Task Attribute-Based Scheduling - Extending Ada's Support for Scheduling

A. Burns and A.J. Wellings

Department of Computer Science
University of York, UK
Email: {burns,andy}@cs.york.ac.uk

Abstract. Scheduling policies are of crucial importance to real-time systems. Ada currently gives full support to the most popular policy, fixed priority dispatching. In this paper we argue that the language should be extended to support new paradigms such as EDF, and combined paradigms. Possible extensions to the language in the form of new pragmas are described.

1 Introduction

One of the major features of Ada95 is its support for the fixed priority scheduling. A complete and consistent model is provided that allows this common dispatching approach to be used in real industrial applications. The model is also retained in the Ravenscar subset [2–4]. Ada does not however restrict itself to this single paradigm; it allows other dispatching policies to be defined. Unfortunately without the force of the language definition, new policies have not been introduced by vendors or specified in any secondary or de facto standards.

The current process of considering amendments to the language allows the dispatching rules of Ada to be expanded. For example, an AI on non-preemptive (fixed priority) scheduling has already been agreed. This required some rewording of the ARM (sections D.2.1 and D.2.2) but in the end required only a minor change. The only difference between `Fifo_Within_Priority` and `Non_Preemptive_Fifo_Within_Priority` is the former has an additional dispatching point (whenever a higher priority task is on a ready queue).

Much more is now understood about scheduling than was the case when the original Ada definition was created. If Ada is to remain a viable language of choice in the real-time domain it must increase its expressive power in this area. Indeed significant advances would help reassert Ada's prominence.

The literature on scheduling methods implies that Ada could be expanded in three areas:

- Support for new paradigms, eg. EDF,
- Support for combining paradigms,
- Support for user-defined paradigms.

In this paper we consider the first two of these areas. The use of dynamic priority assignment (and other language features) does support a certain level of user-defined scheduling, including coverage for some of the new paradigms. However we feel that direct support, via the definition of APIs in Annex D, will increase the likelihood of implementations become available, will standardize on the approaches, and show the continuing development of Ada. For a detailed discussion on adding user-defined schedulers to Ada see proceedings of the last workshop [9].

2 Support for New Dispatching Policies

In addition to fixed priority dispatching (identified by the use of the dispatching policy `Fifo_Within_Priority`) the following approaches are commonly referenced and could be used in different applications if Ada supported them:

- EDF - Earliest Deadline First
- Value-based scheduling
- Round-robin scheduling
- Ready-Queue Priority

We shall consider each of these in turn.

Inevitably adding new schemes to Ada will require that the scheduler will need access to programmer-supplied information, which must be associated with each task in the system. In Ada, this associated is usually performed using task attributes. In this paper we propose a single change to the Ada dispatching points which will allow various scheduling approaches to be implemented.

A task dispatching point occurs whenever a task dispatching attribute is set.

An attribute is designated as a ‘dispatching attribute’ by use of a new pragma:

```
pragma Dispatching_Attribute(<Attribute-Definition>);
```

An example of usage will be given later.

We also propose that the initial value of a task attribute can be set via a pragma in a task specification. This is needed to make sure that a task activates and starts its execution under the correct dispatching parameter. There does not seem to be a standard way of doing this; so we invent the following pragma to achieve this initialisation. Again an example of usage will be given later.

```
pragma Initial_Attribute_Setting(
    with package My_Attribute is new Ada.Task_Attribute,
    My_Attribute.Attribute);
```

Using the new dispatching rule and the new pragma, we now consider how various alternative scheduling schemes can be specified in Ada. We assume that whatever value is used as a measure of “execution eligibility”, inheritance occurs when one Ada task is blocked by another (for example, where a parent task is waiting for a child task to elaborate, the child tasks inherits the parent’s execution eligibility if it is greater that is own base execution eligibility).

Ada defines the following pragma to allow the programmer to set the dispatching policy in a partition.

```
pragma Task_Dispatching_Policy(Policy_Identifier)
```

The following identifiers might be legal:

```
Fifo_With_Priorities -- currently defined
Fifo_Within_Edf
Fifo_Within_Value
Round_Robin
```

The above assumes that there is essentially only a single level dispatching policy. Section 3 of this paper considers the very important issue of combining schemes. This will have an impact on the individual proposals in this section

2.1 EDF Scheduling

Earliest deadline first is a popular paradigm as it has been proven to be the most efficient scheme. If a set of tasks is schedulable by any dispatching policy then it will also be schedulable by EDF. However EDF is not without its drawbacks; it is unpredictable during overloads and schedulability tests for non-trivial task characteristics are not straightforward. To support EDF requires two language features:

- representation of deadline
- representation of preemption thresholds for protected objects

The first is obviously required; the second is the EDF equivalent of priority ceilings and allows protected objects to be 'shared' by multiple tasks [baker]. It is an anomaly that Ada's support for real-time does not extend as far as having a direct representation for 'deadline', but the task attribute features does allow a deadline attribute to be assigned.

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Task_Attributes;
with Ada.Real_Time; use Ada.Real_Time;
with System;
package Edf_Scheduling_Support is
  package Deadlines is new
    Ada.Task_Attributes(Time, Time_Last);
    pragma Dispatching_Attribute(Deadline.Attribute_Handle);
  subtype Preemption_Level is System.Priority;
end Edf_Scheduling_Support;
```

Note that, if the pragma `Initial_Attribute_Setting` is not used, the default value is to preset the deadline of all tasks as the maximum time possible into the future.

A program can manipulate deadlines and even catch a deadline overrun via the ATC feature. For example, consider the representation of a simple periodic task in Ada with EDF scheduling.

```
task type Periodic_Task(First_Release : access Time;
                        Release_Interval: access Time_Span;
                        Deadline: access Time_Span) is
   pragma Initial_Attribute_Setting(
        Deadlines, First_Release.all + Deadline.all);
end Periodic_Task;

task body Periodic_Task is
  Next_Release: Time;
begin
  Next_Release := First_Release.all;
  delay until Next_Release;
  loop
    select
      delay until Deadlines.Value;
    then abort
      -- code
    end select;
    Next_Release := Next_Release + Release_Interval.all;
    Deadlines.Set_Value(Next_Release + Deadline.all);
               -- dispatching point
    delay until Next_Release;
  end loop;
end Periodic_Task;
```

Note, that as soon as the deadline is reset then the task has its new deadline and, therefore, is likely to be preempted before it executes the delay until statement. However, as its deadline approaches, it will be rescheduled in time to iterate around the loop.

To represent preemption levels it would be possible to use the existing priority scheme, and just assign priorities to protected objects in the usual way. However, as we may wish to retain the original use of priority in a mixed scheme, we propose a new pragma here - `Preemption_Level` which takes the same parameter as pragma `Priority`. The new `Locking_Policy` could be called `Preemption_Level_Locking`.

The dispatching rules for an EDF scheme (to be identified by the policy id: `Fifo_Within_Edf`) would need to be explicit about all dispatching points. But this should be straightforward. Note a change to a tasks' deadline must be one such point - the rules regarding deadline changes will be similar to those of dynamic priorities. At this time we feel it is an open issue as to whether a new queuing discipline is needed to give comprehensive support to EDF.

2.2 Value-Based Scheduling

In the more flexible applications, QoS becomes a factor. Unfortunately the definitions of QoS does not allow a single scheduling model to be defined. However there is support for using a value-based approach - where each task has a value which can be change dynamically) and the task with the highest value is the one that is dispatched. Of course, priority gives exactly the same behaviour as value and could be used in a pure value-based approach. Nevertheless, there might be some utility in having an extra parameter. First, for mixed schemes both might be required; and secondly, value usually has a wide range (e.g. all the integers) rather than the restricted range of priority.

If a distinctive value parameter is needed then again an attribute of the task could be used and any assignment to the attributed would be a dispatching point.

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Task_Attributes;
with Ada.Real_Time; use Ada.Real_Time;
with System;
package Value_Based_Scheduling_Support is
  package Values is new Ada.Task_Attributes(Integer, 0);
end Value_Based_Scheduling_Support;
```

A periodic task scheduled according to value could be written as:

```
task type Periodic_Task(First_Release : access Time;
                        Release_Interval: access Time_Span;
                        Deadline: access Time_Span;
                        Value : Integer) is
  pragma Initial_Attribute_Setting(Values, Value);
end Periodic_Task;

task body Periodic_Task is
  Next_Release: Time;
begin
  Next_Release := First_Release.all;
  delay until Next_Release;
  loop
    select
      delay until Next_Release + Deadline.all;
    then abort
      -- code
    end select;
    Next_Release := Next_Release + Release_Interval.all;
    delay until Next_Release;
  end loop;
end Periodic_Task;
```

Note, in this case, the value attribute is not changed.

2.3 Round-Robin Scheduling

The motivation for a round-robin schedule has already been discussed at IRTAW [8] and a proposal has recently been published [5]. This proposal envisages a mixed scheme (fixed priority and round robin) - see Section 3. Here we simply note that a general round-robin scheme can be implemented via task attributes:

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Task_Attributes;
with Ada.Real_Time; use Ada.Real_Time;
with System;
package Round_Robin_Scheduling_Support is
  Default_Quantum : constant Time_Span := Milliseconds(...);
  package Round_Robin_Quantum is new
    Ada.Task_Attributes(Time_Span, Default_Quantum);
end Round_Robin_Scheduling_Support;
```

2.4 Ready-Queue Priority Scheduling

A less well known scheme, but one that has actually been implemented in real applications, is a variation on fixed priority. In the standard approach a wide range of priorities is preferred as this increases the likelihood of schedulability - schedulability is maximised if each task has its own distinct priority. But for efficient implementation (minimising stack usage etc) the smallest range of priority (commensurate with schedulability) is desirable. An interesting compromise [7], that minimises the number of priority levels needed at run-time whilst giving near optimal schedulability involves giving tasks two priorities. The first is used for queuing (e.g. on the ready queue), this is a wide ranging priority parameter. The second is the actual priority the task executes at. The range of this parameter is much less. In effect, the wide range of the first parameter is collapsed (as far as possible) to form the second range. Some consistence between the two parameters is required. For example if task `A` has a lower queuing priority than task `B` then it must have a lower (or equal) execution priority. To support this policy (which might be designated `Constrained_Fifo_Within_Priority`) a new pragma is needed (`Ready_Queue_Priority`) which would leave the existing priority to imply execution priority. To avoid confusion with dynamic priorities this feature would be restricted.

2.5 Implications for Ravenscar

A number of the schemes discussed above are applicable to the restricted domain to which Ravenscar is aimed. Ravenscar itself, as designated by:

```
pragma Profile(Ravenscar);
```

uses (requires) fixed priority dispatching (`Fifo_Within_Priority`). It follows that other profiles could be defined with different dispatching schemes:

```
pragma Profile(Ravenscar_Edf);
pragma Profile(Ravenscar_Constrained_Fifo_Within_Priority);
pragma Profile(Ravenscar_Value);
```

3 Mixed Scheduling Schemes

One of the more recent, but significant, trends in scheduling is the mixing of schemes. This reflects the view that systems are often not isolated entities. They either coexist with other systems on the same processing resources, or are composed of separate subsystems with different non-functional properties. POSIX has always had a two-tier model (process scheduling and thread scheduling), and some domain standards (e.g. APEX [6]) requires two levels.

Mixed systems often also require protection mechanisms, both temporal protection (one system cannot steal processing resources from another) and memory protection. The former can be accomplished by budget control; the latter is somewhat outside the scope of a language definition (and therefore this paper). In this section we consider the combination of fixed priority dispatching, EDF, round-robin and the value-based approach. Non-preemption is really a stand-alone scheme, although a non-preemptive EDF policy could be defined as easily as a non-preemptive fixed priority policy. The ready-queue priority method is also a specific scheme for efficient implementation and so is not considered in this discussion.

Although multi-level scheduling hierarchies are possible we restrict our consideration to two-tier schemes. In a somewhat arbitrary manner, but following existing standards and approaches, we consider only two possible schemes for the base dispatching policy.

The first approach is to have fixed priority at the base. It is the default Ada approach and has the flexibility that comes from the application being able to interpret (assign) whatever meaning it wishes to the notion of priority. On top of the priority policy (e.g. the policy to be used at each priority level) the programmer could use FIFO, EDF, Round-Robin, a Value-based scheme or any other implementation defined algorithm. For example it is possible to use three of these schemes together to implement a flexible approach called Jorvik [1]. Here background tasks runs as Round-Robin; all real-time application task join an EDF queue, but if these task have hard guarantees then they must jump to a fixed higher priority to ensure they complete by their deadlines - see Figure 1 for this structure.

The other contender for a base policy is round-robin in the sense of a timed partitioning of the CPU. The APEX standard uses this approach and time-slicing is clearly the easiest way of partitioning a single processor across a number of distinct applications. In effect there is a cyclic executive as the primary dispatcher. Note, this is not strictly round-robin as each application is not necessarily given the same quantum of execution time. Rather a repeated time-line is laid out in which each application is given a slot.

In the following subsection we consider how these two schemes could be incorporated into Ada. First we consider, scheduling within a partition and then scheduling across partitions.

3.1 Scheduling within a Partition

Within Priority

The basic approach could be requested by the use of a new dispatching identifier.

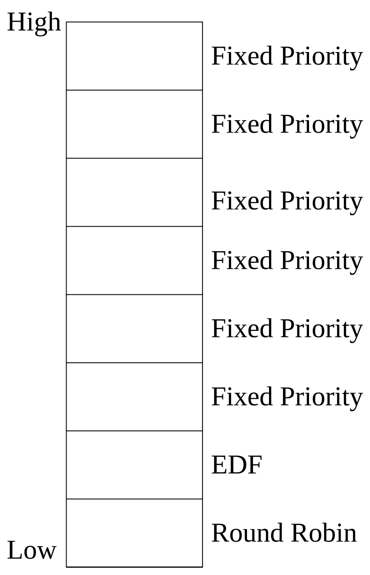

Fig. 1. A Use of a Scheduling Hierarchy

```
pragma Task_Dispatching_Policy(Priority_Specific);
```

When `Priority_Specific` is used, the dispatching policy is defined on a per priority level. This is achieved by the use of a new configuration pragma:

```
pragma Priority_Policy (Policy_Identifier, Priority);
```

The `Policy_Identifier` shall be `Fifo`, `Edf`, `Round_Robin` or an implementation-defined identifier. At all priority levels, the default `Priority_Policy` is `Fifo`. The locking policy associated with the `Priority_Specific` task dispatching policy is `Ceiling_Locking`.

A task will be subject to the policy in effect for the priority it is assigned. The use of `Set_Priority` to assign a task a new priority may have the effect of changing the task's dispatching policy. This is reasonable semantics as the dispatching policy is a property of the system not an individual task.

Other Schemes

Whilst it is possible to defines other combination which can be applied at the partition level, most of these offer little added value. We have mentioned the possibility of priority within round-robin scheduling, however there is no mechanism within a partition to exploit this. Priority within round-robin is best applied at the partition level.

3.2 Partition-Level Scheduling

Partitions in Ada allow groups of task to be established. The reference manual is silent about the scheduling of partitions as they are part of the distribution model and distributed scheduling is certainly not well enough understood to be part of a language definition. But the manual does allow a number of partitions to co-exist on a single processor. In this situation it would be possible to define a partition dispatching policy (time-sliced, priority etc). It would also be more natural to give memory protection between partitions as this is the most natural abstraction for this in the language. Here we consider round-robin scheduling of partitions and priority based scheduling within a partition. However, other schemes are possible (for example, value-based scheduling at the partition level and EDF within a partition)

Inevitably, adding techniques to allow partition level scheduling requires access to partition identifies. Here we assume the following entities:

```
type Partition_Is is range 0 .. Implementation_Defined;
```

which is currently available in the Distributed Systems Annex.

Round-Robin Partition Scheduling

The most simple form of round-robin scheduling is to assume that all partitions are given the same system-defined quantum. Hence, all that is required is a pragma to indicate that partition-level scheduling is round-robin.

```
pragma Partition_Dispatching(Policy_Identifier,
     [,Policy_Argument_Definition]);
```

where the following would be an example of use:

```
Quantum : Ada.Real_Time_Time_Span := Milliseconds(..);
pragma Partition_Dispatching(Round_Robin,Quantum);
```

Cyclic Partition Scheduling

Although, round-robin scheduling allows temporal firewalls to be constructed between partitions, greater flexibility is given by allowing a complete cyclic schedule to be expressed. It will also be necessary to define more precisely the details of the model. For example, in the advent of a partition not using its full time slice (due to no runnable tasks remaining) is the spare resource passed on (by running the next partition early) or is a strict schedule enforced by idling until the time for the next switch?

The inclusion of such a model in Ada would be a significant addition to the languages capabilities. Ada has been used with APEX (for example, by running separate Ravenscar programs in each partition) but the schedule was expressed outside the program.

One approach is to define an Ada-based configuration language to help construct partitions. The following would be part of this language.

```
type Partition_Slot is
  record
    Partition : Partition_Id;
    Length : Ada.Real_Time_Time_Span;
  end;

type Schedule is array(Positive range <>) of Partition_Slots;

Actual_Schedule : Schedule := ...

pragma Partition_Dispatching(Cyclic_Schedule, Actual_Schedule);
```

4 Conclusion

In this paper a number of extensions to the dispatching policies of Ada have been proposed. These would give rise to the following additions to Annex D:

```
pragma Dispatching_Attribute();
pragma Initial_Attribute_Setting();
pragma Locking_Policy(Preemption_Level_Locking);
pragma Task_Dispatching_Policy(Fifo_Within_Edf);
pragma Task_Dispatching_Policy(Fifo_Within_Value);
pragma Task_Dispatching_Policy(Round_Robin);
pragma Task_Dispatching(Constrained_Fifo_Within_Priority);
pragma Ready_Queue_Priority();
pragma Profile(Ravenscar_Edf);
pragma Profile(Ravenscar_Constrained_Fifo_Within_Priority);
pragma Profile(Ravenscar_Value);
pragma Task_Dispatching_Policy(Priority_Specific);
pragma Priority_Policy(Policy_Identifier, Priority);
pragma Partition_Dispatching();
```

An assessment of these extensions by the Workshop, together with the definitions of those that gain support, could make a significant impact to the facilities in, and perception of, the amended version of Ada.

References

1. G. Bernat and A. Burns. Implementing a flexible scheduler in Ada. In D. Craeyneest and A. Strohmeier, editors, *Proceedings of Reliable Software Technologies - Ada Europe 2001*, volume 2043 of *LNCS*, pages 179–190. Springer, 2001.
2. A. Burns. The Ravenscar Profile. *ACM Ada Letters*, XIX(4):49–52, Dec 1999.
3. A. Burns, B.Dobbing, and T. Vardanega. Guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report YCS-2003-348, University of York, Department of Computer Science, 2003.
4. A. Burns, B. Dobbing, and G. Romanski. The Ravenscar tasking profile for high integrity real-time programs. In *Reliable Software Technologies, Proceedings of the Ada Europe Conference, Uppsala*, pages 263 – 275. Springer Verlag, 1998.

5. A. Burns, M. González Harbour, and A.J. Wellings. A round robin scheduling policy for Ada. In *Reliable Software Technologies, Proceedings of the Ada Europe Conference, to appear*. Springer Verlag, 2003.
6. ARINC AEE Committee. Avionics application software standard interface, September 1995.
7. R. Davis, N. Merriam, and N.J. Trace. How embedded applications using an rtos can stay within on-chip memory limit. In *12th Euromicro Workshop on Real-Time Systems - Session*, 2000.
8. M. Aldea Rivas and M. González Harbour. Extending Ada's real-time systems annex with the POSIX scheduling services. In M. González Harbour, editor, *Proceedings of the 10th International Real-Time Ada Workshop*, pages 20–26. ACM Ada Letters, 2001.
9. M. Aldea Rivas and M. González Harbour. Application-defined scheduling in Ada. In J.L Tokar, editor, *Proceedings of the 11th International Real-Time Ada Workshop*, pages 77–84. ACM Ada Letters, 2002.