Report of Session: Flexible Scheduling in Ada

Chair: Alan Burns
Rapporteurs: Andy Wellings, Tullio Vardanega

1 Introduction

The goal of this session is to determine whether Ada's support for scheduling is adequate for modern and future real-time systems.

The chair presents an overview of the Ada model of priority-based scheduling and argues that the ARM needs to be changed to allow other scheduling schemes to be defined. He also notes that, although implementations are currently allowed to provide other schemes, experience has shown that this will not happen, unless the language definition sanctions what other schemes should be supported.

The chair's view is that adding new schemes would not imply that they should all be supported by an implementation of Annex D. If they were supported, however, they should be supported as specified in the Annex.

Schemes that Ada could support include: new paradigms such as EDF, Value-based, Round Robin, Servers (of various types), Combined paradigms, as well as primitives for "roll-your-own" user-defined scheduling.

The notion of Combined scheduling would include:

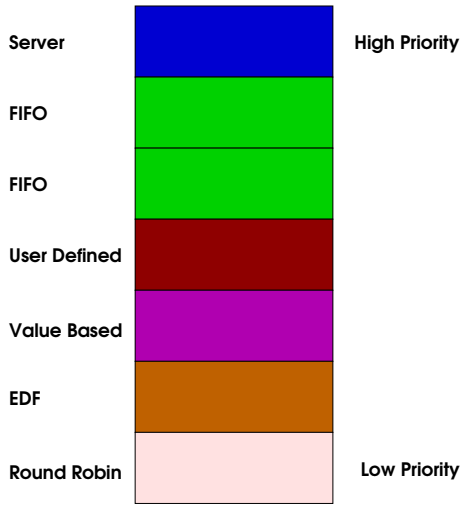
- Within a partition
  - FIFO within priority
  - others
- Across partitions (Partition-level scheduling)
  - round-robin
  - cyclic scheduling
  - others

A discussion arises on whether this additional feature was an important topic for Ada or whether users just wanted access to standard OS scheduling approaches. Members of the groups also note that the latter option occurs more and more frequently as modeling and design tools increasingly offer automatic code generation assuming the existence of an underlying OS.

No resolution of the issue is sought, but it is deferred, keeping an open mind on it, until after the session will have explored the solution space in more detail.

2 Alternative Scheduling Schemes

Alan Burns then proposes a model that leaves the underlying Ada scheduling model unchanged (that is priority-based scheduling), but that could accommodate different schemes at each priority level. The following diagram illustrates the approach.



To support such an approach requires Preemption levels as well as ceiling priorities and various Dispatching pragmas.

The issue arises whether different scheduling schemes should be allowed to coexist at the same priority level, in particular those that work well together, e.g.: constant bandwidth servers and EDF. The view from the group is that this should not be allowed.

Alan Burns then reviews the proposal in the Burns and Wellings position paper on attribute-based scheduling, where each new scheduling attribute (deadline, value etc) is represented as a pre-defined task attribute and a task dispatching point occurs whenever the attribute is set. One issue identified with this approach is how to initialise a task's attribute.

An alternative approach is suggested during the discussion, whereby the scheduling attribute was an tagged type hierarchy, the root of which simply being the priority and

which subsequent derivations could extend to introduce other attributes and the relevant scheduling policies.

Alan then moves on to review how EDF, valued-based and round robin scheduling could be supported (cf. the Burns and Wellings paper in these proceedings).

One of the main issues with round robin scheduling arises when a task executes with an inherited priority when its quantum expired. In this situation the task must be allowed to continue until it returned to its based priority otherwise significant priority inversion would occur. It is noted during discussion that it is not clear that POSIX adopted a similar approach and that Michael Gonzalez-Harbour (the workshop's main POSIX interface) should request an interpretation on this issue.

One of the issues discussed in previous workshops had been how to provide aperiodic servers (for example sporadic or deferrable servers). There currently is no AI on this topic. Alan indicates that it is not clear what is the best way to support these facilities. At least three alternative approaches could be envisioned: (1) support some explicit ones (in the way of POSIX support for sporadic server); (2) do not support any as they can be programmed; (3) support abstract server with implementation-defined details.

The chair notes that Ada already supports a set of mechanism that allow an application to have control over many aspects of scheduling. For example: Dynamic priorities; Budget control; Timer events. The question is whether these mechanisms are sufficient. For example, can a sporadic server be programmed with them? Finding a well-based answer to this question seems to be more a research project than an exercise to complete within this workshop. However later in the workshop, the notion of task group budgets was introduced and a simple deferrable server was programmed to illustrate its functionality.

3 Application-Defined Scheduling

The paper by Ribas and Gonzalez-Harbour focuses on providing the hooks to allow full application-level scheduling. Michael Gonzalez-Harbour presents the key points of that paper. He argues that Ada should continue to be the reference language for real-time applications and that it must evolve to meet new application required and in response to other real-time languages (such as the Real-Time Specification for Java). Michael's position is that Ada should provide low-level mechanisms that would enable the implementation of complex scheduling frameworks. He is convinced that fixed priority scheduling is not enough to meet future needs.

Michael then presents a vision of a real-time scheduling framework with low-level mechanisms that enable support for the most popular general-purpose schedulers, which can thus meet complex special application requirements integral support of scheduling and synchronization.

The proposal includes a library of implementations for the most popular schedulers and a set of patterns that facilitate putting together the basic scheduling components: budget monitoring, deadline monitoring, mode changes, task groups. Michael lists the requirements that he believes should be satisfied:

- possibility of acceptance tests
- compatibility with previous software components
- blocking outside of the scope of the scheduler (i.e., read)
- synchronization
- change of scheduling parameters
- explicit invocation of the scheduler
- deadline monitoring
- execution budget monitoring
- coexistence of several schedulers

On discussion of the proposed API, the suggestion is made that it might be possible to provide a generic package containing the scheduler. The generic parameters would contain the procedures needed to handle the various scheduling events. If this was feasible then it may be possible for the run-time support system to optimism the implementation so that no added tasks were introduced.

A further simplification is suggested, which allows only one scheduler at a priority level, so that all tasks assigned at that level be scheduled by the scheduler.

Following Michael's outline of an example of an EDF application-level scheduler (cf. the paper for details), Niklas Holsti presents an informal proposal for representing dispatching attributes as tagged types:

```
type Scheduling_Policy is tagged record
 Base_Priority : System.Any_Priority;
 Preemption_Level : Natural; -- or whatever
end record;

type EDF_Attributes is Scheduling_Policy with record
  Deadline : Ada.Real_Time.Time;   -- Could be private
end record;

My_EDF_Attributes : EDF_Scheduling := (...);
task My_EDF_Task is
 pragma Scheduling_Policy (My_EDF_Attributes'Access);
end My_EDF_Task;
```

The base type would have primitive operations, which could be overridden as desired as the type hierarchy grows. Such operations would have to be protected, which Niklas's proposal requires but he does not know how to ensure.

Andy Wellings notes that this problem would be solved the moment we would have protected interfaces, which we would use in the place of simple tagged types. Niklas admits that his proposal has a number of open issues that concern: (1) the atomicity of calls to Scheduling_Policy primitives; (2) the precedence of scheduler task over scheduled tasks; (3) the dispatching overhead of tagged-type operation; (4) the use of tagged types at kernel level, which may obstruct certification.

The discussion then moves on to the issue of space and time partitioning, which appears to be increasingly considered in a number of US defense applications, in the place of a whole range of hand-coded scheduled systems (most of which do not use tasking). Joyce Tokar presents some reflections on this issue.

The basis for space and time partitioning is ARINC 653, which requires that: (i) each partition is a separate application, with separate memory space (spatial partitioning) (ii) has a virtual CPU per partition, which therefore allows multiple software criticality levels on the same hardware and which obviously is of great commercial interest to system providers, in view of the hardware cost reduction.

This approach may be attractive to system builders. One important implication from using ARINC is that one must use the ARINC process (concurrency) model and therefore either exclude or map any language-specific concurrency.

Time partitioning (which presently is the most delicate area of work and of implementation) results from each partition being guaranteed a dedicated window of CPU time. Andy Wellings notes that multi-level scheduling analysis of such architectures is far from easy.

Partitions have attributes: memory (amount and location); processing requirements (time and duration); access rights (communication ports); fault response; operating mode (in a range of given values); and an API to set and get the corresponding values.

All processes are created statically at system start-up. Processes execute concurrently (FIFO within priorities). Partitions communicate via ”sampling ports” and ”queuing ports”. Processes within a partition have four methods for communication. Three hierarchical levels of health monitoring: module, partition, process.

Joyce then discusses possible Ada implementation models of ARINC 653: mapping an Ada task to an ARINC process appears to be way too complicated; better would be to get the Ada runtime out of the way and build a direct binding to ARINC. Based on this reasoning, Joyce outlines a proposal for a pragma profile that facilitates mapping of Ada applications to ARINC 653:

```
pragma Profile (ARINC_653) ::=
 pragma Restrictions { Max_Tasks => 0,
   No_Allocators, No_Asynchronous_Control, No_Exceptions,
   No_Protected_Types, No_Synchronous_Control}
```

Arnaud Charlet challenges the exclusion of Ada exceptions, which in his knowledge are being used by Ada ARINC systems. Joyce sees no major issue in allowing Ada exceptions. A further addition to this list of restrictions regards the use of time delay, which current Ada ARINC systems seem not to use.

Andy Wellings notes at this point that it might perhaps be preferable to target a wider range than just ARINC by identifying the common set of Ada runtime components (exception, tasks, etc.) and then defining a base profile for it.

Bruce Lewis briefly addresses the issue of partition scheduling. The base option is static non-preemptive partitioning, which makes it difficult to find a feasible schedule, and carries all of the known difficulties with traditional static cyclic scheduling of processes. An increasingly considered alternative is preemptive fixed priority partitioning. The question that Bruce throws at the group is how Ada could help the take up of the latter option, perhaps by providing mechanisms for partition scheduling and partition communications other than those in Annex D. The issue is challenging, but the group finds that it unfortunately is not in the mainstream of the workshop. Hence the discussion of it is not pursued further.

The session chair then summarises the status of the discussion, listing the possible options for the way ahead:

1. no further change to the current Ada scheduling model
2. priority-specific policies, including Round Robin as proposed in [1]
3. application-defined scheduling, with in all cases, the stack resource policy (preference control)
4. EDF via a specific scheme (as opposed to 3.)
5. Generalised dispatching attribute scheme (attached to the scheduling policy to executed to any change to each attribute)
6. ARINC profile (specifically, as opposed to wider-spectrum than just ARINC)
7. Budget control (with the now-available execution time hooks)
8. Partition-level scheduling

The chair calls the group to a straw vote on some of these options, the results of which are summarised in the table below:

The vote on option 8 means that the group feels it does have neither the energy nor the deep understanding to pursue the issue further, but it does not oppose to it anyway. The vote on options 5 and 7 is deferred because it follows from the decision made on options 1-4.

Option	For	Against	Abstained
1	0	16	3
2	17	0	2
3	12	2	6
4	11	0	7
6	10	2	6
8	0	0	18

Option 2 The group fully carries option 2. Alan Burns however notes that, in order to allow a dispatching policy that requires side support to operate efficiently (like e.g.: EDF with Constant Bandwidth Server) the proposed syntax in the paper needs to have curly brackets instead of squared ones, as follows:

```
pragma Partition_Dispatching (
 Policy_Identifier, {,Policy_Argument_Definition});
```

The type of the Quantum parameter to pragma Priority_Policy needs to be determined, so that it can be expressed in a configuration pragma. The group agrees (15-0-3) that the Round Robin proposal in the paper should be turned into an AI to be submitted by Alan Burns to the ARG.

Options 3 and 5 Andy Wellings notes that the scheduling events listed in Michael Gonzalez-Harbour's proposal do not quite completely map with the Ada definitions of those, which is much richer than the tasking model implied by Michael's POSIX-based model. Arnaud Charlet compounds this argument by referring to the list of task-related events captured by GNULLI, which also is significantly larger. Michael undertakes to resubmit a refined proposal for later in the workshop.

A reduced version of option 3 is also discussed, which is limited to the provision of the stack resource policy for preference control. Alan Burns undertakes to prepare a position on this reduced option, and to submit it to the group later in the week.

Option 6 Joyce undertakes to have an AI proposal on this option ready for presentation later in this workshop.

Option 7 Andy Wellings presents a Group_Budgets child extension to package Ada.Real_Time.Execution_Time, which would allow budget control for the implementation of Scheduling Servers. The package specification given below is slightly different from that presented (as it was further refined during and after the workshop):

```
with Ada.Task_Identification;
package Ada.Real_Time.Execution_Time.Group_Budgets is

  type Group_Budget is limited private;

  type Handler is access protected procedure(
                 GB : in out Group_Budget);

  type Task_Group is array(Positive range <>) of
                 Ada.Task_Identification.Task_ID;

  Min_Handler_Ceiling : constant System.Any_Priority :=
                 <Implementation Defined>;

  procedure Set_Handler(GB: in out Group_Budget;
                        H : Handler);
  function Get_Handler(GB: Group_Budget)
    return Handler;

  procedure Replenish (GB: in out Group_Budget;
                       To : Time_Span);
  procedure Add(GT: in out Group_Budget;
                Interval : Time_Span);

  procedure Add_Task(GB: in out Group_Budget;
            T : Ada.Task_Identification.Task_ID);
  procedure Remove_Task(GB: in out Group_Budget;
            T : Ada.Task_Identification.Task_ID);
  function Is_Member(GB: Group_Budget;
            T : Ada.Task_Identification.Task_ID)
    return Boolean;
  function Is_Member(
             T : Ada.Task_Identification.Task_ID)
    return Boolean;

  function Budget_Has_Expired(GB: Group_Budget)
    return Boolean;
  function Budget_Remaining(GB: Group_Budget)
    return Time_Span;

  function Members(GB: Group_Budget)
    return Task_Group;

  Group_Budget_Error : exception;
private
    --  not specified by the language
end Ada.Real_Time.Execution_Time.Group_Budgets;
```

Andy's approach decouples the delivery of the budget expiry notification from the decision of what actions to undertake to treat the event (e.g.: stop the overrunning task). Michael Gonzalez-Harbour notes that the model does not easily scale up to multiprocessor environments, and he therefore suggests that the effects in that case should be left undefined. Andy undertakes to work further on this option so as to present a proposal to the group in the "Generating New AIs" session, later in the workshop (cf. the relevant rapporteur's report in these proceedings for details on Andy's proposal and on the ensuing discussion).

References

[1] A. Burns, M. González-Harbour, and A. J. Wellings. A Round Robin Scheduling Policy for Ada. In J.-P. Rosen and A. Strohmeier, editors, *Reliable Software Technologies — Ada-Europe 2003*, number 2655 in LNCS, pages 334–343. Springer, 2003.