Correcting the EDF protocol in Ada 2005

A. Zerzelidis, A. Burns and A.J. Wellings
Real-Time Systems Research Group
Department of Computer Science
University of York, UK

Abstract

Earliest Deadline First (EDF) dispatching has been introduced into the Ada 2005 definition. Included in this definition is support for Baker's protocol for preemption level control over access to protected objects. Unfortunately the current model fails to implement all the situations covered by Baker's approach. A counter example is provided that illustrates this deficiency with the language as currently defined. A minor change to the language definition is proposed that removes the flaw.

1. Introduction

One of the major enhancements contained within the Ada 2005 definition, in terms of the language's support for real-time applications, is the introduction of Earliest Deadline First (EDF) dispatching. Ada is the first mainstream engineering language to support this scheduling algorithm. EDF has been proven to be the most effective such algorithm available, in the sense that if a set of tasks is schedulable on a single processor by any dispatching policy then it will also be schedulable by EDF. Support for EDF requires two language features:

- representation of deadline for a task,
- representation of preemption level for a protected object.

The first is obviously required; the second is the EDF equivalent of priority ceilings and allows protected objects to be 'shared' by multiple tasks [1].

A deadline is usually defined to be *absolute* if it refers to a specific (future) point in time; for example there is an absolute deadline on the completion of this paper by 12th January 2007. A *relative* deadline is one that is anchored to the current time: "We have one month to finish this paper". Obviously, if the current time is known then a relative deadline can be translated into an absolute one (and vice versa). Repetitive (periodic) tasks often have a static relative deadline (for example 10ms after release) but will have a different absolute deadline every time they are released. The EDF scheme is more accurately expressed as *earliest absolute deadline first*.

The need to support preemption levels comes from the application of Bakers's protocol for controlling access to shared entities (see following discussion). Unfortunately, the language's support for this protocol is not complete. There are situations in which the behaviour of an Ada program will not follow the protocol and will therefore perform sub-optimally.

In this paper we illustrate this problem with the language as currently defined, and show how the problem can be rectified with a simple change to the wording of one paragraph in the Reference Manual. First however, Baker's protocol is defined and in Section 3 the EDF language features of Ada 2005 are described.

2. Baker's Preemption Level Protocol for Protected Objects

One of the major features that Ada 95 provided is support for monitors via the introduction of the protected object. Although the rendezvous is a powerful synchronization and communication primitive, it does not easily lead to tight scheduling analysis. Rather a more asynchronous scheme is desirable for real-time applications. The protected object provides this form of communication. With this construct, and the use of fixed priorities for tasks and a priority ceiling protocol for protected objects, it is possible to predict worst-case completion times for tasks.

With standard fixed priority scheduling, *priority* is actually used for two distinct purposes:

- to control dispatching, and
- to facilitate an efficient and safe way of sharing protected data.

The latter is often known as the priority ceiling protocol (in Ada it is called the *ceiling locking policy*). In Baker's stack-based protocol, two distinct notions are introduced for these policies[1]:

- earliest deadline first to control dispatching[1],
- preemption levels to control the sharing of protected data.

With preemption levels (which is a very similar notion to priority), each task is assigned a static preemption level, and each protected object (shared unit) is assigned a ceiling value that is the maximum of the preemption levels of the tasks that call it. At run-time, a newly released task, T1 say, can preempt the currently running task, T2, if and only if:

- the absolute deadline of T1 is earlier (i.e. sooner) than the absolute deadline of T2, and
- the preemption level of T1 is higher than the ceiling preemption level of every locked protected object.

With this protocol it is possible to show that, on a single processor, mutual exclusion (over the protected object) is ensured by the protocol itself (in a similar way to that delivered by fixed priority scheduling and ceiling priorities)[2]. Baker also showed, for the classic problem of scheduling a fixed set of periodic or sporadic tasks, that if preemption levels are assigned according to each task's relative deadline then a task can suffer at most a single block from any task with a longer deadline. Again this result is identical to that obtained for fixed priority scheduling. **Note preemption levels must be assigned inversely to relative deadline (the smaller the relative deadline, the higher the preemption level).**

3. Supporting EDF Scheduling

It is an anomaly that Ada 95's support for real-time does not extend as far as having a direct representation for 'deadline', but Ada 2005 has rectified this by providing the following package:

```
with Ada.Real_Time; with Ada.Task_Identification;
package Ada.Dispatching.EDF is
  subtype Deadline is Ada.Real_Time.Time;
  Default_Deadline : constant Deadline :=
       Ada.Real_Time.Time_Last;
  procedure Set_Deadline(D : in Deadline;
       T : in Ada.Task_Identification.Task_ID :=
       Ada.Task_Identification.Current_Task);
  procedure Delay_Until_And_Set_Deadline(
       Delay_Until_Time : in Ada.Real_Time.Time;
       Deadline_Offset : in Ada.Real_Time.Time_Span);
  function Get_Deadline(
       T : Ada.Task_Identification.Task_ID :=
       Ada.Task_Identification.Current_Task)
       return Deadline;
end Ada.Dispatching.EDF;
```

The meaning of most of the operations of this package should be clear. A call of `Delay_Until_And_Set_Deadline` delays the calling task until time `Delay_Until_Time`. When the task becomes runnable again it will have deadline `Delay_Until_Time + Deadline_Offset`. The inclusion of this procedure reflects a common task structure for periodic activity. Note all tasks are given a deadline - if one has not been explicitly set then `Time_Last` (the far distant future) is given as the default.

A pragma is also provided to give an initial relative deadline to a task to control dispatching during activation.

To keep language changes to a minimum, Ada 2005 does not attempt to define a new locking policy but uses the existing ceiling locking rules without change. Priority, as currently defined, is used to represent preemption levels. EDF scheduling is defined by a new dispatching policy.

```
pragma Task_Dispatching_Policy
                        (EDF_Across_Priorities);
```

In Ada 95, dispatching is defined by a model that has a number of ready queues, one per priority level. Each queue, for the standard model, is ordered in a FIFO manner. Tasks have a base priority (assigned by pragma `Priority` and changed, if desired, by the use of `Dynamic_Priority.Set_Priority`). They may also have a higher active priority, if they inherit such a value during, for example, a rendezvous or execution within a protected object. Preemptive behavior is enforced by requiring a context switch from the current running task, if there is a higher priority non-empty ready queue. This is known in Ada as a *Dispatching Point*. At any dispatching point, the current running task is returned to its ready queue and another task (or indeed the same task if appropriate) is taken from its ready queue and executed. It should be noted that this is an abstract model of the required behavior; an implementation does not need to deliver the required semantics in this way. This also applies to the new model defined below.

3.1. The Rules of EDF Dispatching

The basic preemption rule given earlier for Baker's protocol, whilst defining the fundamental behavior, does not give a complete model. For example, it is necessary to define what happens to a newly released task that is not entitled to preempt. A complete model, within the context of Ada's ready queues, is defined by the following rules from

[1] His paper actually proposes a more general model of which EDF dispatching is an example.

[2] This property requires that there are no suspensions inside the protected object – as is the case with Ada's protected objects.

the Ada 2005 reference manual (see paragraphs 17/2 to 28/2 of section D.2.6 of the manual).

When EDF_Across_Priorities is specified for priority range Low..High all ready queues in this range are ordered by deadline. The task at the head of a queue is the one with the earliest deadline.

A task dispatching point occurs for the currently running task T to which policy EDF_Across_Priorities applies:

- *when a change to the deadline of T occurs;*
- *there is a task on the ready queue for the active priority of T with a deadline earlier than the deadline of T; or*
- *there is a non-empty ready queue for that processor with a higher priority than the active priority of the running task.*

In these cases, the currently running task is said to be preempted and is returned to the ready queue for its active priority.

For a task T to which policy EDF_Across_Priorities applies, the base priority is not a source of priority inheritance; the active priority when first activated or while it is blocked is defined as the maximum of the following:

- *the lowest priority in the range specified as EDF_Across_Priorities that includes the base priority of T;*
- *the priorities, if any, currently inherited by T;*
- ***the highest priority P, if any, less than the base priority of T such that one or more tasks are executing within a protected object with ceiling priority P and task T has an earlier deadline than all such tasks.***

When a task T is first activated or becomes unblocked, it is added to the ready queue corresponding to this active priority. Until it becomes blocked again, the active priority of T remains no less than this value; it will exceed this value only while it is inheriting a higher priority.

When the setting of the base priority of a ready task takes effect and the new priority is in a range specified as EDF_Across_Priorities, the task is added to the ready queue corresponding to its new active priority, as determined above.

In addition, there is a rule (paragraph 30/2 of section D.2.6) that no protected object can have a ceiling of value *Low* – when EDF_Across_Priorities is specified for priority range *Low..High*.

Rule 26/2 (the one in bold in the above quote) contains the key semantics, and is intended to ensure the behaviour required by Baker's Algorithm. Whilst in most cases this is true there is a circumstance in which tasks can execute in the wrong order. This is illustrated by Scenario 2 in the following example.

Example behaviour

Consider the following scenarios. Remember in all of these descriptions, the running task is always returned to its ready queue whenever a task arrives. A task (possibly the same task) is then chosen to become the running task following the rules defined above.

The system contains four tasks: T1, T2, T3 and T4 and three resources that are implemented as protected objects: R1, R2 and R3. Table 1 defines the parameters of these entities (in this table, D is relative deadline, L is preemption level, U is the resources used, t is the arrival time and A the absolute deadline.

Task	D	L	U	t	A
T1	100	1	R1,R3	0	100
T2	80	2	R2,R3	2	82
T3	60	3	R2	4	64
T4	58	4	R1	8	66

Table 1. A Task Set

We are concerned with just a single invocation of each task. The arrival times have been chosen so that the tasks arrive in order of lowest preemption level task first etc. We assume all computation times are sufficient to cause the executions to overlap.

The resources are all used by more than one task, but only one at a time and hence the ceiling values of the resources are straightforward to calculate. For R1, it is used by T1 and T4; hence the ceiling preemption level is 4. For R2, it is used by T2 and T3; hence the ceiling value is 3. Finally, for R3, it is used by T1 and T2; the ceiling equals 2 (see Table 2).

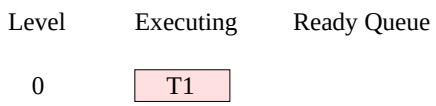
Protected Object	Ceiling Value
R1	4
R2	3
R3	2

Table 2. Ceiling Values

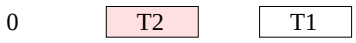
To implement this set of tasks and resources will require ready queues at level 0 (value of *Low* in this example) and values up to 4. Scenario 1 runs through a possible behaviour of the tasks set and illustrates the correct behaviour of the Ada 2005 protocol.

Scenario 1

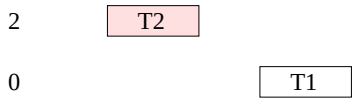
At time 0, T1 arrives. All ready queues are empty and all resources are free so T1 is placed in queue 0. It becomes the running task. This is illustrated in the following where 'Level' is the priority level, 'executing' is the name of the task that is currently executing, and 'Ready Queue' show the other non-blocked tasks in the system.



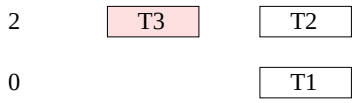
At time 2, T2 arrives and is added to ready queue 0 in front of T1 as it has a shorter absolute deadline. Now T2 is chosen for execution.



Assume at time 3, T2 calls R3. Its active priority will rise to 2.



At time 4, T3 arrives. Task T2 is joined by T3 on queue 2, as T3 has an earlier deadline and a higher preemption level; T3 is at the head of this queue and becomes the running task.

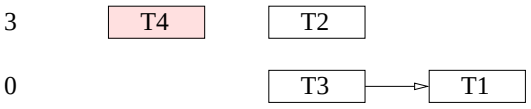


At time 8, T4 arrives. Tasks T3 and T2 are now joined by T4 as it has a deadline earlier than T2 and a higher preemption level (than 2). Task T3 remains the running task, and will execute until it completes.

Scenario 2 - Incorrect Behaviour

Here we make the simple change that, at time 3, T2 calls R2 instead of R3. Its active priority will rise to 3. Now when T3 arrives at time 4, it will not have a high enough preemption level to join ready queue 3 and will be placed on the lowest queue at level 0 (but ahead of T1). Task T2 continues to execute.

At time 8, T4 arrives. It passes both elements of the test and is placed on the queue at level 3 ahead of T2 and therefore preempts it.



The result of this last modification is that T4 is executing even though T3 is in the system and has an earlier deadline than T4. This is not in accordance with Baker's protocol. To force compliance with the protocol the following change has to be made.

4. Correcting the Definition

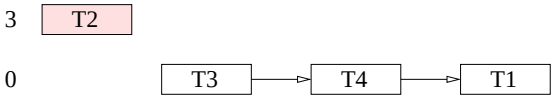
The problem with the current language definition is that Rule 26/2 is not strong enough: task T must have an earlier deadline than all tasks not only at level P but at all lower levels. So the final part of the rule to determine the active priority of a task (i.e. 26/2):

- *26/2 the highest priority P, if any, less than the base priority of T such that one or more tasks are executing within a protected object with ceiling priority P and task T has an earlier deadline than all such tasks.*

must become:

- *26/2* the highest priority P, if any, less than the base priority of T such that one or more tasks are executing within a protected object with ceiling priority P and task T has an earlier deadline than all such tasks and all other tasks on ready queues with priorities strictly less than P.*

Now Scenario 2 described above will behave as follows: at time 8, T4 will 'fail' 26/2* and will be placed on the level 0 queue between T3 and T1.



T2 will continue until it completes its execution in the protected object. Its priority will then fall to 0 and T3 will preempt it. All tasks now execute in deadline order – earliest first.

A formal proof of the revised protocol is not possible within the space allocated to this paper, however a key property can be explored via the following considerations. Strict EDF dispatching is ensured as long as a task T1 with an absolute deadline d1 is never on a ready queue below that of task T2 with absolute deadline d2 when d1<d2 unless T2 is using a shared resource and as a result has an inherited preemption level. This follows from the fact that ready queues are dispatched in priority order. There are two cases to consider:

Task T1 arrived before T2: T2 cannot be placed above T1 (in the ready queue order) if it arrived later and d1 is before d2 – direct consequence of 26/2*.

Task T1 arrived after T2: note, T2 has no inherited preemption level. As T1 has an earlier deadline than T2 it would be placed on the same queue as T2 (or higher) unless it has a lower preemption level. If it have a lower preemption level then we have the properties: T1 arrives after T2, has an earlier deadline than T2, but a lower preemption level. This is in direct contradiction to the preemption level allocation algorithm required by Baker (see discussion at the end of Section 2). Hence T1 cannot have a lower preemption level and would thus be placed on an equal or higher ready queue to that of T2.

5. Conclusion

The inclusion of EDF dispatching into the Ada language is a major enhancement. Ada becomes the first main stream engineering language to support this common protocol for controlling the execution of real-time activities. Operating systems, as well as languages, at best support only fixed priority dispatching. But for many resource restricted applications, the extra performance one can get from EDF scheduling is a major attraction. Ada's move to support EDF and fixed priority dispatching, and their integrated use within the same program will therefore open up the use of Ada in a number of new fields including mobile computing.

Although the definition contained within the Ada 2005 definition attempted to implement Baker's preemption-level protocol within the existing rules for priority ceiling locking for protected objects, the current model is not entirely satisfactory. The problem has been identified in this paper. Fortunately a minor rewording of one paragraph is all that is needed to remove the flaw and complete the protocol's definition.

Acknowledgements

The authors would like to thank Ted Baker for useful discussions on his protocol. Also John Barnes for comments on an earlier draft of the paper. The work reported in this paper is funded, in part, by the EU project ARTIST.

References

[1] T.P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1), March 1991.