are displayed in the musical score. For example if the completion of a task is of no significance to other tasks, its finish event will not be displayed.



External Events



Start Stop Events

2.3 Timing Constraints

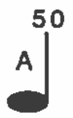
The essence of real time applications are the timing constraints on the tasks to be scheduled. RAGA supports a rich set of timing constraints in order to support a wide range of real time applications and to provide flexibility for the future. In some cases, we have extended common existing timing constraints to a more general form. In this section, we give the RAGA notation for each timing constraint and define its semantics using Real Time Logic. The RTL primitives are included in the Appendix for reference purposes.

2.3.1 Duration

The estimated execution time of a task is its durational constraint. In RAGA duration is represented as an attribute of the associated task. This constraint can be expressed in RTL as -

$$\forall i @(\uparrow A, i) + D = @(\downarrow A, i) \tag{1}$$

where A is the task in question and D time units is its duration.



Duration

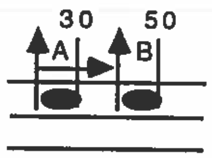
2.3.2 Precedence

Precedence is a partial temporal ordering on a set of tasks [9] using the transitive binary relationships **BEFORE** and **AFTER**. In RAGA, the BEFORE relation is represented as a left arrow between events or tasks. The AFTER relation is represented as a right arrow between events or tasks. Constraint "A starts BEFORE B" is represented in RTL as -

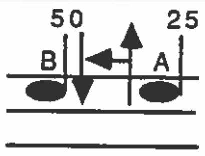
$$\forall i @(\uparrow A, i) < @(\uparrow B, i) \tag{2}$$

Constraint "A starts AFTER B finishes" is represented in RTL as -

$$\forall i @(\uparrow A, i) > @(\downarrow B, i) \tag{3}$$



A starts BEFORE B



A starts AFTER B finishes

2.3.3 Deadlines

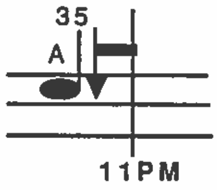
A deadline specifies a time (external or internal) by which the task or set of tasks must complete. Deadlines can be specified completely with points in time and the precedence operator BEFORE. In RAGA, deadlines are represented by a heavy horizontal bar from the finish event of the task to the deadline event. We chose this notation over the precedence arrow (which would have been sufficient) because of the importance of deadlines in real time applications.

Absolute Deadlines: The deadline is specified with respect to an absolute point in time. For example, a typical absolute deadline would be - A must finish BEFORE 11PM, where 11PM is an external event. In RTL we would express this as -

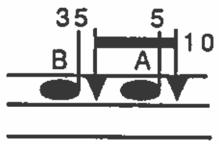
$$\forall i @(\downarrow A, i) <= @(\Omega 11PM, i) \tag{4}$$

Relative Deadlines: Relative deadlines are specified with respect to events which are not fixed in time. For example the completion of a task A may have a deadline of 10 milliseconds after B completes, an internal event. In RTL we would express this as -

$$\forall i @(\downarrow B, i) + 10 <= @(\downarrow A, i) \tag{5}$$



A must finish before 11PM



A must finish within 10 ms of B

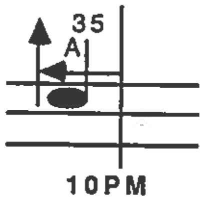
2.3.4 Start Times

Tasks may be constrained to start BEFORE, AFTER, or AT certain points in time. As with deadlines these points in time may be absolute or relative to other events. The start time of a task constrains the scheduler from scheduling the task before a certain point in time across all processors in the system. Start times are represented in the same manner as precedence relations are in RAGA: with a left arrow (BEFORE), right arrow (AFTER), or extended equal sign (AT) from the start event of the task to the specified event. **Absolute Start Times:** The start time is specified with respect to an absolute point in time. For example, A must start BEFORE 10 PM. In RTL, we would express this as -

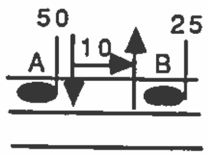
$$\forall i @(\uparrow A, i) < @(\Omega 10PM, i) \tag{6}$$

Relative Start Times: The start time is specified with respect to events which do not have a definite time of occurrence. For example, B must start 10 milliseconds AFTER completion of A.

$$\forall i @(\downarrow A, i) + 10 < @(\uparrow B, i) \tag{7}$$



A must start BEFORE 10PM



B starts AFTER 10 ms of A finishes

2.3.5 Periodicity

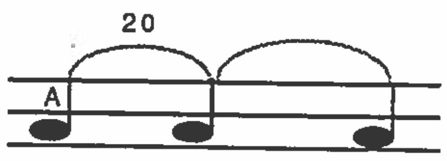
Periodic tasks are typical of real time applications. A periodic task must be repeatedly scheduled at a regular interval - its period. RAGA supports two types of periodicity: *rigid* and *floating*.

Rigid periodicity is the constraint in which the next instance of the periodic task must be scheduled exactly Δ time units after the previous instance. In RTL if a task A with a rigid periodicity of Δ time units is to be executed while the state predicate $PRED$ holds true:

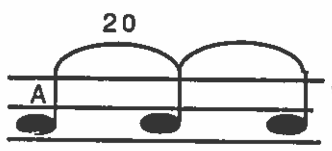
$$\begin{aligned} &\forall x \forall y \forall n \exists i PRED[x, y) \wedge y - x > n * \Delta \rightarrow x + n * \Delta \\ &\qquad = @(\uparrow A, i) \wedge @(\downarrow A, i) \leq x + n * \Delta + \Delta \end{aligned} \tag{8}$$

Floating periodicity means that the next instance of the repeatedly scheduled task must occur sometime within the next Δ window of time. Floating periodicity would be expressed in RTL as -

$$\begin{aligned} &\forall x \forall y \forall n \exists i PRED[x, y) \wedge y - x > n * \Delta \rightarrow x + n * \Delta \\ &\qquad \leq @(\uparrow A, i) \wedge @(\downarrow A, i) \leq x + n * \Delta + \Delta \end{aligned} \tag{9}$$



Floating Periodic Task A



Rigid Periodic Task A

2.3.6 Synchrony

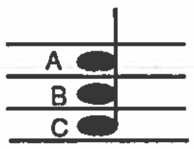
In distributed or multiprocessor real time systems, there may be a requirement that two or more tasks be executed "synchronously" across processors. The synchrony requirement is classified in RAGA as:

Tight Synchrony: When the requirements are that the tasks in N need to be executed precisely in parallel then the constraint is referred to as *tight synchrony*. In RAGA, tight synchrony is represented using the notation for chords in music: a vertical bar through the synchronous notes (tasks). If tasks A, B and C have to be executed in tight synchrony, we would express this in RTL as -

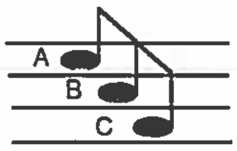
$$\forall i @(\uparrow A, i) = @(\uparrow B, i) \wedge @(\uparrow A, i) = @(\uparrow C, i) \tag{10}$$

Loose Synchrony: When the requirements for synchrony are not rigid then the constraint is referred to as *loose synchrony.* Here the tasks are executed in as much synchrony as possible within a time window. The RAGA representation of loose synchrony is a collection of line segments connecting the affected tasks. For example A and B must be executed within 100 milliseconds in loose synchrony after event E. This is expressed as -

$$\begin{gathered}\forall i @(\Omega E, i) <= @(\uparrow A, i) \wedge @(\downarrow A, i) <= @(\Omega E, i) + 100 \\ \forall i @(\Omega E, i) <= @(\uparrow B, i) \wedge @(\downarrow B, i) <= @(\Omega E, i) + 100\end{gathered} \tag{11}$$



Tight Synchrony of A, B and C

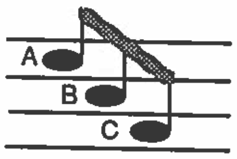


Loose Synchrony of A, B and C

2.3.7 Mutual Exclusion

This constraint restricts the scheduling of a set of tasks such that no two tasks can overlap in time. In real time applications this constraint may be seen as a *resource* constraint [2] [12]. This constraint is represented in RAGA with a dotted line connecting the tasks that need to be mutually excluded. If A and B are to be executed in mutual exclusion, we would express it in RTL as -

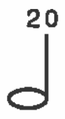
$$\forall i @(\downarrow A, i) <= @(\uparrow B, i) \vee @(\downarrow B, i) <= @(\uparrow A, i) \tag{12}$$



Mutual Exclusion of A, B and C

2.3.8 Idle Time

In the RAGA score, we cannot always *see* the duration of a task as in a Gantt chart. This is true when the duration is represented as an attribute and the task does not "block" the time it occupies on the processor. Therefore, in order to represent idle time on the processor, we use an explicit symbol: an empty note. [*] The empty note is also attributed with its duration.



Idle Time

3 Example - Athlete Body Monitoring System

In order to illustrate the power of the RAGA score we look at an athlete body monitoring system. We first give the specifications of this system then we display the schedule generated by some arbitrary algorithm represented as a Gantt chart, and lastly we present the schedule represented as a RAGA score.

3.1 Specifications for the athlete body monitoring system

A distributed real time system monitors different body parameters in a group of 3 athletes such that each athlete is monitored by one node.

- Task 1: Every 80 ms. (floating) the body temperature of athletes 1 and 3 must be measured for 20 ms.
- Task 2: Every 200 ms. (rigid) the heart beat of athlete 2 should be measured for 40 ms.
- Task 3: Every 500 ms. (rigid) the blood pressure must be measured for athlete 1 for 50 ms.
- Task 4: Every 300 ms. (floating) the fluid level in the lungs of athlete 2 should be measured for 10 ms.
- Task 5: Every 250 ms. (rigid) the ECG is sampled for athlete 3 for 30 ms.

[*]An alternative RAGA representation for idle time could be a musical *rest symbol*.

- Task 6: 150 ms. after start of the system each athlete's blood pressure should be measured in tight synchrony for 30 ms.
- Tasks 7, 8, 9: 100 ms. after start of the system, the adrenaline level is measured on athlete 1 for 20 ms, on athlete 2 for 30 ms, and on athlete 3 for 25 ms.
- Task 10: After completion of task 6, each node must transfer data to a shared disk, requiring 20 ms. (mutual exclusion)

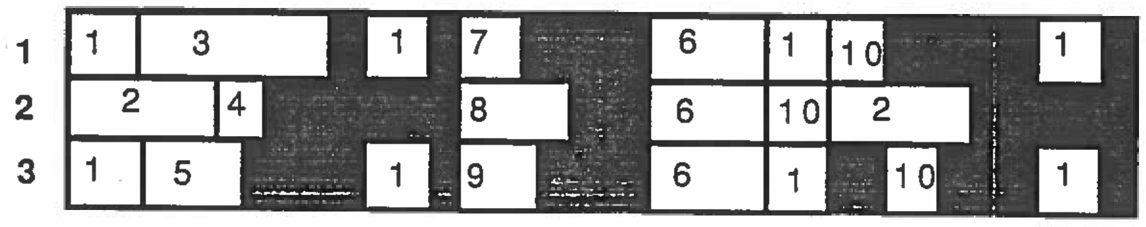
The schedule generated for the above specification is given in Figure 2(a). The same schedule expressed using the RAGA notation is given in Figure 2(b). Figure 2(c) shows a RAGA score that has been displayed through a Gantt chart *filter* and with only one task selected for constraint information.

The power of the RAGA score becomes evident here. Because the periodicity of task 1 is floating, it is cannot be known from the Gantt chart that the task is periodic rather than repeatedly scheduled for external reasons. In addition, the blocks of unused times before the second occurrence of task 1 are not explained in the Gantt chart. In the RAGA score it is clear that task 1 is a task with floating periodicity. The dependence of the start times of tasks 7, 8 and 9 on the 100 ms. event can only be seen in the RAGA score. Further, the simultaneous scheduling of task 6 is not accidental (which could be hypothesized from the Gantt chart) but a requirement for synchrony (which is made unequivocal in the RAGA score). The scheduling of task 10 appears to be arbitrary in the Gantt chart; from the RAGA score it is clearly due to the mutual exclusion constraint. From another perspective, if task 10 was scheduled at the same time across processors we would not be able to detect this violation by looking at the Gantt chart itself. Finally, because the RAGA score is an extension of Gantt charts, Figures 2(a) and 2(c) are simply selective, filtered views of the full RAGA score of 2(b).

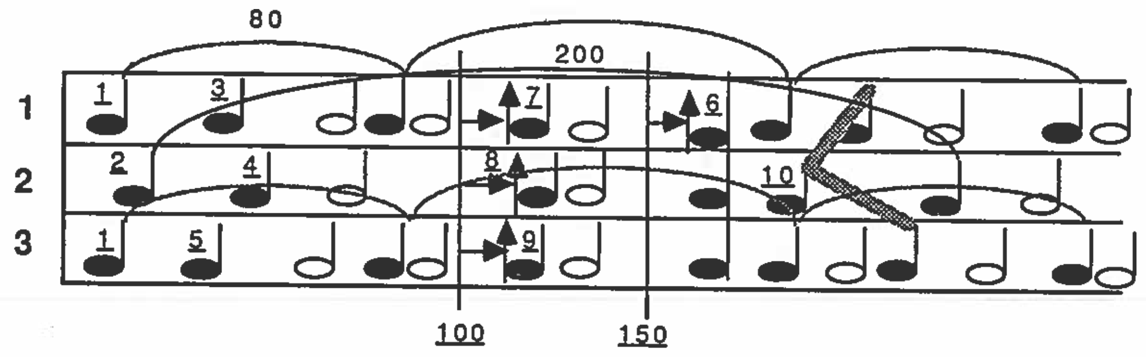
4 RAGA ADT

4.1 Purpose

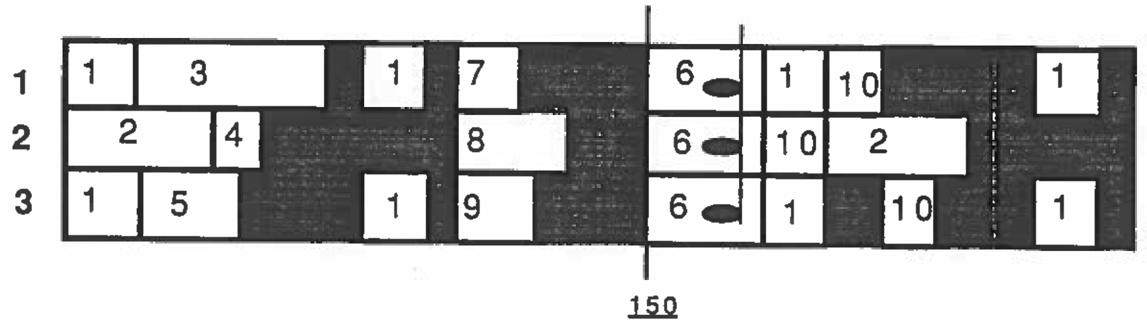
The RAGA Abstract Data Type was designed to encapsulate and define the RAGA score for use in real time systems development. As mentioned earlier, the RAGA score and RAGA ADT are designed for use in the design and implementation of static and dynamic scheduling algorithms, for simulation purposes, and as a structure for recording the runtime history of real time applications.



(a) Gantt Chart



(b) RAGA Score



(c) Selective Filtering of RAGA Score

Figure 2: Gantt Chart and RAGA Score for the Athlete Body Monitoring Schedule

4.2 Methods

In this section, we give a proposed set of primitive operations (methods) for the RAGA ADT. Included are methods for calibration of the ADT, location of time slots, insertion and deletion of tasks, and display of the RAGA score. The Task-Descriptor data structure contains information about the task to be scheduled: task id, duration, period etc. The Event-Descriptor contains information about the type of event (internal or external), it name, and relations BEFORE, AFTER, and AT and pointers to the associated tasks. Some of the methods allow for a time window to be specified. RAGA windows include:

IDLE-WINDOW - the time window from the beginning of earliest idle time on the specified processor(s) to ∞

FUTURE-WINDOW which refers to the time window from the end of the latest task scheduled on the specified processor(s) to ∞

SCHEDULED-WINDOW which refers to the window from the beginning of the earliest scheduled task on any processor to the end of the latest scheduled task on any processor

and user-specified windows - time ranges $[T_1, T_2]$.

The error codes returned specify the timing constraint(s) that could not be satisfied.

4.3 Calibrate

The RAGA ADT is initialized by calibrating it with respect to time and processors. The user must specify the number of processors and the time unit. Calibration of the schedule in time units conforms to the principle of *segmentation* of time [10]. Once the ADT is calibrated, requests for modification of schedules must be feasible within the calibration. Later extensions to the calibration method may include hierarchical notions of time.

Method Calibrate (Number-of-Processors, Time-Units)

Return Value 0 : if calibrate succeeds, Err-Code : if calibrate fails.

4.4 Find-Slot

These methods allows algorithms to find slots in the schedule for inserting tasks. The inputs are the Task-Descriptor and the Event-Descriptor, the target PROCESSOR, and an optional time WINDOW in which to search for the slot. Each Find-Slot method returns the earliest time in which a free slot exists in the window on the specified processor and which satisfies the timing constraints.

Method Find-Slot-Single (Task-Description, Event-Descriptor, Processor, Window)

Method Find-Slot-Synch-T(Task-Descriptor, Event-Descriptor, Processor, Window)

Method Find-Slot-Synch-L(Task-Descriptor, Event-Descriptor, Processor, Window)

Method Find-Slot-Periodic-R (Task-Descriptor, Event-Descriptor, Processor, Window)

Method Find-Slot-Periodic-F (Task-Descriptor, Event-Descriptor, Processor, Window)

Method Find-Slot-Mut-Ex (Task-Descriptor, Event-Descriptor, Processor, Window)

Return Value Time and Processor, if slot found, Err-Code otherwise.

4.5 Insert-Task

These methods are used for inserting the tasks in the schedule once the slots for them have been found. As in Find-Slot, we have these methods for each type of task.

Method Insert-Slot-Single (Task-Descriptor, Time, Processor)

Method Insert-Slot-Synch-T(Task-Descriptor, Time, Processor)

Method Insert-Slot-Synch-L(Task-Descriptor, Time, Processor)

Method Insert-Slot-Periodic-R (Task-Descriptor, Time, Processor)

Method Insert-Slot-Periodic-F (Task-Descriptor, Time, Processor)

Method Insert-Slot-Mut-Ex (Task-Descriptor, Time, Processor)

Return Value 0, if slot found, Err-Code otherwise.

4.6 Modify-Schedule

RAGA ADT allows the schedule to be modified by displacing tasks within the schedule. This is allowed as long as the constraints are not violated. The displacements are possible for tasks with Floating periodicity, mutual exclusion, or Loose synchrony constraints. The task(s) are displaced by Δ time units.

Method Modify-Schedule (Task-Descriptor, Δ)

Return Value 0, if operation successful, Err-Code otherwise.

4.7 Delete-Task

This method allows deletion of tasks form the schedule. Task(s) may be deleted in the same way they were inserted.

Method Delete-Slot-Single (Task-Descriptor, Time, Processor)

Method Delete-Slot-Synch-T(Task-Descriptor, Time, Processor)

Method Delete-Slot-Synch-L(Task-Descriptor, Time, Processor)

Method Delete-Slot-Periodic-R (Task-Descriptor, Time, Processor)

Method Delete-Slot-Periodic-F (Task-Descriptor, Time, Processor)

Method Delete-Slot-Mut-Ex (Task-Descriptor, Time, Processor)

Return Value 0, if operation successful, Err-Code otherwise.

4.8 Display Schedule

These methods allow the user to display the RAGA score. The display methods provide filters that support a variety of views of the schedule. One important view is the Gantt Chart View which displays a RAGA score exactly as it would appear in a Gantt chart (using the block representation of tasks and omitting all timing constraint information). We provide methods to display the schedule by tasks, by time and by processor. OPTIONS allows the user to select the specific set of constraints that he/she wishes to view.

Method Display-Schedule-By-Task (Task Descriptor, OPTIONS)

Method Display-Schedule-By-Time (Window, OPTIONS)

Method Display-Schedule-By-Processor (Processor, OPTIONS)

5 The Use of RAGA for Scheduling in Real Time Systems

The RAGA ADT has been designed so that any scheduling algorithm can be used for building the schedule. In other words we have avoided building any policies into the ADT itself. The RAGA ADT can be used within offline schedulability analyzers to store the pre-computed static schedule. At runtime each processor in the distributed real time system receives a copy of the system-wide schedule for use by the dispatcher and/or for use by the distributed scheduler for scheduling sporadic tasks at runtime.

We illustrate the use of the RAGA ADT by giving pseudo-code for two well-known real time scheduling algorithms. The first is an off-line scheduling algorithm and the second is a dynamic run-time scheduling algorithm for sporadic tasks. Again, we stress that the ADT does not dictate policy. Its purpose is to serve as an efficient data structure with a well-defined interface for storing, querying, and displaying the schedules.

In addition to its use in scheduling algorithms, the RAGA ADT can support other activities involved in real-time system development. RAGA can be used as a visualization tool for simulation and system monitoring such as that provided in the ARTS ARM (Advanced Real time Monitor) [11].

```
procedure Static-Scheduler ()
    begin
    whileunscheduled tasks remain
        repeat
            Select-Task-with-highest-periodicity
            slot = Find-Slot-Periodic-R (Task-Descriptor, ANY-PROCESSOR, IDLE-WINDOW)
            if found
                Insert-Task-Periodic-R (Task-Descriptor, slot→time, slot→processor);
            else
                Cannot Schedule; Print error message
    Display-Schedule-By-Time(SCHEDULED-WINDOW, ALL-OPTIONS)
    end
```

Figure 3: Static Scheduling Algorithm using the RAGA ADT

5.1 Static Offline Scheduling

The pseudo code ~~below~~ above is distributed version of the Rate-Monotonic static scheduling algorithm [6] which orders tasks in non-increasing order by period. For uniprocessor systems, the Rate-Monotonic Algorithm has been shown to be optimal in the sense that no other fixed priority algorithm can schedule a set of tasks that can be scheduled by the Rate-Monotonic Algorithm. The distributed version shown in Figure 3 selects the processor with the earliest free slot by specifying the window IDLE-WINDOW in the call to Find-Slot. Recall that IDLE-WINDOW is defined as the window from the beginning of the earliest idle time on the specified processor(s) to ∞.

5.2 Dynamic Scheduling

The Focused Addressing Scheduling Algorithm [8] is a dynamic scheduling algorithm in which sporadic tasks are dynamically scheduled across the system based on prior (partial) knowledge of system work loads. The Focused Addressing Algorithm assumes that sporadic tasks are non-periodic tasks with deadlines. The scheduler tries to schedule the task locally, if possible. If it cannot, it uses the partial information available to it to send the task to a "focused" processor that is likely to be able to schedule the task. In addition, it sends out a *request-for-bids* to subset of the other processors in the system. The bidders return their bids to the focused processor. The focused processor receives the task and also tries to schedule it locally. If it cannot, it selects the best of the bids. Portions of the Focused Addressing Algorithm using the RAGA ADT are given in Figure 4 below. In the pseudo-code, we have simplified the criteria for selection of the focused processor to choose the processor with the earliest available slot. The Find-Slot method can be used to schedule a task locally, to select the focused processor, and to select bidders because it contains the *static* schedule of remote processors and can be updated to reflect their current load.*

6 Conclusions and Future Work

In this paper, we have introduced the RAGA score as a tool for use in the design, implementation, and testing of real time scheduling algorithms. The RAGA ADT provides a well-defined data structure for representation of both the timing constraints associated with real time tasks and the schedules produced for these tasks. The musical notation in RAGA allows the constraints and the schedules to be displayed in a natural and easily comprehensible format.

Continuing work on RAGA includes the following areas of investigation:

- **Implementation:**Because we intend the RAGA ADT to be usable as a run-time data structure in dynamic scheduling algorithms for real time kernels, it is necessary to develop highly efficient data structures and algorithms for the RAGA ADT, especially for the Find-Slot method.
- **Testing:** We plan to test the user appeal of the RAGA musical notation on both naive users and expert real time users. We also plan to refine the RAGA ADT by implementing a wide range of scheduling algorithms, evaluating its strengths and shortcomings, and making necessary modifications.
- **Notions of time and space:** We plan to examine results from the AI and distributed systems community about notions of time and space: time hierarchies, interval-based vs. point-based time representations [1], virtual time – to see what may be applicable to RAGA. We are interested in formalizing the binding of a RAGA ADT to real time and real processors at run time.
- **Communication and resource constraints:** These are two very important aspects of real time scheduling [4] [5] which RAGA does not currently address.

*Keeping information consistent and up-to-date across processors in a distributed system is an area of research in itself.

```
procedure Dynamic-Scheduler (Task-Descriptor)
    begin
        slot = Find-Slot-Single (Task-Descriptor, HERE, IDLE-WINDOW)
        if guaranteed (Task-Descriptor, slot)
            Insert-Task-Single (Task-Descriptor,
                slot→time, slot→processor);
        else
            slot = Find-Slot-Single (Task-Descriptor, ANY-PROCESSOR, IDLE-WINDOW )
            if slot
                focused-processor = slot-¿processor
                Migrate-Task (Task-Descriptor, focused-processor)
                broadcast Bidding-Request (Task-Descriptor, focused-processor)
            else
                broadcast Bidding-Request (Task-Descriptor, HERE)
    end


procedure Receive-Bidding-Request (Task-Descriptor, Processor)
    begin
        slot = Find-Slot-Single (Task-Descriptor, HERE, IDLE-WINDOW)
        if guaranteed (Task-Descriptor, slot)
            Send-Bid (processor, Task-Descriptor, bid-value, HERE)
    end


procedure Receive-Task (Task-Descriptor)
    begin
        slot = Find-Slot-Single (Task-Descriptor, HERE, IDLE-WINDOW)
        if guaranteed (Task-Descriptor, slot)
            Insert-Task-Single (Task-Descriptor, slot→time, slot→processor)
            Notify (Insert-Task-Single (Task-Descriptor,
                slot→time, slot→processor), ORIGINAL-PROCESSOR);
            Ignore All Bids
        else
            Processor = best-bidder
            if Processor
                Migrate-Task (Task-Descriptor, Processor)
            else
                Cannot Schedule Task
    end
```

Figure 4: Dynamic Scheduling Using RAGA ADT

Appendix A
Summary of Real Time Logic*

Constants:
- Integer Constants
- Set of Action Constants
  - Action
    - A (name of a primitive or a composite action)
  - Subaction
    - $B.A$ (action A within composite action B)
    - $B.Ai$ (ith occurrence of action A within B)
- Set of Event Constants
  - Start Event
    - $\uparrow A$ where A is an action constant
  - Stop Event
    - $\downarrow A$ where A is an action constant
  - Transition Event
    - $(S := T), (S := F)$ where S is a State attribute
  - External Event
    - ΩE where E is the name of an external event

Variables:

range over integer, action or event constants denoted by names in lower case letters

Functions:
- Addition and Subtraction
- Multiplication by Constants
- Uninterpreted Functions (Range $\subseteq$ Integers)
- Occurrence Function $@(E, i)$

Predicates:
- Equality/Inequality Predicates $(=, <, \leq, >, \geq)$
- State Predicates denoting the truth of a state attribute during an interval

Formulas:

RTL formulas are constructed using the above predicates, universal and existence quantifiers, and first order logic connectives.

*Verbatim [3]

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] M. R. Garey and D. S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing*, 4.

[3] Farnam Jahanian and Aloysius Ka-Lau Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9):890–904, September 1986.

[4] Insup Lee and Susan B. Davidson. Adding time to synchronous process communication. *IEEE Transactions on Computers*, 36(8):941–48, August 1987.

[5] Dennis W. Leinbaugh and Mohamad-Reza Yamini. Guaranteed response times in a distributed hard real time environment. *IEEE Transactions on Software Engineering*, 12(12):1139–44, December 1986.

[6] C. L. Liu and James W. Layland. Scheduling algorithm for multiprogramming in a hard-real-time environment. *Journal of Association of Computing Machinery*, 20(1):46–61, January 1973.

[7] A. K. Mok. Fundamental design problems of distributed systems for the hard real time environment. PhD Thesis, M.I.T., 1983.

[8] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in distributed hard real-time systems. *IEEE Software*, 1(3), 1984.

[9] J. A. Stankovic S. Cheng and K. Ramamritham. Dynamic scheduling of groups of tasks with precedence constraints in distributed hard real-time systems. *IEEE Real Time Systems Symposium*.

[10] John A. Stankovic and Krithi Ramamritham. The design of the spring kernel. *Proceedings of the Real Time Systems Symposium*.

[11] Hideyuki Tokuda and Clifford W. Mercer. Arts: A distributed real-time kernel. *ACM Operating Systems Review*, 23(3):29–53, July 1989.

[12] Krithi Ramamritham Wei Zhao and John A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, 36(8):949–60, August 1987.