Wellings, A.; Real Sáez, JV. (2015). Session summary: Language abstractions. Ada Letters. 35(1):102-104. doi:10.1145/2870544.2870558.

Session Summary: Language Abstractions

Andy Wellings, *Session Chair*
University of York, UK
andy.wellings@york.ac.uk

Jorge Real, *Rapporteur*
Universitat Politècnica de València, Spain
jorge@disca.upv.es

1 Introduction

This session reviewed the support for two classes of language abstractions for which position papers had been submitted. The workshop first considered the ability to monitor and control the affinity of protected handlers for both interrupts and timing events, based on the ideas and results presented in papers [5] and [3]. Then we considered the incorporation of the concept of cyclic tasks directly into the language, according to the proposal in [1].

2 Support for event-handler affinity

The session chair set the context of the issues at hand. There is a lack of support in Ada for controlling the affinity of event handlers (namely timing-event, execution-time and group-budget handlers) [5]. Similarly, interrupt handlers cannot currently have their dispatching domains set either.

Implementation of the required support for these facilities is highly dependent on the type of the underlying execution platform. Andy identified four possible scenarios:

Bare machine There is no operating system as such: the Ada run-time environment has full control over interrupts and devices.

Real-time kernel In this case, the kernel, run-time and application all run in the same address space, with same privileges.

Real-time kernel with loadable kernel modules Here all or part of the application can run as a kernel module, hence sharing address space and privileges with the kernel.

User-mode application on top of a real-time kernel The Ada run-time environment and the application run in separate address space to the kernel and run with limited privileges.

The chances for a particular implementation to efficiently support event-handler affinity will very much depend on the ability to control the underlying hardware resources. This support is generally limited when the run-time relies on an existing OS kernel, but on real-time embedded systems, programmers should have control over (or at least knowledge about) which processor will execute which handlers, so that the associated interference can be bound and properly considered in the schedulability analysis.

The Ada Real-Time Annex philosophy is to allow the specification of packages which, since the Annex is optional, may or may not be supported on all systems. Examples of these are asynchronous task control, EDF scheduling or group budgets. So encouraging development and implementation of essential real-time programming abstractions seems like a reasonable goal.

So the core issue is whether or not the language should define different acceptable implementation approaches so that there is more commonality on the various execution platforms. This would probably only apply to low-level features, such as interrupts, event handlers or representation aspects.

Currently a function exists in package `Ada.Interrupts` to obtain the CPU to which an interrupt is connected. The proposal from Wellings and Burns [5] suggests that, in order to control the execution of interrupt handlers on a multiprocessor system, this package should also provide mechanisms to set the dispatching domain (and potentially an individual processor) for an interrupt. A standard exception should be raised if the operation is not supported on a particular platform. In addition, the package `Ada.Interrupts.Names` should declare standard names for all the reserved interrupts required by the Ada run-time system. For example, clock interrupts that service timing events and those that allow tasks to be released when a delay expires (either relative or absolute). The implementation should also document which reserved interrupts result in which of the event handlers being executed. And for those implementations where extra tasks are introduced to execute the event handlers, those tasks should have the same affinity as the associated interrupt.

The position paper submitted by Sáez et al [3] showed the implementation of timing-event affinities for a particular implementation of Ada on top of Linux. In that implementation, the run-time introduces tasks to execute timing-event handlers. Even though this is not in line with the spirit of timing event handlers, which should ideally be executed by the clock interrupt handler, the proposed experimental implementation takes advantage of this by using one task per processor (using task affinities) so that they execute on a known CPU. In this manner, there is a degree of control about where (on which CPU) the interference of the handler will occur. The paper [3] gives an alternative implementation for `Ada.Real_Time.Timing_Events` to support this feature.

2.1 Summary of Workshop position

The Workshop's conclusions after discussion of this part of the session were:

- Ada interrupt handlers should be able to have dispatching domain (or individual processor) set, with raising a standard exception if this feature is not supportable by the underlying platform.
- It was considered that reserved interrupts could still be invisible, but documentation should indicate which dispatching domain is in charge of executing their handlers. There was however no consensus around this aspect.
- It was agreed that Ada should include mechanisms to set the dispatching domain or processor where a timing-event handler executes. There was no consensus however regarding execution-time and group-budget handlers.
- We need to work on the motivation and vision of the issue, towards the production of a related Ada Issue for its consideration by the Ada Rapporteur Group.

3 Support for the concept of cyclic tasks

Cyclic tasks (both periodic and aperiodic) are fundamental patterns in real-time and embedded systems. There are however no abstractions in Ada to model them, and therefore programmers need to care not only about the logic of their tasks, but also about their particular release mechanisms. There have been contributions in this regard in the last few years. Previous editions of this workshop have considered different versions of a library of standard real-time utilities to capture these patterns. An initial version

was proposed in [6], which was later extended to cover multi-moded systems [2]. More recent work, in the context of the Ada-Europe International Conference on Reliable Software Technologies, elaborated on this basis to adapt the library to multiprocessor systems [4]. But until today, the IRTAW has been wary of suggesting language changes in this regard.

The second half of this session aimed at revisiting this language constraint, based on the position paper by Patrick Bernardi [1]. In that paper, Bernardi proposes a cyclic task syntax, which can be either time- or event-triggered. Cyclic tasks may also specify how to handle deadline-miss and budget-exhaustion events by means of exceptions.

3.1 Summary of Workshop position

Upon examination of Bernardi's proposal, the workshop agreed that, after some rework and refinement, this could be a good starting point for a proposal to the Ada Rapporteur Group. Points in favour are the the proposal makes more about the cyclic nature of real-time systems, takes advantage of implementation experience, and naturally resolves initialisation issues, since cyclic tasks can be released after their initialisation phase. The workshop, however, identified several outstanding issues that need clarification or further consideration:

- The proposal suggests that a deadline-miss exception should be raised in a tardy task. This would be an asynchronous exception, which is not supported in Ada.
- The proposal does not cover the handling of minimum inter-arrival time violations for sporadic tasks.
- Need to clarify whether deadlines relate to the design's release time or to the actual release time of the tasks.
- Need to revisit and refine the syntax for releasing an aperiodic or sporadic task.
- Need to consider the situation of a program trying to release a task with periodic behaviour.
- In its current state, the proposal does not allow to identify that a task is cyclic from its specification: one has to read the body.
- Need to assess the flexibility of the model, making sure that all aspects are covered.

The workshop agreed on the need to generate an Ada Issue from a revised version of [1] covering these open issues.

References

[1] P. Bernardi. Incorporating Cyclic Task Behaviour into Ada Tasks. *Ada Letters*, This issue, 2015.

[2] J. Real and A. Crespo. Incorporating Operating Modes to an Ada Real-Time Framework. *Ada Letters*, 30(1):73–85, April 2010.

[3] S. Sáez, J. Real, and A. Crespo. Implementation of Timing-Event Affinities in Ada/Linux. *Ada Letters*, This issue, 2015.

[4] S. Sáez, S. Terrasa, and A. Crespo. A Real-Time Framework for Multiprocessor Platforms Using Ada 2012. In S. Romanovsky and T. Vardanega, editors, *16th International Conference on Reliable Software technologies – Ada-Europe 2011*, volume 6652. Springer, June 2011.

[5] A. Wellings and A. Burns. Interrupts, Timing Events and Dispatching Domains. *Ada Letters*, This issue, 2015.

[6] A. J. Wellings and A. Burns. A Framework for Real-Time Utilities for Ada 2005. *Ada Letters*, XXVII(2), August 2007.