Combining EDF and FP Scheduling: Analysis and Implementation in Ada 2005

A. Burns, A.J. Wellings and F. Zhang

Real-Time Systems Group
Department of Computer Science
University of York, UK.

Abstract. Earliest Deadline First (EDF) and Fixed Priority (FP) scheduling represent the two main dispatching policies within the research domain of real-time systems engineering. Both dispatching policies are now supported by Ada. In this paper the two approaches are combined to maximize the advantages of both schemes. From EDF comes efficiency, from FP predictability. A system model is presented in which a relatively small number of high-integrity tasks are scheduled by FP, with the rest of the tasks being handled via an EDF domain of lower priority. Two aspects of integration are covered in this paper. Firstly, Response-Time Analysis (for FP) and Processor-Demand Analysis (for EDF) are brought together to provide a single analysis framework. Secondly, the programming of systems which combine FP and EDF is addressed within the facilities provided by Ada 2005. Both partitioned and dynamic schemes are covered.

1 Introduction and Related Work

It is remarkable that the two most common scheduling schemes for real-time systems, EDF (Earliest Deadline First) and FP (Fixed Priority – sometimes known as rate monotonic) were both initially defined and analyzed in the same paper [19]. Since the publication of this seminal work in 1973, there has been a vast amount of research material produced on both of these schemes, and on the many ways of comparing and contrasting them. In this paper we will not rehearse the 'which is better' debate but will motivate the use of the combination of both approaches. We will show how combined EDF+FP dispatching can be analyzed and how systems can be programmed in Ada 2005 that make use of separated and combined EDF and FP domains. Indeed we will show how tasks can migrate from EDF to FP dispatching during execution.

In general, EDF has the advantage of optimality – its makes the best use of the available processor, whilst FP has the advantage of predicability and efficiency of implementation over current real-time operating systems. For general task parameters, FP has (up to recently) also have the advantage that schedulability is easier to evaluate (but see new results reviewed in Section 3).

The motivation for combining the two schemes comes from the wish to exploit the benefits of both approaches: EDF for its effectiveness and FP for the predictability it affords to high priority tasks [12]. In keeping with a number of recent papers on hierarchical approaches to scheduling, we shall explore the properties of a system that has FP as its basic dispatching mechanism. So high integrity tasks will run under FP, but the majority of tasks within the system will be managed by EDF (at a conceptually low priority). As a result of this scheme, FP tasks are isolated from the EDF ones, any overrun of the EDF tasks will not effect the FP ones, and the FP tasks will have regular execution patterns which helps to reduce input and output jitter.

The desire to combine EDF and FP is not new [26, 15, 6]. In this paper we combine RTA (Response-Time Analysis) for FP and PDA (Processor-Demand Analysis) for EDF using an adaption of the hierarchical scheduling approach presented by Zhang and Burns in 2007[23]. We are able to present necessary and sufficient analysis of combined EDF and FP systems.

Although EDF does have a number of advantages over FP, its use by industry has been limited. Priority based systems are supported by a wide range of real-time kernel APIs (such as POSIX) and programming languages such as Real-Time Specification for Java (RTSJ) and Ada. EDF support is found only on academic research platforms. Recently Ada [9] has been extended to include a number of features of relevance to the programming of real-time systems. One of these is the support for EDF dispatching and combined EDF and FP dispatching – including the sharing of protected objects between tasks scheduled by the different schemes.

The structure of this paper is straightforward, a system model is presented in Section 2, existing analysis for FP and EDF systems is covered in Section 3 with Section 4 containing the new integrated analysis. The programming of integrated FP and EDF systems is addressed in Section 5. Sections 3 to 5 address statically partitions systems. Although this may be appropriate for application where all tasks must be guaranteed, many applications will contain soft tasks. These require a more dynamic approach that is able to guarantee hard tasks and effectively schedule soft tasks. One such scheme is illustrated in Section 6. Conclusions are drawn together in Section 7.

2 System Model

We use a standard system model in this paper, incorporating the preemptive scheduling of periodic and sporadic task systems. A real-time system, $\mathcal{A}$, is assumed to consist of N tasks (τ_1 .. τ_N) each of which gives rise to a series of jobs that are to be executed on a single processor. Each task τ_i is characterized by several parameters:

- A *period* or *minimum inter-arrival time* T_i; for *periodic* tasks, this defines the exact temporal separation between successive job arrivals, while for *sporadic* tasks this defines the minimum temporal separation between successive job arrivals.
- A *worst-case execution time* C_i, representing the maximum amount of time for which each job generated by τ_i may need to execute. The worst-case utilization (U_i) of τ_i is C_i/T_i.
- A *relative deadline* parameter D_i, with the interpretation that each job of τ_i must complete its execution within D_i time units of its arrival. In this paper we assume $D_i \leq T_i$. The *absolute deadline* of a job from τ_i that arrives at time t is $t + D_i$.

Once released, a job does not suspend itself. We also assume in the analysis, for ease of presentation, that tasks are independent of each other and hence there is no blocking factor to be incorporated into the scheduling analysis. The use of protected objects to allow data sharing is however allowed in the program model.

System overheads are ignored in this treatment. Their inclusion would not impact on the structure of the results presented, but would complicate the presentation of these results. In practice, these overheads must of course not be ignored [11].

There are no restrictions on the relative release times of tasks (other than the minimum separation of jobs from the same task). Hence we assume all tasks start at the same instant in time – such a time-instant is called a *critical instant* for the task system[19]. In this analysis we assume tasks do not experience release jitter.

In the initial specification of the model, the system $\mathcal{A}$ is statically split into two sets: $\mathcal{A}^{FP}$ and $\mathcal{A}^{EDF}$. All task in $\mathcal{A}^{FP}$ are scheduled using fixed distinct priorities with the priorities being assign by some appropriate scheme (for the application) such as Deadline Monotonic[18]. Tasks within $\mathcal{A}^{EDF}$ are scheduled by EDF. The total utilization of the tasks in these two sets is denoted by U_{FP} and U_{EDF}. For any system to be feasible: $U_{FP} + U_{EDF} \leq 1$.

At any time during execution, if a task within $\mathcal{A}^{FP}$ has an active job (released but not yet competed) then it, or another member of $\mathcal{A}^{FP}$ will execute. The order of execution of the tasks from $\mathcal{A}^{FP}$ is determined by the static priority parameter of the task. If there are no active jobs from $\mathcal{A}^{FP}$ then an active job from $\mathcal{A}^{EDF}$ is chosen for execution. The job with the earliest (soonest) absolute deadline is picked for execution. At all times, the arrival of a high priority job will preempt a lower priority or EDF job. Similarly the arrival of a job with an earlier absolute deadline will preempt the current EDF job. As a consequence of these rules, tasks in $\mathcal{A}^{FP}$ are unaffected by the existence of EDF tasks (other than by the use of shared protected objects). The EDF tasks, however, suffer interference from the FP tasks.

In order to illustrate the application of the methods described in this paper, the example provided in Table 1 will be used. It consists of ten tasks, three of which are high-integrity I/O routines that require their deadlines to be met on all occasions and additionally require minimum jitter over their input and output operation. The other seven tasks are less critical and may be subject to occasional overruns in their execution times. Nevertheless they require their deadlines to be met if there are no execution time faults.
Note the total utilization of this task set is high at 0.97 and that a number of tasks have deadline less than period.

3 Existing Schedulability Analysis

We are concerned with analysis that is necessary, sufficient and sustainable [3]. In general, schedulability is asserted by either a direct test for this property or via the intermediate calculation of response times. Response times are then trivially compared with (relative) deadlines to complete the analysis. Direct and response-time analysis exists for both FP and EDF. However, Response-Time Analysis (RTA) is the more common approach for FP systems, and the values of the response times will be needed in the integrated scheme. Hence RTA will be used for fixed priority analysis. For EDF the re-

Task ID	T	C	D
τ_1	10	1	4
τ_2	50	2	50
τ_3	65	1	30
τ_4	10	2	8
τ_5	20	1	20
τ_6	30	5	20
τ_7	50	4	50
τ_8	100	13	100
τ_9	200	26	150
τ_{10}	1500	80	900

Table 1. Example Task Set

verse is true, with a direct test known as Processor-Demand Analysis (PDA) being the most appropriate to apply.

3.1 RTA for FP Systems

Response-Time Analysis [16, 1] examines each task in turn, and computes it latest completion time (response-time) which is denoted by R_i. An equation for R_i is obtained by noting that if task τ_i is released at time t and this is a critical instant, then in the interval (t,R_i] there must be time for τ_i to complete and every higher priority job that is released in the interval to also complete. Hence,

$$R_i = C_i + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{1}$$

For all i, $\mathbf{hp}(i)$ denotes the set of tasks with higher priority than τ_i. This equation for R_i is solved by forming a recurrence relation:

$$w_i^{n+1} = C_i + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j \tag{2}$$

As long as the start value (w_i^0) is less than R_i then this recurrence relation will find the worst case response time ($w_i^{n+1} = w_i^n = R_i$) or it will correctly determine that the task is not schedulable ($w_i^{n+1} > D_i$). Finding an effective start value for w_i^0 is the subject of considerable conjecture (see for example some recent results [14]). We shall return to this issue in the integrated approach.

The task set illustrated in Table 1 can be analysed using RTA (with the priorities assigned by the optimal deadline monotonic ordering). Unfortunately not all tasks are schedulable, as shown in Table 2. Note the value 1 denotes the highest priority in this example (and 10 the lowest); this is the normal representation in the scheduling literature but is the reverse of the order used by Ada.

Task ID	T	C	D	P	R
τ_1	10	1	4	1	1
τ_2	50	2	50	6	15
τ_3	65	1	30	5	10
τ_4	10	2	8	2	3
τ_5	20	1	20	3	4
τ_6	30	5	20	4	9
τ_7	50	4	50	7	19
τ_8	100	13	100	8	48
τ_9	200	26	150	9	FAIL
τ_{10}	1500	80	900	10	FAIL

Table 2. RTA Analysis

3.2 PDA for EDF Systems

Processor-Demand Analysis [5, 4] considers the entire system in one sequence of tests. It does not compute R_i but uses the property of EDF that at any time t only jobs that have an absolute deadline before t need to execute before t. So the test takes the form (the system start-up is assumed to be time 0):

$$\forall t > 0 : \ \ h(t) \ \leq \ t \tag{3}$$

where $h(t)$ is the total load/demand on the system (all jobs that have started since time 0 and which have a deadline no greater than t). A simple formulae for $h(t)$ is therefore (for $D \leq T$):

$$h(t) \ = \ \sum_{j=1}^{N} \left\lfloor \frac{t + T_j - D_j}{T_j} \right\rfloor C_j \tag{4}$$

The need to check all values of t is reduced by noting that only values of t that correspond to job deadlines have to be assessed. Also there is a bound on t. An unschedulable system is forced to fail inequality (3) before the bound L. A number of values for L have been proposed in the literature, here we give one that we shall use in the integrated approach. It is called the *synchronous busy period* [20, 22] and is denoted here by L_B. It is calculated by forming a similar recurrence relationship to that used for RTA:

$$s^{m+1} \ = \ \sum_{i=1}^{N} \left\lceil \frac{s^m}{T_i} \right\rceil C_i \tag{5}$$

The recurrence stops when $s^{m+1} = s^m$, and then $L_B = s^m$. Note that the recurrence cycle is guaranteed to terminate if $U \leq 1$ for an appropriate start value such as $s^0 = \sum_{i=1}^{N} C_i$.

With all available estimates for L there may well be a very large number of deadline values that need to be checked using inequality (3) and equation (4). This level of computation has been a serious disincentive to the adoption of EDF scheduling in practice. Fortunately, a new much less intensive test has recently been formulated [24, 25]. This test, known as QPA (Quick Processor-demand Analysis), starts from time L and integrates backwards towards time 0 checking a small subset of time points. These points are proved [24, 25] to be adequate to provide a necessary and sufficient test.

The QPA algorithm is encoded in the following pseudo code in which `D_min` is the smallest relative deadline in the system and `d_min(t)` is the smallest absolute deadline strictly less than `t`:

```
t := d_min(L)
loop
  s := h(t)
  if s <= D_min then exit (schedulable)
  if s > t then exit (unschedulable)
  if s = t then
    t := d_min(s)
  else
    t := s
  end if
end loop
```

In each iteration of the loop a new value of `t` is computed. If this new value is greater than the old value the system is unschedulable. Otherwise the value of `t` is reduced during each iteration and eventually it must become smaller than the first deadline in the system and hence the system is schedulable.

Theorem 1. ***([24, 25])*** *A general task set is schedulable if and only if $U \leq 1$, and the iterative result of the QPA algorithm is $s \leq D_{min}$ where D_{min} is the smallest relative deadline of the task set.*

If the example of Table 1 is scheduled entirely with EDF then the QPA test easily determines that it is schedulable.

3.3 Hierarchical Scheduling

In a recent paper on hierarchical scheduling [23], the following result was given for the situation where a collection of EDF tasks are scheduled within a fixed priority server (eg. a deferrable [17] or sporadic server [21]).

Theorem 2. ***(adapted from [23])*** *In a two level hierarchical system when the local scheduler is EDF, a set of periodic or sporadic tasks in the application is schedulable if all of the following conditions are true:*

1. *The utilization of the task set is less than the capacity of the server.*
2. *All tasks in the application are released simultaneously at time 0.*
3. *$\forall d_i \in (0, L_B)$, $R(h(d_i)) \leq d_i$ where L_B is the worst-case synchronous busy period, and the term $R(h(t))$ is the worst-case response time for the load $h(t)$ when executed within the server.*

We shall use this result in the following integrated analysis for FP and EDF.

4 Integrated Analysis of FP and EDF

Integration comes from linking RTA and PDA (QPA) using the approach developed for hierarchical scheduling. Under EDF the amount of work that must be completed by time t is represented by $h(t)$. But during the execution of an EDF job, some FP jobs may preempt and use some of the available time. It follows that a minimum requirement on the combined task set is that:

$$\sum_{j \in \mathbf{A}^{FP}} C_j \ + \ C_{min} \ \leq D_{min}$$

where C_{min} is the computation time of the EDF task with the shortest deadline (D_{min}).

By modeling the entire EDF load as a single low priority task within a FP system, $h(t)$ becomes the computation time of this lower priority task. And hence the response time of this task is the earliest completion time for the EDF load generated by time t. Define $R(h(t))$ to be this completion time. The fundamental schedulability test for EDF becomes:

Theorem 3. *In the combined EDF+FP system when all EDF tasks run at the lowest priority level, the EDF tasks are schedulable if and only if:*

1. $U \leq 1$ *where* U *is the total utilization of the whole task set (including EDF and FP tasks).*
2. $\forall d_i \in (0, L_B)$, $R(h(t)) \leq d_i$, *where* L_B *is the synchronous busy period of the whole task set.*

Proof. From Theorem 2's discussion $\forall t > 0$, $R(h(t)) \leq t \Rightarrow h(t) \leq A(t)$ where $A(t)$ is the worst-case available processor execution time in a given time interval [0,t]. Also from Theorem 2, the system is schedulable if and only if $\forall d_i > 0$, $R(h(d_i)) \leq d_i$.

The upper bound L_B can be obtained from the argument of Liu and Layland [19], if there is an overflow in any tasks' arrival pattern, then there is also an overflow without idle time prior to it when all tasks arrive simultaneously. Since all EDF tasks run at the lowest priority level, when all tasks arrive simultaneously at time 0, then the busy period of the EDF tasks ends when there is no pending tasks in the system. Hence the schedulability check can be bounded to the synchronous busy period of the whole task set. □

To obtain the value of $R(h(t))$ within the FP domain, equation(1) becomes:

$$R(h(t)) \ = \ h(t) \ + \sum_{j \in \mathbf{A}^{FP}} \left\lceil \frac{R(h(t))}{T_j} \right\rceil C_j \tag{6}$$

This is again solved using the recurrence relationship identified in equation (2). The value of L_B is calculated by equation (5).

The QPA algorithm outlined in Section 3.2 is essentially unchanged. Since $R(h(t))$ is non-decreasing with t, we can use the same argument used in the proof of Theorem 1 to show that only line in the pseudo code needs altering: the assignment to `s` which must now be:

```
s := R(h(t))
```

To test a complete system the response time equation is evaluated for a series of ‘loads’ starting with $h(L_B)$ and decreasing each iteration of the QPA loop. This usage of RTA facilitates an efficient start value (w^0) for equation (6). In general, the utilization of the FP tasks will be relatively small and the size of $h(t)$ large when compared to the typical FP task’s execution time. In these circumstances the start value derived by Bril et al [8] is the best one to use:

$$w^0 = \frac{h(t)}{(1 - U_{FP})} \tag{7}$$

So in the example the three fixed priority tasks ($\tau_1 ... \tau_3$) are easily schedulable – the values for these task are given in Table 3.

Task ID	T	C	D	P	R
τ_1	10	1	4	1	1
τ_2	50	2	50	3	4
τ_3	65	1	30	2	2

Table 3. RTA Analysis of FP Tasks

The synchronous busy period estimation of L_B is 988. Using equation (4) to compute $h(988)$ gives the value 815. Equation (7) for $h(t) = 815$ provides an initial value for the RTA analysis of 965. Equation (6) is then solved with this initial value to give $R(815) = 967$. As $988 > 967$ this initial QPA test is positive (ie. $R(h(988)) < 988$). From the value of 967 the iteration continues. Table 4 shows this process for all the stages that the QPA analysis requires. Termination occurs when t becomes less then the shortest EDF deadline in the system (ie $6 < 8$). The result is that on all iterations $R(h(t)) \leq t$ and therefore the EDF part of the system is deemed schedulable. And hence the dual scheduled complete system is schedulable.

t	$h(t)$	w^0	w^1	w^2	$R(h(t))$	t	$h(t)$	w^0	w^1	w^2	$R(h(t))$
988	815	965	967		967	373	271	321	323		323
967	803	951	954		954	323	224	265	268		268
954	800	947	948		948	268	184	218	220		220
948	765	906	908		908	220	158	187	188		188
908	750	888	889		889	188	128	152	155		155
889	643	761	764		764	155	113	134	136		136
764	570	675	677		677	136	73	86	88		88
677	485	574	576		576	88	41	49			49
576	424	502	505		505	49	17	20	22	23	23
505	367	435	436		436	23	10	12	15		15
436	313	371	373		373	15	2	2	6		6

Table 4. Full Analysis of EDF Subsystem

Note Table 4 includes all the calculations needed to verify this system. A useful way of estimating the effort required to complete the verification process is to count the number of ceiling/floor function calls that are required. Within this example, for all but two of the steps only two iterations are needed to compute the response time for the EDF load. For one step three iterations were needed; for another only one. The combination of an efficient start value to this part of the algorithm and the use of the QPA approach means that a total of only 69 ceiling/floor computations have been necessary to test the entire system (3 for the FP part, 22 processor demand calculations and 44 w evaluations).

5 Programming EDF and FP in Ada 2005

To implement combined EDF and FP systems requires either support from an underlying operating system (OS) or from a programming language. Unfortunately few commercial OSs support EDF let alone combined EDF/FP. Similarly, programming languages aimed at the embedded and real-time domain are weak in their provisions for real-time abstractions and notations. The only engineering languages that does support a variety of dispatching policies is Ada; this section therefore focussed on the implementation of combined EDF/FP systems in that language.

A typical time-triggered task type, dispatched by FP, has the form:

```
task type Periodic_FP (Pri : Priority; Period_in_MS : Integer) is
  pragma Priority(Pri); -- fixed priority
end Periodic_FP;

task body Periodic_FP is
  Next_Release : Time;
  Period : Time_Span := Milliseconds(Period_in_MS);
begin
  Next_Release := Clock;
  loop
    -- application code
    Next_Release := Next_Release + Period;
    delay until Next_Release;
  end loop;
end Periodic_FP;

Actual_FP_Periodic_Task : Periodic_FP(16,25);
  -- task with priority of 16 and a period of 25 ms
```

The time types and the clock function are all defined in the predefined language package `Ada.Real_Time`. Note as is typical in FP systems, the actual deadline of the task is not represented in the program's code.

To inform the run-time system that FP dispatching is required, the following pragma is used:

```
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
```

5.1 Supporting EDF scheduling

To support EDF requires two language features:

– representation of the 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 [2]. A predefined package provides support for deadlines:

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

Within a complete system, the priority range is split into different ranges for different dispatching policies. For the approach adopted in this paper, a collection of high values (larger integers) are used for FP, and a range of lower values are used for EDF. If the base priority of a task is within the range of priorities defined as `EDF_Across_Priorities` then it is dispatched by EDF. Note also that the ready queues for priorities within this range are ordered by absolute deadline (not FIFO).

The typical code pattern for a periodic task type, scheduling by EDF, is therefore now:

```
task type Periodic_EDF (Pri : Priority; Period_in_MS,
                        Deadline_in_MS : Integer) is
  pragma Priority(Pri);
  pragma Relative_Deadline(Milliseconds(Deadline_in_MS));
end Periodic_EDF;

task body Periodic_EDF is
  Next_Release: Time;
  Period : Time_Span := Milliseconds(Period_in_MS);
  Rel_Deadline : Time_Span := Milliseconds(Deadline_in_MS);
begin
  Next_Release := Clock;
  loop
    -- application code
    Next_Release := Next_Release + Period;
    Delay_Until_and_Set_Deadline(Next_Release, Rel_Deadline);
  end loop;
end Periodic_EDF;

Actual_EDF_Periodic_Task : Periodic_EDF(5,25,20);
  -- 5 is within the EDF_Across_Priorities range.
  -- Period of 25 ms, and relative deadline of 20 ms.
  -- The first absolute deadline of the task is 20ms from the time the
  -- task is created.
```

The different priority ranges are indicated by the following pragmas (in this example, the bottom ten are allocated to EDF and the next higher ten to FP).

```
pragma Priority_Specific_Dispatching(FIFO_Within_Priorities, 20, 11);
pragma Priority_Specific_Dispatching(EDF_Across_Priorities, 10, 1);
```

Note the code for a task object of type `Periodic_EDF` could be dispatched by FP by merely changing the range into which its priority falls – there is no need to make any changes to the code of the task.

5.2 Protected objects

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.

In Baker's stack-based protocol, two distinct notions are introduced for these policies[2]:

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

With preemption levels, each task is assigned a static preemption level (represented by the task's base priority), and each protected object 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.

The definition of Ada 2005 provides support for Baker's algorithm – see Burns and Wellings[10] for details.

[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.

6 Dynamic Partitioning of EDF and FP Tasks

Rather than partition the two types of tasks (with the high integrity tasks being scheduled as FP and the others as EDF), a dynamic scheme can be used. This is particularly appropriate where the system has a mixture of high-integrity (hard) and soft real-time tasks. In one such approach, all tasks are initially scheduled using EDF, but if it becomes critical for a high-integrity task to execute, it is pulled from the EDF level and is allowed to complete as a FP task where the execution behaviour is more predictable. This means of scheduling tasks is known as *dual priority scheduling* [13, 7].

Each high-integrity task has a promotion time, S_i, that indicates when (relative to its release) it must move to its higher level. The scheduling equations when they compute worst-case response time R_i immediately allow this value to be obtained: $S_i := T_i - R_i$. The full analysis is given in [13]. Here, we focus on the programming aspects.

To support dual priority scheduling in Ada 2005 requires two features:

– a means of dynamically changing a task's priority, and
– a means of making such a change at a particular point in time.

The former is supported by a straightforward dynamic priority package. More significantly, Ada 2005 has introduced a new abstraction of a timing event to allow code to be executed at specified times without the need to employ a task/thread. These events are like interrupts, but they are generated by the progression of the real-time system clock. Associated with a timing event is a handler that is executed at the allotted time. An implementation should actually execute this handler directly from the interrupt handler for the clock device. This leads to a very efficient implementation scheme.

It would be quite possible for each task to have its own handler that alters its base priority when required. However, using the 'tagged' feature of the event type it is possible to have just a single handler that uses the event parameter to reference the correct task. To do this, the type `Timing_Event` is first extended (in a library package) to include a task ID field:

```
type Dual_Event is new Timing_Event with
record
  TaskID : Task_ID;
end record;
```

The single handler has a straightforward form (it is placed in the same library package as the type definition):

```
protected Dual_Event_Handler is
  pragma Interrupt_Priority(Interrupt_Priority'Last);
  procedure Change_Band(Event : in out Timing_Event);
end Dual_Event_Handler;

protected body Dual_Event_Handler is
  procedure Change_Band(Event : in out Timing_Event) is
    The_Task : Task_ID;
    P : Priority;
  begin
    The_Task := Dual_Event(Timing_Event'Class(Event)).TaskID;
    P := Get_Priority(The_Task);
    Set_Priority(P+10, The_Task);
  end Change_Band;
end Dual_Event_Handler;
```

The specific task ID is obtained by two view conversions, the parameter `Event` is converted first to a class-wide type and then to the specific type `Dual_Event`. The run-time system does not know about the extended type but the underlying type is not changed or lost, and is retrieved using the view conversions.

Now consider a high integrity task that has a base priority 4 in the EDF level and 14 in the upper FP level. Initially it must run with priority 14 to make sure all its initialisation is complete. It has a period of 50 ms and a relative deadline set to the end of its period (i.e. also 50 ms). Its promotion point is 30 ms after its release.

```
task Example_Hard is
  pragma Priority(14);
end Example_Hard;

task body Example_Hard is
  Dual_E : Dual_Event := (Timing_Event with TaskID => Current_Task);
  Start_Time : Time := Clock;
  Period : Time_Span := Milliseconds(50);
  Promotion : Time_Span := Milliseconds(30);
begin
  Dual_E.Set_Handler(Start_Time + Promotion,
     Dual_Event_Handler.Change_Band'access);
  Set_Deadline(Start_Time + Period);
  Set_Priority(4); -- now dispatched according to EDF
  loop
    -- application code of the task
    Start_Time := Start_Time + Period;
    Dual_E.Set_Handler(Start_Time + Promotion,Dual_Event_Handler.
       Change_Band'access);
    Set_Priority(4);
    Delay_Until_And_Set_Deadline(Start_Time,Period);
  end loop;
end Example_Hard;
```

If the event triggers then the task's priority will be raised to 14 and it will be subject to fixed priority dispatching; as a result it will complete the execution of its code by its deadline (guaranteed by the scheduling analysis). It will then set its new release time (`Start_Time`), set its event handler again and lower its priority back to the value within the EDF range. Its deadline will be quite soon and so it is likely to continue executing into its delay statement.

If the system is not heavily loaded, the hard task will complete its invocation before the promotion point. The second call of `Set_Handler` will then cancel the previous call.

A final point to note with this example concerns the use of protected objects by the tasks. If such an object is used by a hard task then it must have a ceiling in the 11..20 range, otherwise an error would occur if the task with its promoted priority calls the object. Using the event handler to also change the ceiling priorities of such protected objects is unlikely to be justified.

7 Conclusions

This paper has considered the means by which EDF and FP scheduling can be used together to produce a hybrid scheme that has many of the benefits of the individual approaches. From EDF comes efficiency, from FP comes predictability. With this combination it is possible to run systems that have a number of critical tasks that require the determinacy of FP together with soft tasks that utilize most of the remaining capacity of the processor.

Two aspects of the integration of EDF and FP have been covered. First the integration of the forms of analysis available – specifically, RTA for FP and PDA (with QPA) for EDF. Secondly, integration of the implementations via the programming model available in Ada 2005. This enables industrially relevant systems to be produced that make use of FP and EDF, and moveover allows a late binding to the scheduling scheme to be employed. Indeed a dynamic approach is discussed that allows tasks at run-time to migrate between scheduling routines.

References

1. N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.
2. T.P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1), March 1991.
3. S.K. Baruah and A. Burns. Sustainable schedulability analysis. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006.
4. S.K. Baruah, R.R. Howell, and L.E. Rosier. Feasibility problems for recurring tasks on one processor. *Theorectical Computer Science*, 118:3–20, 1993.
5. S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptive scheduling of hard real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 182–190, 1990.
6. S.K. Baruah, A.K. Mok, and L.E. Rosier. Hybrid-priority scheduling of resource-sharing sporadic task systems. In *IEEE Real-Time Systems and Applications Symposium (RTAS)*, 2008.
7. G. Bernat and A. Burns. Combining (n m)-hard deadlines with dual priority scheduling. In *Proceedings 18th IEEE Real-Time Systems Symposium*, pages 46–57, 1997.
8. R.J. Bril, W.F.J. Verhaegh, and E-J.D. Pol. Initial values for on-line response time calculations. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 13–22, 2003.
9. R. Brukardt(ed). Ada 2005 reference manual. Technical report, ISO, 2006.
10. A. Burns and A. J. Wellings. *Concurrency and Real-Time Programming in Ada 2005*. Cambridge University Press, 2007.
11. A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley Longman, 4th edition, 2009.
12. G. Buttazzo. Rate monotonic vs. EDF: Judgement day. *Real-Time Systems Journal*, 29(1):5–26, 2005.
13. R.I. Davis and A. J. Wellings. Dual priority scheduling. In *Proceedings 16th IEEE Real-Time Systems Symposium*, pages 100–109, 1995.

14. R.I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority pre-emptive systems. *IEEE Transaction on Computers*, 57(9):1261–1276, 2008.
15. M.G. Harbour and J.C.P. Gutirrez. Response time analysis for tasks scheduled under EDF within fixed priorities. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 200–209, 2003.
16. M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, 29(5):390–395, 1986.
17. J.P. Lehoczky, L. Sha, and J.K. Strosnider. Enhanced aperiodic responsiveness in a hard real-time environment. In *Proceedings 8th IEEE Real-Time Systems Symposium*, pages 261–270, 1987.
18. J.Y.T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*, 2(4):237–250, 1982.
19. C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.
20. I. Ripoll and A.K. Mok. Improvement in feasibilty testing for real-time tasks. *Journal of Real-Time Systems*, 11(1):19–39, 1996.
21. B. Sprunt, L. Sha, and J. P. Lehoczky. Aperiodic task scheduling for hard real-time systems. *Real-Time Systems*, 1:27–69, 1989.
22. M. Spuri. Analysis of deadline schedule real-time systems. Technical Report 2772, INRIA, France, 1996.
23. F. Zhang and A. Burns. Analysis of hierarchical EDF preemptive scheduling. In *IEEE Real-Time Systems Symposium (RTSS)*, pages 423–435, 2007.
24. F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. Technical Report YCS 426, University of York, 2008.
25. F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transaction on Computers – to appear*, 2008.
26. K.M. Zuberi, P. Pillai, K.G. Shin, and K.G. Emeralds. A small memory real-time microkernel. In *ACM Symposium on Operatinf Systems Principles*, pages 277–291, 1999.