Building Partitioned Architectures Based on the Ravenscar Profile

Brian Dobbing
Aonix Europe Limited
Partridge House, Newtown Road
Henley-on-Thames, Oxon RG9 1HG U.K.
Tel: +44 1491 415016
brian@aonix.co.uk
www.aonix.com

1. ABSTRACT

The requirement to support software partitioning is a recurring theme within High Integrity and Safety Critical systems. The partition concept is used to implement differing access protection levels for applications of varying criticality levels executing on the same processor. Partitions can also be used in fault tolerant systems that require high availability, redundancy or dynamic re-configuration.

The *Ravenscar Profile* was a major output of the 8th International Real-Time Ada Workshop. The profile defines a subset of the Ada95 tasking constructs that matches the requirements of Safety Critical, High Integrity and Hard Real-Time systems by eliminating constructs with high overhead or non-deterministic behavior (semantically or temporally) whilst retaining those elements that form the basic building blocks for constructing analyzable and deterministic real-time software.

This paper describes how a COTS Ada95 compilation system that implements the Ravenscar Profile can be used in the implementation of a partitioned architecture in an Integrated Modular Avionics context based on the ARINC 653 Application Executive (APEX) standard.

2. INTRODUCTION

Partitioning is an essential component of systems that contain a mixture of critical and non-critical software. Partitioning has traditionally been enforced using processor and memory address space boundaries, such that the non-critical code does not use the same physical resources as the critical code. However increasing pressure on the quantity of software now being deployed in embedded systems, together with demands on high reliability such that software can be dynamically reconfigured so as to execute on any available physical resource, result in scenarios where a highly trusted partitioning capability must be deployed to maximize use of processing resources.

The critical partition must be protected by a spatial firewall that ensures absence of intrusion by less-trusted code. Such a firewall is often enforced by hardware when a suitable memory management unit is available. At the most primitive level, this firewall can be implemented using the supervisor/user mode divide.

In addition to the spatial firewall, a temporal firewall is required to ensure that all critical deadlines are met. This firewall must be enforced by an underlying executive which, by implication, is also high integrity code. The executive can elect to perform fixed time-slice round-robin scheduling of all partitions, or to implement a partition priority scheme such that the critical partition gets as much CPU time as it needs.

Support for high integrity partitioning is starting to emerge in COTS real-time operating systems, such as WindRiver's Tornado 3.0 / CirrusOS™ [1]. This support is already established in avionics standards such as the ARINC 653 Application Executive (APEX) [2].

The composition of a partitioned system executive is in two major parts:

- The *kernel* which is the operating system for the processor. This controls the execution of the partitions that are resident on that processor;
- The *partition operating system* (POS) which controls the internal execution of the application

partition, including in some scenarios a set of services for its local threads.

Clearly the kernel and the POS must be developed and verified to at least the same level of criticality as the highest level partition that they must control. A "safe subset" of Ada95 [5] is the leading candidate when considering the development language, and therefore the *Ravenscar Profile* [6], which was designed especially to meet the verification requirements of such software, is an obvious first choice.

3. APEX OVERVIEW

The ARINC 653 Application Executive (APEX) [2] is an established standard within integrated modular avionics. APEX defines the kernel in terms of a Module Operating System (MOS) that controls the *module*, which is one processing unit within the overall system (usually a processor and its memory space). The MOS manages all hardware-related aspects of the module execution, including communication with other modules in a distributed environment. The MOS also controls the scheduling of all of its partitions using fixed time-slice round-robin scheduling.

Partition-level control within APEX is defined by the Partition Operating System (POS) that supports threaded execution within the application partition. The thread definitions are statically configured to simplify verification and implementation. The POS controls the internal scheduling of its threads and includes operations to detect and respond to error situations such as overrun of CPU time budget. The POS also provides several thread services for synchronization and communication, such as events, blackboards, buffers, semaphores etc.

4. OBJECTADA/RAVEN OVERVIEW

ObjectAda/Raven™ [3] is the Aonix implementation of the Ravenscar Profile. In addition to the Ada95 tasking subset defined by the Profile, *Raven* defines a sequential (non-tasking) subset that is consistent with the overall goals of determinism and ease of verification for high integrity systems. For example, a *Raven* implementation does not provide a standard storage pool for dynamic heap allocation, although user-defined storage pools can be defined.

Furthermore, *Raven* supports an open execution architecture that allows a *Raven* program to be developed as:

- either a stand-alone application usually executing in supervisor mode with direct control of the board
- or an application usually executing in user mode that is controlled by an external executive.

This variation is achieved via use of different Board Support Packages (BSP). The BSP can provide the external services required by the runtime system:

- either directly at the hardware level, for example via the real-time clock device, interval timer register, serial I/O ports, MMU and cache settings
- or using the services provided by the underlying executive.

The *Raven* runtime system has been developed to the most rigorous certification guidelines as described in RTCA/DO-178B level A [4] and is therefore well suited to providing the core functionality within a certifiable implementation of APEX.

5. IMPLEMENTING APEX IN RAVEN

5.1 MOS Kernel

The MOS kernel is by nature single threaded with multiple interrupt and fault handlers. Once control has been given to the initial partition to execute, the MOS is entirely interrupt-driven. Such interrupts may be as a result of:

- the timer device indicating that the current partition's time-slice has expired
- a trap handler indicating a fault in the current partition's execution
- an I/O device indicating that an inter-module input-output operation is complete

The Ravenscar Profile supports the definition of Ada interrupt handlers as protected subprograms. Furthermore, the *Raven* open architecture supports installation of low-level handlers for these traps via the BSP implementation-specific configuration of the startup code. The startup code can also be configured such that the required MMU settings are programmed to provide the necessary spatial firewalls between the partition address spaces, and such that the MOS executes in supervisor mode with the required protection levels.

5.2 POS Services

A partition can be multi-threaded. The threads are statically defined in a configuration table and so this maps well to the Ravenscar Profile model of requiring all tasks to be created during program elaboration code.

The partition application code uses an APEX-defined API for its inter-thread services. Consequently there is no direct use of the Ada95 concurrency statements and operations such as *delay_until*, *suspension objects* and *protected objects*. However the API defines inter-thread operations with overlapping semantics to these constructs, and so they can be used as part of the implementation of the POS API services. This model also allows the application program API to use services beyond the subset defined by the Ravenscar Profile. For example, the APEX API requires queues of tasks waiting for access to resources, including in FIFO or priority order, with and without a timeout, which is not directly supported in the Ravenscar Profile. The queuing and

timing can be implemented behind the API interface within the implementation of the POS.

In order to achieve precise control over the scheduling of its threads, the POS needs to implement a thread identity mechanism together with the ability to perform "suspend-self" and "resume-any" operations. Thread identity is provided in the Ravenscar Profile via the predefined Ada.Task_Identification package. Further, the suspend/resume operations are supported in the Ravenscar Profile by means of suspension objects. It is therefore possible for the POS implementation to define a per-task suspension object that it can use to resume execution of any task (the task having suspended on its own suspension object). In the Raven open architecture, it is possible for the BSP to extend the per-task internal data structure that is used for task identification (the *Task Control Block*) to add implementation-defined fields, and so this can be used by the POS implementation to store (and find) the per-task suspension object.

The POS services include the requirement for a thread to return to its startup status when certain types of failure are detected. The most natural mechanism to achieve this in Ada is via exception raising and handling. Support for exception handling is not addressed by the Ravenscar Profile, but the recommendation from several users of the Profile (e.g. [7]) is that exception handling is a valuable feature for hard real-time systems, although perhaps out of scope for safety-critical systems. Exception handling is now supported by the *Raven* runtime system. The disadvantage of using exceptions is that it does not fit smoothly with the use of a standard language-independent API that defines the task restart point (since an Ada exception handler needs to be coded instead). An alternative model would be to provide this functionality via an API that provides the classic *setjmp/longjmp* capability, whereby *setjmp* is called to establish the restart point and *longjmp* is called to perform the return to that point. This feature is also provided by the *Raven* runtime system.

6. CONCLUSION

The requirement to support software executing at different levels of criticality using the same physical resources implies the need to implement a kernel that supports spatial and temporal partitioning of the software. The ARINC 653 Application Executive (APEX) is an example of a standard that defines a partitioned architecture for integrated modular avionics systems.

The Ravenscar Profile has been designed specifically to provide a runtime execution framework that is both functionally and temporally deterministic, in order to simplify verification of software developed to the highest criticality levels. The Aonix ObjectAda Raven runtime system is an example of an implementation of the Ravenscar Profile that is certifiable to RTCA/DO-178B level A.

This paper has outlined how the ObjectAda Raven runtime system is suitable for use as the core in the implementation of both Module and Partition Operating Systems in an APEX implementation. This further enhances the view that the Ravenscar Profile is an excellent subset definition for implementing hard real time, high integrity and safety critical systems.

7. REFERENCES

[1] *Tornado 3.0 Beta program 2000*, Wind River Systems, www.windriver.com

[2] *Minimal Operational Performance Standards for Avionics Computer Resource*, version 4.0, RTCA SC-182 / EUROCAE WG-48, August 1999 Also in *Avionics Application Software Standard Interface*, ARINC AEE Committee, June 1996.

[3] *ObjectAda Real-Time Windows x PowerPC / Raven*, Aonix Inc, 1998

[4] RTCA/DO-178B, *Software Considerations in Airborne Systems and Equipment Certification*, December 1992

[5] *Ada95 Reference Manual*, International Standard ANSI/ISO/IEC-8652:1995, January 1995

[6] B.J.Dobbing and A.Burns, *The Ravenscar Tasking Profile for High Integrity Real_Time Programs*. In Reliable Software Technologies – Ada-Europe '98, Lecture Notes in Computer Science 1411, Springer Verlag (June 1998)

[7] T.Vardanega and G.Caspersen, *Using the Ravenscar Profile for Space Applications: the OBOSS case*, Proceedings of the 10th International Real-Time Ada Workshop 2000, to be published in Ada Letters.