15th Triennial World Congress, Barcelona, Spain

SATISFYING COMPLEX TIMING CONSTRAINTS WITH PRE-RUN-TIME SCHEDULING

Jia Xu *,1

* *Department of Computer Science, York University*

Abstract: This paper illustrates, through examples, how a pre-run-time scheduling approach can be used to satisfy complex timing and other constraints in real-time embedded systems.

Keywords: Real-time embedded systems, scheduling, processes, off-line, timing analysis.

1. INTRODUCTION

Real-time embedded systems often have many different types of processes with complex timing and other constraints. Some of the processes may be periodic and some of them may be asynchronous. Some of the processes may have hard deadlines and some of them may have soft deadlines. For some of the processes, especially the hard real-time processes, complete knowledge about their characteristics can and must be acquired before run-time. For other processes, a prior knowledge of their worst case computation time and their data requirements may not be available. Some processes may have complex constraints and dependencies between them. For example, a process may need to input data that are produced by other processes. In this case, a process may not be able to start before those other processes are completed. Such constraints are called precedence relations. Exclusion relations may exist between processes when some processes must prevent simultaneous access to shared resources such as data and I/O devices by other processes. For some periodic processes, they may not be able to start immediately at the beginning of their periods. In this case, those processes have release time constraints.

This paper describes an approach based on pre-run-time scheduling for satisfying such complex timing and other constraints in real-time embedded systems. A formal specification of the algorithms used in the approach is provided in Xu and Lam (Xu and Lam, 1998). The description here is not intended to provide a formal specification of the algorithms, but to illustrate, through examples, how a pre-run-time scheduling approach can be used to effectively satisfy complex timing constraints in real-time embedded systems.

2. A PRE-RUN-TIME SCHEDULING APPROACH

The approach outlined below provides a system and methods for scheduling five types of real-time processes:
Set P-h-k: Periodic processes with hard deadlines and known characteristics.
Set A-h-k: Asynchronous processes with hard deadlines and known characteristics.
Set P-s-k: Periodic processes with soft deadlines and known characteristics.
Set A-s-k: Asynchronous processes with soft deadlines and known characteristics.
Set A-s-u: Asynchronous processes with soft deadlines and unknown characteristics.
For P-h-k and P-s-k processes, each such process p_i consists of one or more segments, with precedence relations defined on them to enforce the proper ordering of segments belonging to the same process.

[1] Current address: Department of Computer Science, York University, 4700 Keele Street, North York, Ontario M3J 1P3, Canada. This work was partially supported by a Research Grant from the Natural Sciences and Engineering Council of Canada.

It is assumed that the following are known for each such process before run-time: period prd_{p_i}; worst-case execution time c_{p_i}; release time r_{p_i}; deadline d_{p_i}; the set of data that the process reads and writes; any exclusion relationships with other process segments; any precedence relationships with other periodic process segments.
For A-h-k and A-s-k processes, each such process a_j consists of a single segment and the following are known for each such process before run-time: deadline d_{a_j}; worst-case execution time c_{a_j}; minimum time between two consecutive requests min_{a_j}; the set of data that the process reads and writes; any exclusion relationships with other process segments.
For A-s-u processes, each such process consists of a single segment and nothing else is known about each such process before run-time.

The system integrates pre-run-time scheduling with run-time scheduling to guarantee that the executions of the processes will satisfy all the specified constraints and dependencies. Whenever a new set of processes arrives in the system, the system schedules their executions in two phases: a pre-run-time (off-line) phase performed by a pre-run-time scheduler, and a run-time (on-line) phase performed by a run-time scheduler.

In each pre-run-time phase, the pre-run-time scheduler executes five steps.

Step 1. The pre-run-time scheduler divides asynchronous processes with hard deadlines and known characteristics, called A-h-k processes, into two subsets. One subset of asynchronous processes, called A-h-k-p, is converted into new periodic processes by the pre-run-time scheduler before run-time. When the pre-run-time scheduler converts an asynchronous process into a new periodic process, it prevents possible timing conflicts with other periodic and asynchronous processes, by reserving enough "room" in each new periodic process's deadline, to accommodate the computation times of all the periodic and asynchronous processes processes that have shorter deadlines that might preempt that new periodic process at run-time. The processes in the other subset of asynchronous processes, called A-h-k-a, remains asynchronous and are scheduled by the run-time scheduler at run-time. The pre-run-time scheduler reserves processor capacity for such process by adding the computation time of each A-h-k-a process to the computation time of every periodic process that has a hard deadline that is greater than that A-h-k-a process's deadline.

Whether each asynchronous process is converted into a new periodic process or not, is based on whether the processor capacity that needs to be reserved for the new periodic process exceeds the processor capacity that needs to be reserved for the asynchronous process if unconverted.

Example 1.

Suppose that there exists 4 asynchronous processes with hard deadlines and known characteristics (A-h-k processes), and 4 periodic processes with hard deadlines and known characteristics (P-h-k processes) as follows.
a_0: $c_{a_0} = 2, d_{a_0} = 2, min_{a_0} = 1,000$;
a_1: $c_{a_1} = 2, d_{a_1} = 7, min_{a_1} = 1,000$;
a_2: $c_{a_2} = 10, d_{a_2} = 239, min_{a_2} = 1,000$;
a_3: $c_{a_3} = 10, d_{a_3} = 113, min_{a_3} = 113$.
p_4: $r_{p_4} = 0, c_{p_4} = 26, d_{p_4} = 200, prd_{p_4} = 200$;
p_5: $r_{p_5} = 30, c_{p_5} = 16, d_{p_5} = 50, prd_{p_5} = 200$;
p_6: $r_{p_6} = 0, c_{p_6} = 26, d_{p_6} = 200, prd_{p_6} = 200$;
p_7: $r_{p_7} = 0, c_{p_7} = 16, d_{p_7} = 200, prd_{p_7} = 200$.
The adjusted computation times for p_4, p_5, p_6, p_7 will respectively be:
$c_{p_4}' = c_{p_4} + c_{a_0} + c_{a_1} = 26 + 2 + 2 = 30$;
$c_{p_5}' = c_{p_5} + c_{a_0} + c_{a_1} = 16 + 2 + 2 = 20$;
$c_{p_6}' = c_{p_6} + c_{a_0} + c_{a_1} = 26 + 2 + 2 = 30$;
$c_{p_7}' = c_{p_7} + c_{a_0} + c_{a_1} = 16 + 2 + 2 = 20$.
If a_3 is converted into a periodic process $newp_3$, d_{newp_3} = c_{a_3} + c_{a_0} + c_{a_1} = 10 + 2 + 2 = 14. Using Mok's formula (Mok, 1983), $prd_{newp_3} = (d_{a_3} - d_{newp_3} + 1)$ = 113 - 14 + 1 = 100.
If a_3 is converted into a periodic process $newp_3$, the processor capacity that needs to be reserved for $newp_3$ will be: $RPC_{newp_3} = c_{newp_3}'/prd_{newp_3} = 14/100 = 0.14$;
If a_3 is not converted, then the processor capacity that needs to be reserved for a_3 will be: $RPC_{a_3} = (\sum_{p_i \in P-h-k \wedge d_{a_j} \le d_{p_i}} ((\lceil d_{p_i} - r_{p_i})/mina_j) \rceil c_{a_j})/prd_{p_i})) + (c_{a_j}/min_{a_j}) = \lceil (d_{p_4} - r_{p_4})/min_{a_3} \rceil c_{a_3}/prd_{p_4} + \lceil (d_{p_6} - r_{p_6})/min_{a_3} \rceil c_{a_3}/prd_{p_6} + \lceil (d_{p_7} - r_{p_7})/min_{a_3} \rceil c_{a_3}/prd_{p_7} + c_{a_3}/min_{a_3}$ = 20/200 + 20/200 + 20/200 + 10/113 = 0.388 $\geq RPC_{newp_3} = 0.14$
a_3 will be converted into a new periodic process $newp_3$ = $(r_{newp_3}, c_{newp_3}, d_{newp_3}, prd_{newp_3})$ = (0, 10, 14, 100).
Using a similar process, it can be determined that the A-h-k processes a_0, a_1, a_2 should not be converted into periodic processes, because the processor capacity that needs to be reserved for the corresponding new processes exceeds the processor capacity that needs to be reserved when they remain asynchronous. □

Step 2. The pre-run-time scheduler determines the schedulability of the set of all periodic processes with hard deadlines and known characteristics, called P-h-k processes, which also includes the new periodic processes converted from A-h-k-p processes. The pre-run-time scheduler constructs a pre-run-time schedule in which one or more time slots are reserved for the execution of every P-h-k process, including every new P-h-k process converted from an A-h-k-p process. The time slots reserved for each P-h-k process also include time reserved for the executions of all A-h-k-a processes that have deadlines that are shorter than that P-h-k process' deadline, and which may preempt the execution of that P-h-k process. The present approach

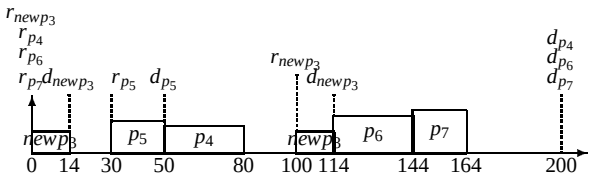

Fig. 1. A feasible schedule for all the P-h-k processes in Example 1, constructed in Step 2.

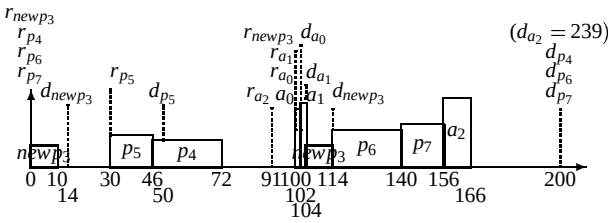

Fig. 2. A possible run-time execution of the A-h-k-a processes a_0, a_1, a_2, together with the P-h-k processes in Fig. 1.

is deliberately designed in a way that allows the pre-run-time scheduler to use practically almost any existing method that statically schedules a set of processes, including manual methods, to construct the pre-run-time schedule of periodic processes in Step 2 and in Step 4, without requiring any change in the methods used in any of the other steps of the present approach, so that the system and methods have the flexibility to incorporate and take advantage of any future new static scheduling method for satisfying any additional desired constraints among the most important and numerous type of processes in real-time applications, — the periodic processes.

Example 2.

If the algorithm described in (Xu and Parnas, 1990) is used to schedule all the P-h-k processes in Example 1 above, the pre-run-time schedule illustrated in Fig. 1 will be obtained. □

Step 3. The pre-run-time scheduler uses knowledge about the time slots reserved for the P-h-k processes in the pre-run-time schedule to determine, before run-time, the worst-case response times of all A-h-k-a processes. It is possible for the pre-run-time scheduler to use a simulation procedure which uses the A-h-k-a Scheduler and Main-Run-Time Scheduler to simulate the execution of all the processes (see description of the run-time scheduler for this approach) to determine the worst-case response time of each A-h-k-a process. In the simulation procedure, one execution for each arrival time between time 0 and LCM - 1 for each A-h-k-a process a_i is simulated, under the assumption that all other A-h-k-a processes whose deadlines are shorter or equal to a_i's deadline arrive at the same time as a_i at time t_s, and are put into execution before a_i; and the process that has the greatest computation time among all other A-h-k-a processes that have a greater deadline and can block a_i at that time will arrive one time unit before a_i's arrival time.

The pre-run-time scheduler verifies the schedulability of each A-h-k-a asynchronous process by checking whether its deadline is greater than or equal to its worst-case response time. Thus, the pre-run-time scheduler provides an a priori guarantee that all periodic and asynchronous processes with hard deadlines and known characteristics will always meet their deadlines.

Example 3.

In the examples, **e'(x)** denotes the actual time at which asyn. or periodic process *x*'s execution ends at run-time.

a_0's worst-case response time $R_{a_0} = R(a_0, t_s)$ will happen when a_0 arrives at time $t_s = 0$. Since no process excludes a_0, and a_0 has the shortest deadline among all processes, a_0 will always be put into execution immediately after it arrives, thus a_0's response time $R_{a_0} = R(a_0, 0) = \max\{R(a_0, t_s)\} = c_{a_0} = 2 \leq d_{a_0} = 2$.

a_1's worst-case response time $R_{a_1} = R(a_1, t_s)$ will happen when a_1 arrives at time $t_s = 0$. Since no process excludes a_1, and only one process a_0 has a shorter deadline compared with a_1's deadline, when a_1 arrives at time $t_s = 0$, assuming that a_0 will also arrive at time $t_s = 0$, a_1 will only be delayed by a_0's execution time, thus a_1's response time $R_{a_1} = \max\{R(a_1, t_s)\} = R(a_1, 0) = c_{a_0} + c_{a_1} = 2 + 2 = 4 \leq d_{a_1} = 7$.

a_2's worst-case response time R_{a_2} = The maximum value of $R(a_2, t_s)$ will happen when a_2 arrives at time $t_s = e(p_3) - c_{a_2} - c_{p_3} - c_{a_0} - c_{a_1} - 1 = 114 - 10 - 10 - 2 - 2 - 1 = 114 - 25 = 91$. (Fig. 2.) At time 91 a_2 will be delayed because the conditions of Case 1 of the A-h-k-a Scheduler will be true. Note that a_2 excludes $newp_3$, and $d_{newp_3} = 14 < d_{a_2} = 239$; if a_2 is allowed to execute at time 91, it will cause $newp_3$ to miss its deadline if a_0 and a_1 also preempt $newp_3$. As $d_{a_0} = 2 < d_{a_2} = 239$ and $d_{a_1} = 4 < d_{a_2} = 239$, the worst-case response-time of a_2 will happen when a_0 and a_1's arrival times actually occur at time $s(newp_3) = 100$. Thus a_2's response time $R_{a_2} = \max\{R(a_2, t_s)\} = R(a_2, 91) = e'(a_2) - r_{a_2} = 166 - 91 = 75 < d_{a_2} = 239$. Since the response time of every A-h-k-a process is less than or equal to its deadline, one would be able to guarantee that they are all schedulable. □

Step 4. The pre-run-time scheduler determines the schedulability of all the periodic processes with soft deadlines and known characteristics, called P-s-k processes, under the condition that all the P-h-k processes and A-h-k-a processes that were guaranteed to be schedulable in the previous steps are still schedulable. The pre-run-time scheduler re-constructs the pre-run-time schedule in which one or more time slots are reserved for the execution of every P-h-k process (including every new P-h-k process converted from an A-h-k-p process), and for every P-s-k process. The time slots reserved for each P-h-k or P-s-k process also include time reserved for the executions of all

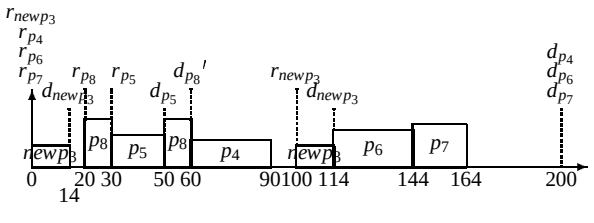

Fig. 3. A feasible schedule of the P-h-k and P-s-k processes in Example 4, constructed in Step 4.

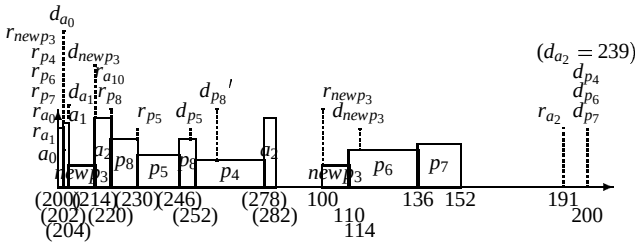

Fig. 4. A possible run-time execution of the A-h-k-a processes a_0, a_1, a_2, together with the P-h-k and P-s-k processes in Fig. 3.

A-h-k-a processes that have deadlines that are shorter than that P-h-k or P-s-k process' deadline, and which may preempt the execution of that P-h-k or P-s-k process. The pre-run-time scheduler uses the methods in the previous step that take into account knowledge about the time slots reserved for the P-h-k and P-s-k processes in the pre-run-time schedule to determine again, before run-time, the worst-case response times of every A-h-k-a process.

Example 4.

Suppose that in addition to the hard deadline processes in Examples 1-3 above, there exists the following periodic process with a soft deadline and known characteristics (P-s-k process).

p_8: $r_{p_8} = 20, c_{p_8} = 16, d_{p_8} = 60, prd_{p_8} = 200$;

Suppose further that p_8's deadline upperlimit is 100.

p_8's adjusted computation time is:

$c_{p_8}{}' = c_{p_8} + c_{a_0} + c_{a_1} = 16 + 2 + 2 = 20.$

The feasible pre-run-time schedule in Fig. 3 can be obtained using the algorithm described in (Xu and Parnas, 1990) in which each guaranteed periodic process reserves a time frame that includes reserved processor capacity for any A-h-k-a process that has a shorter deadline than that guaranteed periodic process's deadline.

The response times of the A-h-k-a processes in the examples above are re-calculated using the information in the schedule in Fig. 3 using a simulation method similar to that shown in Example 3 to verify that all A-h-k-a processes' worst-case response times are less than or equal to their deadlines. a_0's response time R_{a_0} and a_1's response time R_{a_1} will remain the same as in Example 3. $R(a_2, t_s)$ will happen when a_2 arrives at time $t_s = e(p_3) - c_{a_2} - c_{p_3} - c_{a_0} - c_{a_1} - 1 = 214 - 10 - 10 - 2 - 2 - 1 = 214 - 25 = 191$. a_2's response time $R_{a_2} = \max\{R(a_2, t_s)\} = R(a_2, 191) = e'(a_2) - r_{a_2}$ = 282 - 191 = 91 $< d_{a_2} = 239$. (See Fig. 4.) □

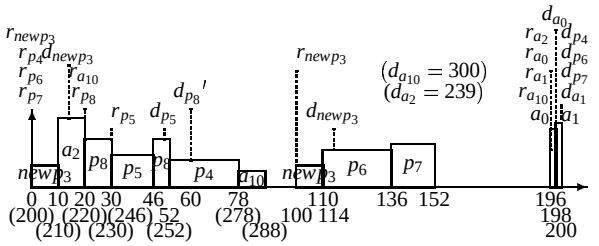

Fig. 5. A possible run-time execution of the A-s-k process a_{10}, together with the A-h-k-a processes a_0, a_1, a_2, and the P-h-k and P-s-k processes in Fig. 4.

Step 5. The pre-run-time scheduler uses knowledge about the time slots reserved for the P-h-k and P-s-k processes in the pre-run-time schedule to determine, before run-time, the worst-case response times of asynchronous processes with soft deadlines and known characteristics, i.e., A-s-k processes.

Example 5.

Suppose there exists the following asynchronous process with a soft deadline and known characteristics (A-s-k process).

a_{10}: $c_{a_{10}} = 10, d_{a_{10}} = 300, min_{a_{10}} = 300.$

By using a simulation method similar to that used in Example 3, one can determine that the maximum value of $R(a_{10}, t_s)$ will happen when a_{10} arrives at time $t_s = 196$. (See Fig. 5 for an illustration of this case.) a_{10}'s worst-case response time will be $R_{a_{10}}$ = $\max\{R(a_{10}, t_s)\}$ = $R(a_{10}, 196) = e'(a_{10}) - r_{a_{10}}$ = 288 - 196 = 92 $< d_{a_{10}}$ = 300. □

At the end of the pre-run-time phase, a feasible pre-run-time schedule for all the periodic processes with known chracteristics will be constructed, while the worst-case response times of all the asynchronous processes with known characteristics will be determined.

Run-time phase. A run-time scheduler uses knowledge about the time slots reserved for the periodic processes in the pre-run-time schedule to schedule the executions of all the periodic and asynchronous processes, that is, the P-h-k processes (including every new P-h-k process converted from an A-h-k-p process), P-s-k processes, A-h-k-a processes, A-s-k processes, as well as asynchronous processes with soft deadlines and unknown characteristics, called A-s-u processes, in a way that guarantees that every periodic process's execution will complete before the end of that periodic process's time slot in the pre-run-time schedule , and all the asynchronous processes with soft deadlines and known characteristics, are guaranteed to be completed within the worst-case response time pre-determined in Step 4 and Step 5 after their arrival, so that all the constraints and dependencies of all processes with known characteristics will always be satisfied.

The A-h-k-a Scheduler Subroutine operates as follows:

At any time t:
if some A-h-k-a process a_i has arrived at time t,
or if some process x_i completes its computation at time t
or if t is both the release time and start time in the pre-run-time schedule for some P-h-k or P-s-k process p, i.e., $t = r_p = s(p)$
then
begin

for each A-h-k-a process a_i that has already arrived and not yet completed, i.e., $\neg(e'(a_i) \leq t)$, if a_i satisfies any of the following conditions, then Delay a_i:

(1) Delay a_i either if the immediate execution of a_i may cause the execution of a P-h-k or P-s-k process p with a shorter or equal deadline to exceed the end of its time slot in the pre-run-time schedule; or, if it may cause some A-h-k-a process a_j to be blocked for the duration of two processes a_i and p which both have greater deadlines compared with a_j's deadline.
(2) Delay a_i if there exists any process x that has started but not completed and excludes a_i.
(3) Delay a_i if there exists any process x that has started but not completed and has a deadline that is shorter than or equal to a_i's deadline.
(4) Delay a_i if there exists any A-h-k-a process a_j that has started but not completed and which excludes a P-h-k or P-s-k process p with a shorter or equal deadline compared with a_i's deadline.
(5) Delay a_i if the immediate execution of a_i may cause the start time of the execution of a P-h-k or P-s-k process p with a shorter or equal deadline to be delayed beyond the beginning of its time slot in the pre-run-time schedule, when p may be preempted by some other periodic process p_1, and a_i cannot be preempted by p_1.
(6) Delay a_i if there exists any process x that has started but not completed and which excludes some other A-h-k-a process a_j which has a deadline that is shorter than both x and a_i's deadline, because that may cause a_j to be blocked by the duration of more than one process with greater deadlines.
(7) Delay a_i if there exists a P-h-k or P-s-k process p that has become ready, and which has a deadline that is shorter than or equal to a_i's deadline, when a_i does not exclude p and does not exclude any A-h-k-a process with deadline that is shorter than p's deadline.

Select, among all processes $a_i \in$ A-h-k-a, such that a_i has already arrived and not yet completed, and a_i is NOT Delayed, the process which has the shortest deadline. If more than one process is thus selected, select among them the process that has the smallest index. end;

return to Main Run-Time Scheduler;

The Main-Run-Time Scheduler operates as follows:

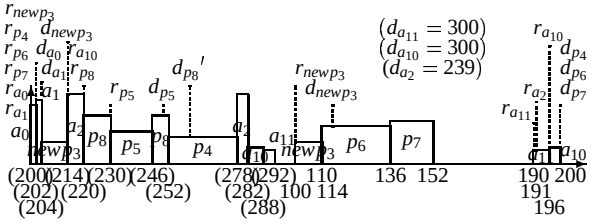

Fig. 6. A possible run-time execution of the A-s-u process a_{11}, and the A-s-k process a_{10}, scheduled by the Main Run-Time Scheduler together with the A-h-k-a processes a_0, a_1, a_2, and the P-h-k and P-s-k processes in Fig. 5.

At any time t:
if some process x has arrived at time t, or has completed at time t,
or if t is both the release time and start time in the pre-run-time schedule for some P-h-k or P-s-k process p
then execute the A-h-k-a-Scheduler-Subroutine;
if some A-h-k-a process a_i is selected for execution at time t by the A-h-k-a Scheduler
then execute a_i
else
if at least one P-h-k or P-s-k periodic process is ready, then execute one P-h-k or P-s-k periodic process that is ready, according to the order in which their time slots appear in the pre-run-time schedule.
If no P-h-k or P-s-k process is ready, and at least one A-s-k process is ready, then execute the A-s-k process which has the shortest deadline among all A-s-k processes that are ready.
If no A-s-k process is ready, and at least one A-s-u process is ready, then execute an A-s-u process which has the shortest deadline among all A-s-u processes that are ready.

Example 6.

Continuing with the set of processes in Examples 1-5 above, suppose there exists the following asynchronous process a_{11} with a soft deadline and unknown characteristics (A-s-u process). (a_{11}'s characteristics are only known after its arrival.)
a_{11}: $c_{a_{11}} = 10, d_{a_{11}} = 300$.
Suppose that A-s-u process a_{11} makes a request at time 190; A-h-k-a process a_2 makes a request at time 191; A-s-k process a_{10} makes a request at time 196; and A-h-k-a processes a_0 and a_1 make requests at time 200. The A-h-k-a Scheduler Subroutine and the Main-Run-Time Scheduler will schedule all these processes and all the processes in the previous examples as illustrated in Fig. 6. □

We note that existing algorithms or protocols that perform all scheduling activities at run-time cannot handle precedence constraints, release time and exclusion constraints simultaneously in an efficient way, hence are not capable of guaranteeing the schedulability of the set of processes given in these examples.

3. CONCLUSION

Compared with previous systems and methods that perform all scheduling activities at run-time, using a pre-run-time scheduling approach is better suited for satisfying complex timing constraints for the following reasons:

(1) In most real-time applications the bulk of the computation is usually performed by periodic processes for which the characteristics are known a priori. Complex constraints and dependencies are normally defined on the periodic processes. When a pre-run-time scheduling approach is used, all the periodic processes are scheduled before run-time, there is practically no limit on the time that can be used for scheduling the periodic processes. This allows one to use better methods that can handle a great variety of application constraints and dependencies, and that can achieve higher schedulability for the most important and most numerous type of processes in real-time applications.

(2) With a pre-run-time approach, the run-time overhead required for scheduling and context switching is much smaller.

(2.1) The number of asynchronous processes that the run-time scheduler needs to schedule, should be small, as in most real-time applications, the number of asynchronous processes with hard deadlines is usually small (Xu and Parnas, 1993).

(2.2) A significant portion of the asynchronous processes will be transformed into periodic processes when using this approach. For those asynchronous processes that are not transformed into periodic processes, their interarrival times are likely to be long.

(2.3) Most of the important scheduling decisions have already been determined before run-time. In particular, the relative ordering of all the periodic processes was determined before run-time when the pre-run-time schedule was computed.

(2.4) A significant portion of the parameters used by the run-time scheduler to make scheduling decisions for asynchronous processes, are known before run-time, so one can pre-compute major portions of the conditions that are used for decision making, and the amount of computation that needs to be performed for scheduling asynchronous processes at run-time can be minimized.

(2.5) From the pre-run-time schedule, one would know in advance exactly which periodic process may preempt which other periodic process at run-time. Thus one can use this information to minimize the amount of context switching.

(3) When a pre-run-time scheduling approach is used, once the pre-run-time schedule has been determined for all the periodic processes, the run-time scheduler can use this knowledge to achieve higher schedulability for the small number of asynchronous processes that needs to be scheduled at run-time.

(3.1) The run-time scheduler can use knowledge about the pre-run-time schedule to schedule asynchronous processes more efficiently, e.g., it would be possible to completely avoid blocking of a periodic process with a shorter deadline by an asynchronous process with a longer deadline.

(3.2) When determining the worst-case response times of asynchronous processes, one does not need to make overly pessimistic assumptions, e.g., one does not need to assume that for each process, all the periodic processes with shorter deadlines can arrive at the same time to delay that process. Thus one can obtain tighter worst-case response times for asynchronous processes.

(4) When a pre-run-time scheduling approach is used, verifying that all timing constraints will always be satisfied is much easier.

(4.1) It is straightforward to verify that all the timing constraints and dependencies between the periodic processes are satisfied in a pre-run-time schedule.

(4.2) When using the technique of pre-run-time scheduling, timing constraints and dependencies are directly "embedded" in the pre-run-time schedule, thus for the majority of the processes, one can avoid the use of complicated run-time synchronization mechanisms for which it is often extremely difficult to obtain reasonable and accurate execution time bounds.

(4.3) The number of asynchronous processes is reduced, and the ordering of the periodic processes is fixed in the pre-run-time schedule. This significantly reduces the complexity of verifying that the asynchronous processes will meet timing constraints. (Xu and Parnas, 2000)

4. REFERENCES

Mok, A.K. (1983). Fundamental design problems of distributed systems for the hard-real-time environment. *Ph.D Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology*.

Xu, J. and D.L. Parnas (1990). Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. on Software Engineering* pp. 360–369.

Xu, J. and D.L. Parnas (1993). On satisfying timing constraints in hard-real-time systems. *IEEE Trans. on Software Engineering* pp. 1–17.

Xu, J. and D.L. Parnas (2000). Priority scheduling versus pre-run-time scheduling. *Real-Time Systems* pp. 7–23.

Xu, J. and K.L. Lam (1998). Integrating run-time scheduling and pre-run-time scheduling of real-time processes. *Proc. 23rd IFAC/IFIP Workshop on Real-Time Programming*.