Toward Extensions to the Ravenscar Profile

P. Rogers, J. Ruiz, T. Gingold

AdaCore

{rogers|ruiz|gingold}@adacore.com

Abstract. We describe an on-going effort to identify a set of enhancements to the Ravenscar profile, intended primarily for applications in the real-time systems domain, but perhaps also applicable to the other domains supported by Ravenscar as currently defined. In addition to the technical issues we also include matters of cost, leading to the question of whether a new profile is appropriate.

Keywords: Ada, Ravenscar Profile, real-time systems, run-time library

1. Introduction

The Ravenscar Profile has proven to be a very successful definition of a powerful tasking subset, allowing the use of concurrency in projects that would have otherwise been impossible. The profile is a significant advance in software technology and a major strength of the Ada language.

The profile is not intended to address all conceivable uses for Ada tasking. In particular, Ravenscar is intended to support four distinct application domains:

- hard real-time applications requiring predictability,
- safety-critical systems requiring formal, stringent certification,
- high-integrity applications requiring formal static analysis and verification, and
- embedded applications requiring both a small memory footprint and low execution overhead.

The profile is necessarily a subset of full Ada tasking because of the wide range and demanding nature of the analyses involved. Those tasking constructs or functionality that preclude analysis at the source level, or analysis of the tasking portion of the underlying run-time library, are disallowed. Complexity is a primary issue but is also a factor in memory (code) footprint and execution overhead. For example, constructs with distributed costs, such as abort statements, both complicate the run-time library implementation and impose size and performance penalties even when not used in the application.

A loss of expressive power, however, inevitably results from the elimination or restriction of tasking features. This loss can lead to the re-introduction of complexity at the application level when developers "implement" some of the prohibited language-defined functionality. We say they "implement" it because this code appears at the application level, as a form of abstraction inversion, using those constructs allowed by the profile. Obviously the reintroduced complexity must be acceptable for the domain in question, and for those domains involving stringent analyses it might very well be rejected.

We believe that some of the lost expressive power could be recovered, thereby preventing the concomitants of abstraction inversion. In particular, some of the tasking restrictions could be relaxed because we expect that their implementations can be realized without introducing unpredictability or undue complexity into the underlying run-time library. Indeed, the fact that application developers can effectively implement some extensions implies that the run-time library could do so too.

Whether a run-time library supporting extensions would remain appropriate for all four of the intended Ravenscar application domains is an open question. An enhanced profile will remain applicable to hard real-time systems if the extensions can be implemented without sacrificing predictability. We believe that will be possible. The issue, then, is the effect of complexity on the analyses required by the other domains. For example, would the enhanced run-time library remain amenable to certification?

If support for the four domains is possible then the enhancements could constitute a revision to Ravenscar, rather than a new profile altogether. Otherwise, a new profile is appropriate, specific to the real-time systems domain (because we believe predictability can be maintained). It is worth noting that extensions to an existing

profile are more likely to be accepted by the ARG than a wholly new profile. Nevertheless, a separate profile would maintain the consistency and simplicity of the current Ravenscar profile, especially if the proposed changes constitute a different programming model. We do not believe the proposed enhancements represent a different model.

In addition to technical matters, business issues also determine whether a given set of extensions is feasible. If an additional, distinct run-time library is required, vendors are not likely to implement it because of the considerable recurring support costs (over and above the initial implementation cost).

In the remaining sections of this paper we expand on the motivations and issues introduced above, introduce a set of requirements and proposed extensions, and discuss their possible implementations.

2. Motivation

For those applications requiring primarily predictability, some of the restrictions in Ravenscar result in a noticeable loss of expressive power. When applications developers then implement some of the precluded functionality, they introduce complexity, in effect moving it from the run-time library (were it implemented there) to the application. We suspect this loss may be avoidable because we believe we can implement some of the missing features in the run-time library without loss of predictability, and with no more complexity than that of an implementation at the application level using profile-approved constructs. Presumably a degree of complexity that is acceptable at the application level would be acceptable at the run-time library level instead, for applications in this domain.

The original Ravenscar definition was created prior to revisions of the language that removed some sources of unpredictability. For example, with the Real-Time Systems Annex in force, there is no longer any question of which protected entry would be selected when multiple entries are open in a given protected object. Therefore, we may be able to relax some of the profile's restrictions, without loss of predictability.

3. Requirements

3.1 Maintain timing predictability

Any enhancements to the profile must maintain the timing predictability of the underlying run-time library implementation. Preserving the ability to perform schedulability analysis for applications in the real-time systems domain is key.

3.2 Maintain memory (data) size predictability

Any enhancements to the profile must maintain the storage predictability of the underlying run-time library implementation. This will require the user to statically specify capacities for certain enhancements so that fixed-size data structures are possible within the run-time library implementation.

3.3 Minimize complexity

Any enhancement requiring complexity that precludes the analyses in any of domains will either be rejected, or will result in this entire set of enhancements being proposed as a new profile.

3.4 Maintain space and speed efficiency

The current Ravenscar profile can be implemented with both low timing overhead and a small code memory footprint. Those characteristics must be maintained.

3.5 Do not preclude "original" Ravenscar usage

Any enhancements must allow the current (Ada 2012) definition of Ravenscar to be used. The proposed enhancements are enabled by default, presumably, but must not prevent the user from restricting the subset to that of the current profile definition.

4. Extensions Under Consideration

The following is a list of extensions we are going to evaluate. These enhancements are those considered most important for addressing the loss of expressive power. Other extensions, although conceivable, are not included because they are not so central to the language. For example, Ada.Task_Attributes could, with some restrictions, be efficiently and predictably supported.

4.1 Protected entry queueing

We propose removing the profile's specification of Max_Entry_Queue_Length so that multiple callers can be queued on any given protected entry. The pragma Queuing_Policy would be applied automatically, with policy Priority_Queuing specified.

Every protected object containing a protected entry declaration would be required, for each entry, to specify the capacity of the corresponding queue. The value specified would be required to be static so that a fixed-size data structure could be used to represent the queue.

4.2 Multiple protected entries per protected object

We propose removing the profile's specification of Max_Protected_Entries, so that multiple entries per protected object are allowed. [besides, isn't this effectively the same as an entry family?]

Whenever multiple entries are open within a protected object and have callers queued, an entry will be chosen per the text of RM D.4 paragraph 12 (i.e., by priority of the callers or, if necessary, by the textual order of the entry declarations).

4.3 Asynchronous task control

We propose allowing use of the Ada.Asynchronous_Task_Control package, i.e., the removal of the application of No_Dependence to Ada.Asynchronous_Task_Control.

5. Approach

We will start from the Ravenscar Profile and add functionality, rather than starting from the Real-Time Systems Annex functionality and removing features. We will reject any given proposed enhancement if the underlying implementation would be so drastically different as to require a distinct, new run-time library.

We assume any additional data structures in the run-time library will be fixed-size, configured by the user.

Conclusions

References