Session Summary: Distribution and Real-Time

Chair: Michael González Harbour

Rapporteur: Luís Miguel Pinho

1. Objectives

The objective of this session was to discuss aspects related to the integration of real-time and distribution capabilities in Ada. Special focus was given to:

- the integration of the Real-Time and Distributed Systems Annexes;
- Ravenscar and Distributed Systems.

2. Introduction

In the previous edition of IRTAW, in a session devoted to "Distributed Ada and Real-Time" conclusion was drawn that it was necessary to further assess the integration of the real-time and the distributed capabilities of Ada. Moreover, it was considered that this integration could benefit from the, at the time emerging, Real-Time CORBA (RT-CORBA) standard.

Additionally, as the Ravenscar profile is gaining increasing acceptance, it is of paramount importance to address an appropriate model for distributed Ravenscar applications. It must be pointed out that the use of the Distributed Systems Annex (DSA) for high-integrity applications presents various challenges, thus requiring a more suitable model.

The session was essentially devoted to these two issues: integration of real-time and distribution capabilities of Ada, and models for Ravenscar distributed applications. The Session chairman selected the following topics for discussion, which reflected the contents of position papers and open issues from previous workshop editions:

- priorities of RPC handlers and messages;
- non-blocking APCs;
- documentation requirements;
- the option of a new annex;
- Ravenscar and the Distributed Systems Annex;
- inter-partition scheduling.

3. Integration of the Distributed and Real-Time Systems Annexes

The chairman, Michael González Harbour, initially presented the model of RT-CORBA, identifying features of the standard that could potentially be considered to a model for real-time distributed Ada applications. The RT-CORBA standard specifies:

- Interfaces for priority control of schedulable entities;
- Mechanisms for setting the priority of the server;
- Mechanisms to avoid or bound priority inversion;
- Management of resource allocation.

Michael considered that these features should also be addressed in the integration of the Real-Time and Distributed Systems Annexes, although the solutions might not be the same as those for the case of RT-CORBA.

Afterwards, Michael presented the proposal (by himself and José Javier Gutiérrez García) for extensions to the DSA to include support for distributed real-time applications. This proposal is described in more detail in the position paper "Towards a Real-Time Distributed Systems Annex in Ada".

3.1. Priorities of RPC handlers and messages

In real-time applications, in order to be able to predict and bound the response times to RPC requests, it is necessary to be able to specify the priorities at which the RPC handlers execute, and the priorities at which the messages are transmitted in the network (if supported). Thus, several extensions are provided in the proposal to allow this specification:

- A new global priority type, for representing a value with a global meaning in the distributed system. Appropriate mapping functions translate this global priority type to a value adequate for each CPU and network.
- Mechanisms for specifying the priority at which the RPC handlers start their execution, both initially and after servicing a RPC request.
- Mechanisms for specifying (at the client side) the priorities at which RPC requests are served in the server, as well as the messages' priorities in the network.
- Mechanisms for configuring the pool of RPC handlers, as well as more detailed semantics on the handling of pending RPC requests.

3.2. Non-blocking APC

The LRM in clause E.4(12) states that:

"The task executing a remote subprogram call blocks until the subprogram in the called partition returns, unless the call is asynchronous. For an asynchronous remote procedure call, the calling task can become ready before the procedure in the called partition returns."

And in E.4(17):

"All forms of remote subprogram calls are potentially blocking operations"

Thus, Michael stated that, for real-time applications, it would be desirable to have non-blocking Asynchronous RPC calls. Therefore, he introduced a new pragma (Non_Blocking) with the same behaviour as pragma Asynchronous, but requiring the call to be non-blocking.

3.3. Documentation issues

The proposal also addresses the issue of documentation requirements, both for the implementation of the compiler and runtime, and for the implementation of the PCS. Thus, Michael proposed that the implementation should document:

- additional tasks of the implementation (and their priorities);
- mechanism for their activation;
- worst-case execution times;
- additional messages for DSA-related protocols.

3.4. The option of a new annex

Michael considered that there was no need for a new annex, but instead these extensions should be embodied in the DSA. His opinion was that these extensions should be mandatory if both the Distributed and Real-Time Systems Annexes were supported by the implementation.

3.5. Discussion

Following Michael's presentation of the proposal, several issues were brought into the discussion:

- System.Priority can not be used as the global priority type, because it is not a pure type, thus it cannot be shared between partitions. Hence, a new type is required. However, the proposed type (Global_Priority) can not have implementation-defined semantics, because it would preclude the interconnection of different PCS implementations. Thus, it was concluded that this priority type should be specified in the proposal. A natural type, with at least 16 bits, was considered to be an appropriate solution.

- LRM clause E.5(26) specifies that:

  "The PCS is allowed to contain implementation-defined interfaces for explicit message passing, broadcasting, etc. Similarly, it is allowed to provide additional interfaces to query the state of some remote partition (given its partition ID) or of the PCS itself, to set timeouts and retry parameters, to get more detailed error status, etc. These additional interfaces should be provided in child packages of System.RPC."

  Therefore, conclusion was drawn that the proposed interfaces should be made available through child packages of System.RPC. In this way, compatibility with the LRM is achieved.

- The goal of pragma Initial_RPC_Handler_Priority is to specify both the initial value of the handler and the value to which the handler returns after servicing a request. Thus, the name Initial was not considered appropriate. The name Base_RPC_Handler_Priority was proposed, but it was confusing with tasks' base and active priorities. This issue remained open, but a more appropriate name should be devised.

- The default priority for RPC handlers (if not specified with the previous pragma), should follow the model of tasks and be in the middle of the System.Priority range, not System.Priority'Last.

- It was concluded that applications should not configure the PCS, thus the use of pragmas (for instance, to configure the pool of handlers) should be abandoned. The best solution would be constants defined in the PCS itself.

- Although the proposed extensions allow the client application to specify priorities in the network and in the server, it does not allow specifying the priority at which the client side of the PCS executes. However, that was considered a documentation issue. The implementation must document the way that the PCS handles the requests in the client side.

- In addition to RPC-related messages, there could be messages in the network related to the DSA protocols or to the runtime. It was considered that the implementation should document the semantics of such messages.

- There were some doubts on whether non-blocking Asynchronous RPC were needed. It was considered that this was not an important requirement.

- With the proposed model for specifying RPC messages priorities there is a loss of abstraction, because the higher-level model of RPC is lost by having to specify the priorities of the low-level messages. However, it was concluded that, in order to guarantee the goal of schedulable and analysable applications, distributed calls must be known by the application programmer, thus they can not be transparent.

- One issue that was raised was that the proposed extensions do not allow servers to reply with different priorities (for instance, depending on the outcome of the processing). Although it was considered that this would be interesting, no consensus was reached on whether it was important enough to consider it in the proposal.
- Some attendees felt that maybe the Real-Time extensions to the DSA should be abandoned and real-time distributed applications should use RT-CORBA instead. However, Michael counter-argumented stating that using these extensions in the DSA, real-time distributed applications would be more efficient and reliable than using RT-CORBA.

Endorsement:

The Workshop concluded that the extensions proposed were appropriate for real-time distributed applications, hence the group endorsed the presentation of the proposal (considering the issues addressed in the session) to the ARG, as an extension to the DSA.

In addition, an overwhelming majority of the participants approved a proposal to affirm that if these real-time extensions are not incorporated in the DSA, the RPC part of the Annex is of no use to the Real-Time community and, consequently, other solutions will be used.

4. Ravenscar and Distributed Systems

In the second part of the session, Andy Wellings and Neil Audsley presented the work carried out to assess the use of the DSA for high-integrity real-time distributed systems (described in the position paper "Issues with using Ravenscar and the Ada Distributed Systems Annex for High-Integrity Systems"). In this work, they state that the restrictions of Ravenscar preclude the implementation of an efficient runtime support for distributed programming, resulting on augmented code complexity.

Andy presented what he considered to be the possible choices for the model of distributed Ravenscar applications:

- Ravenscar and custom mechanisms for distributed applications support;
- Ravenscar on top of an OS/Middleware;
- Ravenscar and a subset of the DSA;
- Ravenscar and full DSA;
- Ravenscar and a fixed DSA (fixed pool of tasks, fixed priorities, etc.);
- Ravenscar and a new Distributed Real-Time Annex.

Andy also stated that the implementation of this model should be Ravenscar compliant, and that his motivation for a suitable model is that, in his opinion, high-integrity systems can not be built on top of RT-CORBA or the DSA.

Neil presented the restrictions that they had to make to the DSA, in order to guarantee predictability, giving the type of applications (long-lifetime high-integrity distributed systems):

- Producer-consumer model of communications;
- No access types or dynamic dispatching (no RAS/ RACW);
- Asynchronous communication between partitions.

Neil also presented some of the problems they found while building the system:

- The DSA is highly dependent on tagged types and dynamic dispatching, because Ada.Streams are the basis for interconnection between partitions.
- It is not clear if each partition should have its own runtime, or if it is one runtime per processor.

4.1. Discussion

There was some discussion on these problems, and on the appropriateness of the DSA (with and without the real-time extensions) to support Distributed Ravenscar applications:

- The proposed real-time extensions to the DSA incorporate some features that are currently forbidden in Ravenscar (for instance, dynamic task creation and dynamic priorities). Implementations of the distributed runtime must be Ravenscar compliant, thus disallowing the use of these features.
- The problem of Ada.Streams is that their implementation is specified by the LRM, which means relying in tagged types for remote call parameters. Although Ravenscar does not address restrictions to the sequential part of the language, the HRG document "Guide for the Use of the Ada Programming Language in High Integrity Systems" explicitly excludes tagged types and runtime dispatching in these systems.
- Although the LRM does not explicitly mention the issue of runtime systems, in clause E.1(6) it specifies:

  "A library_item is elaborated as part of the elaboration of each partition that includes it. If a normal library unit (see E.2) has state, then a separate copy of the state exists in each active partition that elaborates it. The state evolves independently in each such partition."

  Thus, it was concluded that each active partition should have its own runtime.

Recommendation:

The group concluded that a suitable model for distributed Ravenscar applications is necessary and recommended further work in this issue (maybe for the next IRTAW).