Report of Session: Generating New AIs

Chair: Alan Burns
Rapporteur: Tullio Vardanega

1 Introduction

The chair opens the session summarising the status of the real-time AIs that the group has addressed over the week, whether by refinement of the current version or by discussion of a new proposal:

AI	*Title*	*Status*
00307	Execution Time Clocks	Revised, for discussion
00327	Dynamic Ceiling Priorities	Revised, to ARG
new	Protected/task interfaces	For discussion
new	Round robin scheduling	Confirmed, to ARG
new	Stack resource protocol	Important, for discussion
new	Application defined scheduling	Important, for discussion
new	EDF within priority band	Important, for discussion
new	Group budget control	Important, for discussion
new	ARINC profile	For discussion

Each of the above AIs marked "for discussion" will be championed in the session by an appointed member of the group.

2 Protected Interfaces

Andy Wellings champions this possible AI, whose initial idea was sketched by Tucker Taft in section 7 of [2].

Currently we cannot separate, in a single PO, entries that functionally belong to distinct callers, which appears to be a limitation to the language expressive power as well as to visibility control.

To illustrate the idea Andy outlines a small example that shows how protected interfaces would allow both separation of and access to (protected) entries. The case for the latter feature stems from considering how greatly beneficial to the language expressive power was the addition of access to protected procedures. Any argues that the users are now mature to make effective use of access to (protected) entries. Andy further notes that, in his view, an interface should not be allowed to extend an interface.

The group votes overwhelmingly in favour of encouraging Tucker Taft and the ARG to continue to work on protected and, possibly task, interfaces.

3 Stack Resource Protocol

Alan Burns presents the initial idea on a new locking policy based on the Stack Resource Protocol [1]. This feature is viewed as the key enabling factor to the introduction of other dispatching policies than fixed priority in the language.

Alan notes that the correct operation of the current Ceiling_Locking policy in the face of calls from an equal (ceiling) priority level in single-processor environments assumes the FIFO_within_Priorities dispatching policy to prevent the occurrence of any hazard to mutual exclusion. (This is in fact the reason why the ALRM requires that the use of ceiling locking be accompanied by FIFO within priority dispatching.) To extend the provision of this assurance to the general case, the following semantics is proposed, which is independent of the dispatching policy in force:

- when a task calls a protected operation with ceiling priority equal to the calling task's priority, a check is made that its preemption level is not higher than the corresponding protected object, Program_Error being raised if the check fails.

The feature should take the form of new locking policy by the name: Preemption_Levels_Within_Ceiling_Locking.

A new configuration pragma, pragma Preemption_Level should be defined, which applies to tasks and protected objects, the effect of which would vary in accord with the dispatching policy in force.

The range of preemption levels should be expected to be much wider than the priority range. The actual range though should be implementation defined, with an implementation-defined specified minimum, which could be placed at the bottom of package System.

4 Application-defined Scheduling

Michael Gonzalez-Harbour reports on the refinements made to the proposal as a result of earlier discussion in this workshop: (1) the application scheduler need not be a task; (2) the list of scheduling event continues to include the sole events of interest, with the addition of the ABORT event,

and with the implication that multiple runtime events would then map to one and the same scheduling event.

Andy Wellings enquires what level of overhead should be expected from the runtime to make sure not to apply unneeded control actions to tasks that are not scheduled by application schedulers. Mario Aldea reports that his prototype implementation shows negligible overhead in this respect.

Niklas Holsti suggests using tagged objects for scheduling events, which should be "with"-ed in the scheduler internal data, One primitive operation should be defined for each such event, which could then be invoked by the runtime instead of by the application.

Michael then presents the revised API for this AI.

Some discussion arises on the need for private type Scheduling_Actions, which the application scheduler uses to build its list of actions for the runtime to execute, to be in-out to all the related procedures. Michael argues that the presence of this type makes the semantics of all such procedures clearer.

After Niklas' suggestion, type scheduling_parameters is now abstract tagged null, with no primitive operation attached to it.

At this point, the proposal includes:

- a pragma Application_Scheduler that takes three arguments: (1) an object of scheduler type; (2) the priority level at which this scheduler is attached; (3) the locking policy optionally required for the scheduler, which may be null
- a pragma Application_Defined_Sched_Parameters that takes an object of type Scheduling_Parameters.

Michael suggests a further improvement to the API, which the group approved, whereby a Timer_Expiration protected type is defined within the Application_Scheduler, with an Announce procedure that can be passed as a parameter to the Arm and Add procedures. (For details, the reader should refer to Michael's paper in these proceedings.)

5 Group Budget Control

Andy Wellings outlines a possible AI, intentionally limited to uniprocessor environments, which allows a group of tasks to share budget time, so as to facilitate the implementation of specific server policies. Tasks may be added to and removed from the group.

This facility would take the form of a child to the execution time budget package: Execution_Time.Group_Timers.

Andy presents an example of use of this facility in conjunction with the Timing_Events feature (cf. AI-297), for the implementation of the Deferrable Server.

Two alternative forms of notification for budget exhaustion are explored: one that simply reports it, letting the server find out the task group(s) concerned; the other, which specifies the task group for which this has happened.

The decision on the preferred form of notification has a ramification that concerns the possible handling of the tasks whose budget has expired. Three options are considered:

1. Suspend tasks when their group budget zeroes
2. Notify server when group budget zeroes: control on the task group(s) stays in the proposed predefined package
3. Notify server without assuming anything: the proposed package does not control the task groups for which this has happened.

The group supports option 3, which clearly is the simplest one to support and then expresses overwhelming support for Andy to continue to work on this AI.

6 ARINC 653 Profile

Joyce Tokar presents the text of an AI, based on the Ravenscar one, which defines a new ARINC_653_Processes parameter for pragma Profile as follows:

```
pragma Restrictions(
     Max_Tasks => 0,
     No_Asynchronous_Control,
     No_Protected_Types,
     No_Synchronous);
```

The intent of this profile is to allow an Ada runtime with no scheduling in it, at all, so that, in the case in question, it would all be taken care of by ARINC.

One issue on the naming of the profile is raised by the group, some members of which would rather call it otherwise, for instance "minimum runtime". A short discussion however rapidly shows that there is no clear-cut definition of the "minimum runtime" notion for the term to denote a standard profile.

Then the question is raised as to whether the proposed one be the right set of restrictions. The discussion on this point is inconclusive, though, so that the group encourages Joyce to consolidate this proposal into an AI and to push it forward.

7 AI-00307 Execution Time Clock

Andy Wellings and Michael Gonzalez-Harbour present the result of their refinement work on AI-307, which took stock of the group discussion held earlier in this workshop.

The latest changes are minor and all viewed with favour by the group:

- Timer has become a limited private type instead of a protected type;

- the Notify procedure has become an Announce (or Handler) access type to a protected procedure that takes an access to a Timer as an input parameter;
- a Min_Announce_Ceiling implementation-defined value informs the user about the minimum level that the ceiling priority of the protected object whose procedure is referenced by Announce (or Handler) should have.

Andy notes a possible stylistic clash with the Timing_Events package, where all API procedures take Timing_Event parameters and no access value, whereas in the AI under examination API procedures such as Announce (or Handler) currently take an access value.

The group's decision on this issue is to align with the style in the Timing_Events package and hence remove the access type to Timer and pass Timer objects as in-out parameters to Announce (or Handler).

8 EDF

Alan Burns presents the outlined of a predefined package for EDF_Support.

This package will cause all tasks to have an additional attribute named Deadline defined as follows:

```
package Deadlines is new Ada.Task_Attributes (
   Ada.Real_Time.Time, Ada.Real_Time.Time_Last);
```

The initial value of this attribute would be set by a configuration pragma in the task specification, via pragma Deadline <some_time>, in the absence of which the task would get a Time_Last default deadline attribute via an implicit call to Deadlines.Set_Value(Some_Time).Function Deadlines.Value would return the current task deadline.

Andy Wellings poses the question whether support for EDF dispatching would imply the need for deadline inheritance, but, after some discussion, there appeared to be no obvious reason for it.

Michael Gonzalez-Harbour suggests that each additional task attributes (currently deadline, in the future, perhaps, budget), which have an impact on scheduling and have to be passed to the runtime (hence, initially by a pragma) should be built on top of an attribute class type hierarchy.The suggestion is looked at with favour but it needs more thinking.

The discussion closes with the understanding that Alan Burns will continue to consolidate this possible AI, submit it to the ARG and keep the group informed as to its progress.

References

[1] T. Baker. Stack-Based Scheduling of Realtime Processes. *Real-Time Systems*, 3(1):67–99, 1991.

[2] S.T Taft. Object-Oriented Programming Enhancements in Ada 200Y. *Ada User Journal*, 24(2):119–127, 2003.