Report of Session:
Language Changes for Scheduling, Modeling and Analysis

chair: Andy Wellings
rapporteur: Tullio Vardanega

Introduction

The session included four position papers, three of which share a common theme: how to permit more flexible scheduling in Ada 95.

With this setting, the goal of the session was to analyse the proposed language extensions and to determine those, which IRTAW should carry.

The chair clarified that the supported proposals should be turned into Ada Issues (AIs) by selected representatives of IRTAW-11 and submitted for consideration by the Ada Rapporteur Group (ARG) as part of the on-going language revision process.

The topics to cover in the session ranged:

- mechanisms for flexible scheduling
  - allowing priority ceiling changes
  - accessing delay queues
  - execution time clocks
  - support for sporadic/deferrable server
- application-defined schedulers
- Ada compilation to hardware.

Dynamic Ceilings

The notion of dynamic ceilings is not new to the Ada real-time community, which has always viewed it as an especially convenient means to support mode change situations.

The group recalled that, in the making of the Ada 9X process, the feature was considered too complex for the language. Yet, this is no longer the case as mature analysis techniques have been developed, which permit to comfortably analyse mode change situations.

On this premise, Jorge Real illustrated the latest form of the proposal for dynamic ceilings, which previous IRTAWs encouraged him to produce.

The highlight of the proposed language enhancement is that the ceiling change would be executed by a *predefined* Set_Ceiling protected procedure, whose use would follow the Ceiling_Locking rules.

The ceiling change would take effect when the caller leaves the protected object. If, after the change, the priorities of tasks (re)queued in entries of the protected object happened to be higher than the new ceiling, then Program_Error should be raised, as per ARM D.5(11), to signify that this situation would be a problem of the application, not of the language.

The discussion considered whether Set_Ceiling could be a " ' attribute", so that for a protected object named PO and a ceiling priority C, we would write: PO'Set_Ceiling(C).

Further discussion determined that the presence of the attribute should be optional and explictly required by use of a pragma Adjustable_Ceiling.

At this point of the discussion, the chair asked the group whether they felt this would be the right semantic model. With virtually unanimous consensus (17 for, 0 against), the group carried the proposal.

The chair then turned the discussion on the syntactic aspects of the model. The group noted that the proposed syntax would require the caller to know the protected object name, while it would be important to allow anonymous invocations of the ceiling change procedure.

In order to permit this, the group first conjectured that the Set_Ceiling procedure could have a 'Access attribute. This hypothesis, however, turned out to conflict with conformance rule 6.3.1(9), which requires attributes that are subprograms to be intrinsic and, therefore, not to be able to hold the protection lock of a protected procedure.

The chair briefly recalled that the alternative of using the same model as that of dynamic priorities for tasks was considered and discarded at previous IRTAWs on the grounds of requiring the introduction of protected object identities, too heavy a language change.

Overall, the group showed clear consensus (15 for, 0 against) on Set_Ceiling being a protected procedure. The discussion also revealed the need for a Get_Ceiling protected function.

The chair then outlined a solution based on the following syntax, which would make the change executable, as

```
type General_Set_Ceiling is access protected procedure (P: Priority);
protected types My_PO is
   -- ...
   procedure Set_My_Ceiling(X : Priority) is
   begin
      My_PO'Set_Ceiling(X);
   end Set_My_Ceiling;
end My_PO;
```

a protected operation, only by a protected object on itself and would also allow anonymous changes (via General_Set_Ceiling) and solely for those protected objects that explicitly permit it.

The implication of this proposal, however, is that the change would take effect only when the caller (the changer) leaves the protected object, which would thus require the implementation to support 2 ceilings (the one to become and the one still being).

The proposal, as shaped by the discussion, obtained unanimous consensus from the group (19 for, 0 against) and, at Juan Antonio de la Puente's suggestion, it was agreed that it should also include a metric requirement similar to D.5(14).

Jorge Real was nominated lead for the preparation of an AI based on the group deliberation, with the assistance of Andy Wellings and Alan Burns.

Accessing Delay Queues

Alan Burns outlined the proposal to the group. The proposal wants to allow the programmer to *asynchronously* change the task priority, so that a task may come off the delay queue at a priority different than the one it had when it was first enqueued.

The idea behind the proposal is to associate a procedure to a future time event (whether relative or absolute) in very much the same way as we can presently associate a procedure to the arrival of an external interrupt. In practice, the proposal is for a new child package of Ada.Real_Time, with the following specification:

```
package Ada.Real_Time.Timing_Events is
   type Parameterless_Handler is access protected procedure;
   procedure Signal(At_Time : Time;
                    Handler : Parameterless_Handler);
   procedure Signal(In_Time : Time_Span;
                    Handler : Parameterless_Handler);
end Ada.Real_Time.Timing_Events;
```

By use of this feature, the application could attach (chains of) actions to *one-off* time events, without exposing the reserved interrupts that the runtime attaches to clock interrupts, with the attachment being discarded after the invocation.

The group showed clear consensus for the proposal (17 for, 0 against) and Michael Gonzalez-Harbour encouraged the group to look at whether the proposal could be implemented on top of POSIX.

Alan Burns then tabled the open issues to be addressed prior to turning the proposal into an AI:

- Would Signal be a potentially suspending operation, which could not be called from a handler? The group rejected this option, wanting Signal to be callable from handlers.
- Would order matter upon multiple attachments to the same time event? The discussion showed that FIFO would be adequate, as all the attached handlers would be executed in the corresponding ceiling order anyway.
- What should happen if At_Time denoted a time in the past? Consensus was to have the handler execute immediately, for symmetry with the effect of invoking a delay until to a time in the past from within a task.
- What should happen if the ceiling of the attached handler was inferior to 'Last? The group preference was for the program to become erroneous.

The proposal, completed with the above resolutions, obtained clear consensus from the group (12 for, 0 against).

Alan Burns and Andy Wellings were nominated leads for the corresponding AI. Stephen Michell, Tullio Vardanega and Juan Zamorano volunteered to assist. This subgroup would have to consider whether to permit cancellation of attachments

Execution-Time Clocks

Michael Gonzalez-Harbour championed the newest proposal, which integrates CPU-time clocks into the language and provides for one clock (with a corresponding Clock_Id per activity needing to be timed.

The group felt that the new proposal looks considerably nicer than the one presented at the last IRTAW, but it requires more changes (to the compiler and to the runtime). Michael argued that the experience of Javier Miranda's prototype implementation based on custom GNAT modifications shows that the required changes would be modest.

The group observed that the feature, which has been wanted by the IRTAW community for a long time, is specified as optional in POSIX as well as in the RTSJ. Yet, the issue of making this feature optional to Ada appeared to be difficult, and the relevant discussion was thus aborted.

The notable advantage of the previous approach over the new one appears to be simplicity of implementation, in that it would cause only one change to the language: the prohibition to invoke a delay until on one's own clock time. Nonetheless, the reference implementations developed at the University of Cantabria seem to indicate that the new approach would incur a very small overhead compared to the previous one.

The chair proposed that the strong and long-standing demand by the group for execution-time clocks could take two flavours:

- prefer integration (new) but accept library package (old)
- prefer library package (old) but accept integration (new).

The first option obtained 7 votes for and 0 against; the second 3 for and 0 against. At this point, the chair summarised that there was clear consensus that the group wanted an AI on execution-time clocks, but the participants felt they were not sufficiently informed to make a strong recommendation on either options.

Michael Gonzalez-Harbour was therefore asked to circulate to the group an updated report on his two proposals (the new and the old one), augmented with the discussion of the respective pros and cons, before initiating the write-up of the AI. Andy Wellings volunteered to help out with the AI. At the request of Joyce Tokar, Michael's report should also be forwarded also to Mike Kamrad, who submitted an ARTEWG proposal to the ARG on the subject.

Application-defined Schedulers via Scheduling Events

Michael Gonzalez-Harbour presented the status of his work in progress aimed at the definition of Ada bindings to the equivalent POSIX services and facilities.

The motivation for the work is to allow applications to define their own scheduling policies, by the provision of suitable hooks to the runtime, so as to allow a large variety of policies without having to standardise on any (or all) of them, which is obviously neither practical nor realistic.

The proposal would allow multiple scheduling policies to co-exist. It would also permit isolation of critical tasks from the behaviour of application-level schedulers. Furthermore, two separate virtual address spaces would be permitted (but not mandated) for the sake of protection and optimisation: the usual user address space for regular and application-scheduled tasks; and the scheduler address space, for application scheduler tasks.

Michael reported that a model POSIX-based implementation was developed at the University of Cantabria using MarteOS, which showed acceptable performance and considerable flexibility. Michael indicated that the next step of this work would be to consider integration into the Ada language, which is anticipated to present considerable challenges, especially with respect to the role of protected objects in the model.

The group showed some interest in seeing further development of the proposal, yet a considerable proportion of the participants felt that they would need more consolidated details before forming a definite position on the proposal.

Compiling Ada to Hardware

This fraction of the session was devoted to the presentation of an opportunity of future interest for Ada, presented by Neil Audsley.

The direction would be one of cooperation of Ada with domain-specific languages like VHDL. The mode of cooperation would primarily be one of co-design, co-verification and co-simulation.

At present there are significant differences between Ada and VHDL, which, as the implementation medium for 'system-on-a-chip' (SoC), non-von Neumann architectures, i.e. circuits, offers: (i) true parallelism (hence no need for priorities); (ii) direct connection to devices (hence advanced representation clauses; and a whole new brand of interrupts in the form of FPGA-level events); (iii) reconfigurability.

To begin to investigate the issue, a SPARK/Ravenscar Ada to VHDL compiler was developed at the University of York, which seeks an implementation-independent compilation of the language.

Neil argued that a strategic opportunity for Ada may result from SoC becoming intensively programmable by software-equivalent means, in that the need may arise for proper software implementation languages to play an important role in that arena.

Application-level Implementation of Sporadic Server Technologies

The final slot of the session was devoted to ascertaining whether the combined effect of the carried AIs on CPU time accounting and access to delay queues, would indeed allow the implementation of sporadic server and deferrable server technologies. The answer to this was affirmative.

It was also noted that the CPU-time-accounting AI is by far the most important improvement required, without which the above assertion would fail.

On this grounds, the IRTAW group would be prepared to prioritise the 2 AIs so that the ARG can determine how best to proceed with them.