

High-Integrity Systems Development for Integrated Modular Avionics using VxWorks and GNAT

Paul Parkinson¹, Franco Gasperoni²

¹ Wind River, Unit 5 & 6 Ashted Lock Way, Birmingham, B7 4AZ, United Kingdom
Paul.Parkinson@windriver.com

² ACT Europe, 8 Rue de Milan, 75009 Paris, France
gasperon@act-europe.fr

Abstract. This paper presents recent trends in avionics systems development from bespoke systems through to COTS and emerging Integrated Modular Avionics architectures. The advances in Ada and RTOS technologies are explained and the impact of requirements for RTCA/DO-178B and EUROCAE/ED-12B certification and achievements are presented in the context of the GNAT and VxWorks technologies.

1 Introduction

Many avionics systems have successfully been implemented in Ada on bespoke hardware platforms, and have usually performed a dedicated function with limited interaction with narrowly defined external interfaces to external subsystems. However, in recent years, the need has evolved for new applications with ever-increasing levels of functionality, requiring interaction with many external systems, which has had a negative impact on development timescales. This has conflicted with market forces which have exerted pressure for shorter development cycles to bring new systems to market faster. In addition, avionics programmes have been under pressure to reduce life cycle costs, and as bespoke systems have been regarded as the largest single contributory cost factor, alternatives have been sought.

One of the trends which has emerged as a result is the adoption of COTS (commercial off the shelf), but not just in the sense of commercial grade components recommended by the US Department of Defense, but in terms of system-level boards and importantly, COTS software.

Programme offices have learned that significant cost savings can be made where an already available COTS board can be used to perform a function for which a proprietary system would have previously been developed. The risk to the programme is reduced in terms of development timescales and engineering development costs.

However, lessons have been learned from the mistakes made during early COTS adoption, as some of these actually increased programme risk, rather than reducing it as intended. These pitfalls included the use of single-source supplier, closed proprietary interfaces, and dependencies on hardware architectures which would become obsolete during an in-service lifetime exceeding thirty years.

One of the trends which has emerged as a result is Integrated Modular Avionics (IMA), which has placed demands on software systems, including Ada implementations and Real-Time Operating Systems (RTOS). These systems have placed new demands on the certification efforts required for deployment. These interesting issues are discussed in the following sections.

1.1 The Development of High-Integrity Systems

Standards for safety-critical systems have tracked advances in avionics development, and there is now a range of standards which apply to hardware and/or software, covering civil and/or military programmes, and may apply to a single country or a group of nations. These standards also vary in the approaches which they take towards ensuring safety-criticality. The US standard RTCA/DO-178B [1], and its European equivalent EUROCAE ED-12B, place a strong emphasis on testing to demonstrate absence of errors. The UK defence software standard Def Stan 00-55 [2], instead promotes proof of correctness in the design, rather than absence of errors. Despite the differences in the philosophies, recent research work has shown that there is some correlation between these standards [3], and they all place stringent requirements on avionics applications which need to be certified.

The DO-178B/ED-12B standard defines five levels of safety-criticality, ranging from Level E at the least critical, to Level A at the most critical, as shown in table 1 below:

Table 1. DO-178B/ED-12B Safety Criticality Levels

Failure Condition	Software Level	Outcome
Catastrophic	Level A	Death or injury
Hazardous / Severe - Major	Level B	Injury
Major	Level C	Unsafe workload
Minor	Level D	Increased workload
No Effect	Level E	None

The implications for certification are significant, as there is a significant increase in certification effort required as the safety critical level is increased. This applies to software specification, design, implementation and testing phases. DO-178B defines 28 objectives for Level D, rising to 57 for Level C, through to 66 for Level A, which requires significantly more rigour than the lower levels.

Table 2. DO-178B/ED-12B Testing Requirements

Software Level	Testing Requirements Examples
Level A	MCDC Coverage
Level B	Decision Coverage
Level C	Statement Coverage

The top three safety levels are of particular interest to Ada developers, as they require development tool support to perform code coverage. This is usually performed via instrumentation of the source code with trace points prior to compilation by the Ada compiler; then during program execution, a log is generated which can be analyzed to determine the behaviour of the program. Level C specifies Statement Coverage, which requires every statement in the program to have been invoked at least once. Level B specifies Decision Coverage, which requires every point of entry and exit in the program has been invoked at least once and every decision in the program has taken on all possible outcomes at least once. Finally, Level A requires Modified Condition / Decision Condition (MCDC) testing which is explained below. This increases the number of test permutations immensely, and also places additional burdens on the development tool and systems under test.

MCDC Illustration. The complexity of MCDC testing is illustrated by the following code fragment:

```
if A=0 and then B<2 and then C>5 then P; end if;
```

This contains three variables, three conditions and four MCDC cases (as shown in Table 3). For DO-178B Level B certification, only one test case which causes statement P to be executed, whereas for Level A certification, all four possible test cases need to be generated.

Table 3. MCDC Test Cases

A=0	B<2	C>5	P
T	T	T	T
F	?	?	F
T	F	?	F
T	T	F	F

The DO-178B standard is used to certify a complete system implementation and not just individual components of a system, therefore certification evidence must be produced for all software components used within that system. So for COTS, this means that certification evidence will be required for the Ada application, the Ada runtime system and the RTOS beneath it – these are addressed in the following subsections.

1.1.1 Ada Application & Runtime Certification

Ada applications rely on an Ada runtime system to provide services such as tasking or exceptions in their deployed environment. While the size and complexity of the

runtime system is usually not an issue in commercial applications, in safety-critical systems it is, since the certification of the Ada runtime system can in itself prove to be a considerable task.

To deal with this issue, the approaches that have emerged in the past ten to fifteen years restrict the Ada features that can be used in safety-critical applications. The resulting benefit is a simplification of the underlying Ada runtime. Unfortunately, every Ada vendor has come up with its set of Ada restrictions.

Ada 95 with its safety and security annex, the follow-on work done by the Annex H Rapporteur Group [4] as well as the work on the Ravenscar Profile [5, 6, 7] have brought some clarity in this domain. We are hopeful that Ada 0X will contain at least one standardized high-integrity profile. This will enhance portability in the realm of safety-critical applications. A side effect of this will be the extension of the ACATS test suite. As a result, Ada users would have a number of conformant implementations to choose from.

1.1.2 RTOS Certification and VxWorks/Cert

A number of avionics applications which have been certified for deployment have used proprietary in-house RTOS implementations, generally consisting of a kernel with task scheduler some resource management capabilities. The development, maintenance and certification of such kernels may have been manageable on past programmes, but it difficult to envisage how this approach can be viable for future programmes. This is because today's more complex systems, with multiprocessor configurations and many communications interfaces demand more sophisticated functionality from an RTOS. For programmes developing and maintaining an in-house proprietary RTOS, this places a significant additional burden on the programme's developers.

Another approach which has been adopted in a number of safety critical avionics programmes, is to take a commercial RTOS and to certify it as part of the system certification activities. Honeywell Aerospace adopted this approach in the development of the Global Star 2100 Flight Management System, which runs on VxWorks[®], and was certified to DO-17B Level C [8].

However, the next logical step, was for a commercial RTOS vendor to provide an off-the-shelf product with DO-178B certification evidence which could be used in the development of safety critical systems. This is an appropriate approach because the FAA has produced guidelines in N9110.RSC [9], which outline how "reusable software components" (RSC), including an RTOS, can be certified to DO-178B as part of a system. The net result is that the off-the-shelf component can be reused in many programmes, without the need to redevelop certification evidence for the RTOS

kernel each time, provided that it is not changed. The approach taken by Wind River to provide these reusable software components in detail in a later section.

1.2 The Route to COTS

Avionics programmes have sought to achieve portability and interoperability for hardware platforms in the drive towards COTS. The Ada programming language has helped towards this end by enabling software abstraction from both the processor and system architectures.

However, many Ada applications have run on top of a dedicated Ada runtime and kernel which has been not only specific to a processor architecture, but also to an Ada technology supplier. This has means that when some programmes migrate architectures, for example when transitioning from Motorola 68k to PowerPC, may unexpectedly face problems relating to vendor lock-in due to a proprietary Ada kernel implementation. Programmes have also sought to avoid vendor lock-in when selecting an Ada compiler technology. The Ada '95 programming language standard [10], and associated compiler technology verification tests go a long way to providing evidence of intrinsic compatibility between compiler technologies, far more so than for C/C++ counterparts. However, close examination of the Ada compiler technology is also required to determine its openness, and whether the programme using a specific vendor's compiler will become locked into that technology, or will be able to migrate to another technology at a future date.

1.3 The Emergence of Integrated Modular Avionics

In addition to the drive towards COTS to reduce life cycle costs, avionics programmes have also needed to consider the performance requirements of new avionics applications, which have increased dramatically. In order to address these needs, a new model has been developed, *Integrated Modular Avionics* (IMA), which has the following high-level objectives:

- **Common processing subsystems.** These should allow multiple applications to share and reuse the same computing resources. This facilitates a reduction in the number of deployed subsystems which are not fully utilised and provides a more efficient use of system resources which leaves space for future expansion.
- **Software abstraction.** This should isolate the application not only from the underlying bus architecture but also from the underlying hardware architecture. This enables portability of applications between different platforms and also allows the introduction of new hardware to replace obsolete architectures.

IMA seeks to achieve these objectives through the implementation of a layered software model, with an RTOS and architecture-specific software providing an isolation layer above the hardware, and above this portable applications reside.

IMA is likely to become pervasive in avionics in the next few years, as number of high-profile industry research programmes including ASAAC [11] and the EU-funded Project Victoria are developing software architectures for IMA which will impact on the future developments of software technology from commercial vendors, and may also have input into relevant avionics software standards.

1.3.1 Support for emerging IMA architectures

The Avionics Computing Resource Minimal Operational Performance Specification [12] defines two important concepts which are widely used in IMA, these are Spatial Partitioning and Temporal Partitioning.

Spatial Partitioning. This defines the isolation requirements for multiple applications running concurrently on the same processing platform. The applications should not be able to interfere with each other, and is usually achieved through the use of different virtual memory contexts, enforced by a processor's memory management unit (MMU). These contexts are referred to as *Partitions* in the ARINC-653 Specification [13], and contain an application with its own heap for dynamic memory allocation, and a stack for the application's tasks. Applications running in an IMA partition must not be able to deprive each other of shared application resources or those that are provided by the RTOS kernel. These requirements will have an impact on the design and implementation of the Ada runtime system and the underlying RTOS kernel.

Temporal Partitioning. This defines the isolation requirements for multiple applications running concurrently on the same processing platform. This ensures that one application may not utilise the processor for longer than intended to the detriment of the other applications. The ARINC-653 Specification defines a model where an application may be allocated a timeslot of a defined width, and other applications may be allocated timeslots of similar or differing durations. The ARINC scheduler then enforces the processor utilization, by forcing a context switch to a new application at the end of each timeslot.

2 VxWorks/Cert and VxWorks AE653/Cert

In 1999, Wind River set out to develop a VxWorks product which could be certified to DO-178B Level A, and also to define a certification product roadmap to address the needs of safety critical programmes in the near future. The development of the resulting VxWorks/Cert product has already been used on a number of programmes and is described in the next subsection; VxWorks technology advances

to assist the development of IMA applications are also described in the subsequent subsection.

2.1 Development of VxWorks/Cert

In order to produce a version of VxWorks suitable for high integrity systems, close scrutiny of the FAA guidelines and other up and coming standards was required, including the RTCA SC-182 Avionics Computing Resource Minimum Operational Performance Specification [12] and other standards, such as ARINC-653. Then a multiple pass analysis of the whole of the source code base for VxWorks 5.4 and PowerPC architecture-specific code was undertaken. The end result was a certifiable subset of the VxWorks 5.4 API [14]; functionality that had been excluded was that which could compromise predictability or lead to memory fragmentation.

Restrictions of interest to Ada developers are that C++ runtime support was removed, and restrictions on dynamic memory allocation were introduced. This only allowed dynamic memory allocation to occur at system initialization time, which allowed the execution of Ada elaboration code, or C applications to create data structures from the heap. After a call has been to the kernel API `memNoMoreAllocators()` to indicate that system initialization has completed, the VxWorks kernel prevents further dynamic memory allocation and dynamic creation and deletion of tasks, irrespective of whether they are VxWorks tasks or Ada tasks.

The development of high integrity applications with VxWorks/Cert can be undertaken using an enhanced version of the Tornado™ integrated development environment, which is closely integrated with ACT's GNAT Pro for Tornado development tools. This subject was recently discussed by one of the authors in an industry journal [15].

In order to develop the certification evidence for VxWorks/Cert, a number of qualified CASE tools were employed. These were used in conjunction with automatic test harnesses which generated and collated MCDC test results. These results have been analyzed and audited, and in the rare cases full coverage results are not generated, a manual justification for the proof of correctness of behaviour has been developed. Such cases include entry and exit from an interrupt service routine (ISR), which cannot be tracked by CASE tools in some circumstances (e.g. during kernel start-up). The complete VxWorks/Cert certification evidence has been developed by Verocel in a hyperlinked format, navigable in an HTML browser. This not only greatly assists the certification audit process, by providing fast and accurate cross-referencing capability between different levels of specification, design documentation, source code and test results, but can also be extended, enabling programmes to add certification evidence for their Ada applications.

2.2 Development of VxWorks AE653/Cert

In order to fulfill the needs of IMA development programmes, Wind River has added spatial partitioning and temporal partitioning to VxWorks technology.

2.2.1 Spatial Partitioning

VxWorks AE implements spatial partitioning through *Protection Domains*, which are analogous to ARINC-653 partitions. Protection domains are flexible containers into which application tasks, objects and resources can be placed. Each protection domain employs its own stack and heap locally, which guarantees resource availability, as they cannot be usurped by applications running in other protection domains. Protection domains provide protection against errant memory accesses from applications running in other protection domains, implemented using an MMU virtual memory context. The implementation of protection domains is described in more detail in the *High Availability for Embedded Systems* white paper [16].

The certifiable version of VxWorks AE, *VxWorks AE653*, also guarantees kernel resource availability by binding a separate instance of the VxWorks kernel to each application protection domain, and providing a message passing backplane between protection domains. This implementation contrasts with monolithic kernel implementations where an errant application can deprive other applications of kernel resources. The use of separate kernel instances also means that separate Ada runtimes can be used within each application protection domain.

2.2.1 Temporal Partitioning

The task scheduling models used in avionics systems are interesting. In recent years, there has been a trend in many embedded and real-time systems to migrate from the traditional time-slicing model towards pre-emptive priority-based scheduling schemes, as these provide predictable and deterministic behaviour. These schemes have been successfully used in conjunction with Ada-based applications in major avionics programmes, such as GENESYS [17].

However, because IMA systems can support multiple applications running on the same processor, and for this a time-based partitioning scheme is required. The ARINC-653 Specification defines a temporal partitioning scheduling model which guarantees time slots to partitions, and within these partitions applications tasks can be scheduled using other scheduling policies. VxWorks AE653 implements ARINC-653 partition scheduling via a protection domain scheduler, this provides guaranteed time slots to applications within each protection domain, and uses priority-based preemptive scheduling for tasks within the domain.

3 GNAT Pro High Integrity Edition

GNAT Pro High-Integrity Edition (GNAT Pro HIE in the remainder of this document) is intended to increase safety as well as reduce the certification cost of

applications that need to meet safety-critical certification standards such as RTCA/DO-178B or EUROCAE ED-12B.

In addition to providing two high-integrity Ada profiles, GNAT Pro HIE contains a number of features especially useful for safety critical programming. The following sections will describe each of these elements.

3.1 GNAT Pro HIE Certification Benefits

An important benefit of GNAT Pro HIE when it comes to user code certification is the ease in which object code can be traced to the original source [18].

A compiler option allows display of the low-level generated code in Ada source form. This acts as an intermediate form between the original source code and the generated object code, thus supporting traceability requirements. This intermediate presentation can be used as a reference point for verifying that the object code matches the source code. This expanded low-level generated code can also be used as a reference point for run-time debugging. As an example, consider the following code:

```
procedure Try is
  X : array (1 .. 100) of Integer := (1, 2, others => 3);
begin
  null;
end;
```

Using a special switch you can display (or save to a file) the low-level Ada source code that is internally created by the compiler and from which the final object code is generated. For the above the source code is:

```
procedure try is
  x : array (1 .. 100) of integer;
  x (1) := 1;
  x (2) := 2;
  I5b : integer := 2;
  while I5b < 100 loop
    I5b := integer'succ(I5b);
    x (I5b) := 3;
  end loop;
begin
  null;
  return;
end try;
```

Thus, instead of having to go from high-level Ada source directly to assembly and object code, GNAT Pro HIE allows programmers to introduce the low-level Ada code step when mapping source to object code. Because the low-level Ada code used is very simple, this code maps straightforwardly to the generated assembly and object code. Furthermore, this low-level Ada code is mostly target independent.

As a result, the benefits of this approach are:

- The code certification process is less costly since the semantic gap between each step (Ada to low-level Ada to assembly to object code) is reduced
- The code certification process can be speeded up as it can be done in 3 parallel steps by 3 separate teams (Step 1: Ada to low-level Ada; Step 2: low-level Ada to assembly; Step 3: Assembly to object code)
- Because low-level Ada is mostly target independent, part of the certification work can be reused.

3.2 GNAT Pro HIE Programming Benefits

GNAT Pro HIE contains a number of features especially useful for safety-critical programming:

- The GNAT Pro HIE compiler has a special switch to generate Ada representation clauses about the choice of data representations.
- The compiler produces an extensive set of warning messages to diagnose situations that are likely to be errors, even though they do not correspond to illegalities in Ada Reference Manual terms: Missing returns from functions, inner variable hiding an outer variable with the same name, infinite recursion, uninitialized variables, unused variables, unnecessary with clauses, etc.
- The compiler performs various style checks that can be used to enforce rigorous coding standards, easing the verification process by ensuring that the source is formatted in a uniform manner.
- Annex H of the Ada 95 Reference Manual [10] is fully implemented, and all implementation-dependent characteristics of the GNAT implementation are defined in the GNAT Reference Manual [19], further supporting the goal of reviewable object code.

3.3 GNAT PRO HIE Profiles

GNAT Pro HIE offers the choice of two profiles. Both profiles restrict the Ada constructs that a profile-compliant application can employ. In either case, if an application uses a construct not allowed by the profile, compilation will fail. Furthermore, the binder will check that all units in the applications are compiled using the same profile. The two GNAT Pro HIE profiles are:

- **No-Runtime profile** (no Ada tasking). This profile guarantees that no Ada runtime library is used in conforming applications. With this profile only the application source code will appear in the final executable and needs to be certified. This profile is derived from the GNORT technology described in [18].

- **Ravenscar profile** (Ravenscar Ada 95 tasking [5, 6, 7]). This profile complements the previous one by providing a simple Ada runtime implementing the Ravenscar tasking subset. In this case the Ada runtime is designed to expedite the certification process for an application that needs to use concurrency. In certain situations it may be simpler and less expensive to certify a concurrent program built on the Ravenscar Profile than to certify a sequential program with no runtime library (and where the concurrency needs to be simulated in application code).

The programmer chooses the profile by using a compilation switch that specifies whether she wants No-Runtime, Ravenscar, or the full Ada 95 runtime.

Both profiles are upwardly compatible with the SPARK Ada profile defined by Praxis Critical Systems [20]. In this case, upwardly compatible means that GNAT Pro HIE can compile any SPARK Ada compliant program. Furthermore, when using the “No-Runtime” profile, the user is guaranteed that the final SPARK Ada application will contain only code written by the user.

The implementation of the Ravenscar profile relies on VxWorks/Cert. It is remarkable to consider that Ravenscar Ada 95 tasking can be implemented on top of the VxWorks/Cert API with only 200 source lines of simple Ada code for the runtime and 60 source lines of Ada for the libraries. This is because the VxWorks/Cert kernel provides all the necessary synchronization and scheduling primitives to do a straightforward implementation of Ravenscar. The number of VxWorks/Cert primitives used to implement Ravenscar is around 10. These primitives include creating tasks, setting their priority plus mutex operations.

When using the No-Runtime profile in conjunction with VxWorks/Cert it is possible to make direct calls to VxWorks/Cert to create tasks, synchronize them using the VxWorks mutex primitives, etc. As a matter of fact GNAT Pro HIE, regardless of the profile, comes with a complete Ada binding to the 400 routines in the VxWorks/Cert API.

Note that all Integrated Modular Avionics issues dealing with spatial and temporal partitioning are taken care of transparently by VxWorks AE653. Furthermore, for each VxWorks AE653 protection domain the programmer can select a different GNAT Pro HIE profile.

3.4 GNAT PRO HIE Ada Restrictions

Although restricted in terms of dynamic Ada semantics, both profiles fully support static Ada constructs such as generics and child units. The use of tagged types (at library level) and other Object-Oriented Programming features is also allowed, although the general use of dynamic dispatching may be prohibited at the application level through pragma `Restrictions`. Generally speaking pragma `Restrictions`,

as defined in Ada 95 Annex H, can be used to further constrain the Ada features that a programmer can use in the safety-critical application being developed.

The features excluded from the GNAT Pro HIE profiles are as follows:

- Exception handlers (see the section “Exceptions in GNAT Pro HIE” below)
- Packed arrays with a component size other than 1, 2, 4, 8, 16 or 24 bits
- The exponentiation operator
- 64-bit integer (the type `Long_Long_Integer`) or fixed-point types
- Boolean operations on packed arrays (individual elements may be accessed)
- Equality and comparison operations on arrays
- The attributes `External_Tag`, `Image`, `Value`, `Body_Version`, `Version`, `Width`, and `Mantissa`
- Controlled types
- The attributes `Address`, `Access`, `Unchecked_Access`, `Unrestricted_Access` when applied to a non library-level subprogram
- Non library-level tagged types
- Annex E features (Distributed Systems)

In addition the No-Runtime profile excludes all the tasking constructs while the Ravenscar profile only allows Ravenscar Ada 95 tasking as defined in [5].

Explicit withs of library units are permitted. However, in an environment where the code needs to be certified, the programmer takes responsibility for ensuring that the referenced library units meet the certification requirements.

3.5 Exceptions in GNAT PRO HIE

Exception handlers are forbidden in GNAT Pro HIE, so exceptions can never be handled. However, exceptions can be raised explicitly with a raise statement (which are allowed in GNAT Pro HIE) or, if run time checking is enabled, then it is possible for the predefined exceptions such as `Constraint_Error` to be raised implicitly.

The result of such a raise is to call procedure `__gnat_last_chance_handler`, which must be provided by the programmer at the application level. This procedure should gracefully terminate the program in a fail-safe mode. To suppress all runtime error checking and disallow raise statements one can use the `No_Exceptions` restriction of Annex H.

3.6 Allocators in GNAT PRO HIE

Dynamic memory allocation and deallocation is permitted in GNAT Pro HIE. Use of these features will generate calls to routines `__gnat_malloc` and `__gnat_free` that must be provided by the programmer at the application level. To prohibit the use

of allocators or unchecked deallocation, programmers can use the `No_Local_Allocators`, `No_Allocators`, `No_Implicit_Heap_Allocations`, `No_Unchecked_Deallocation` restrictions defined in Annex H. These restrictions can be enforced at the application level. For instance `No_Local_Allocators` prohibits the use of allocators except at the library level. Such allocators can be called once at elaboration time, but cannot be called during execution of the program.

3.7 Array and Record Assignments in GNAT PRO HIE

Array and record assignments are permitted in GNAT Pro HIE. Depending on the target processor these constructs can generate calls to the C library functions `memcpy()` or `bcopy()`. A certifiable version of these routines is provided in `VxWorks/Cert`.

3.8 Functions returning unconstrained Objects

GNAT Pro HIE allows functions returning unconstrained objects (e.g. unconstrained arrays). To implement this capability the compiler uses a secondary stack mechanism requiring runtime support. A sample implementation comes with GNAT Pro HIE. However, since no certification material is provided for this sample implementation, it is the programmer's responsibility to certify it. To disable this capability GNAT Pro HIE provides a `No_Secondary_Stack` Restrictions pragma.

3.9 Controlling Elaboration with GNAT Pro HIE

One of the objectionable issues in a certification context is the fact that the Ada compiler generates elaboration code without the programmer being aware that such code is being generated. The elaboration code must be generated to meet Ada semantic requirements, and it is always safe to do so. However, the elaboration code needs to be certified and this increases certification costs (why was this elaboration code implicitly generated by the compiler, what Ada requirement does it meet, what test cases do we need to devise for the elaboration code, etc.)

GNAT Pro HIE provides a Restrictions pragma `No_Elaboration_Code`, that allows programmers to know when elaboration code would be generated by the compiler. When the pragma Restrictions `No_Elaborations_Code` is specified in a compilation unit, the GNAT Pro HIE compiler stops with an error every time it needs to generate elaboration code. It is up to the programmer to rewrite the code so that no elaboration code is generated. As an example consider the following:

```
package List is
  type Elmt;
  type Temperature is range 0.0 ... 1_000.0;
  type Elmt_Ptr is access all Elmt;
```

```

    type Elmt is record
      T      : Temperature;
      Next   : Elmt_Ptr;
    end record;
  end List;

  pragma Restrictions (No_Elaboration_Code);
  with List;
  procedure Client is
    The_List : List.Elmt;
  begin
    null;
  end Client;

```

When compiling unit Client the compiler will stop with the error

```

client.adb:4:04: violation of restriction "No_Elaboration_Code"
at line 1

```

To understand why GNAT needs to generate elaboration code for object The_List, remember that Ada requires that all pointers be null initialized. To see the elaboration code that would be generated the user can remove the No_Elaboration_Code restriction and use the -gnatG switch to view the low-level version of the Ada code generated by GNAT. In this case you get

```

package list is
  type list__elmt;
  type list__temperature is new float range 0.0E0 ..
(16384000.0*2**(-14));
  type list__elmt_ptr is access all list__elmt;
  type list__elmt is record
    t : list__temperature;
    next : list__elmt_ptr;
  end record;
  freeze list__elmt [
  procedure list__init_proc (_init : in out list__elmt) is
  begin
    _init.next := null;
    return;
  end list__init_proc;
  ]
end list;

with list; use list;
procedure client is
  the_list : list.list__elmt;
  list__init_proc (the_list);
begin
  null;
  return;
end client;

```

The reason why elaboration code is generated inside procedure Client is because the pointer inside The_List object must be initialized to null. To avoid elaboration code the programmer can add an explicit initialization as shown below:

```

pragma Restrictions (No_Elaboration_Code);
with List; use List;
procedure Client is
  The_List : List.Elmt := (0.0, null);
begin

```

```
    null;  
end Client;
```

By making the initialization explicit, rather than sweeping it under the compiler's rug, initialization becomes part of the requirements mapping and application design. In a certification context it may be preferable to certify code that you write explicitly rather than code that gets generated for you implicitly by a compiler.

3.10 Controlling Implicit Loops and Conditionals with GNAT Pro HIE

Certain complex constructs in Ada result in GNAT generating code that contains implicit conditionals, or implicit for loops. For example, slice assignments result in both kinds of generated code.

In some certification protocols, conditionals and loops require special treatment. For example, in the case of a conditional, it may be necessary to ensure that the test suite contains cases that branch in both directions for a given conditional. A question arises as to whether implicit conditionals and loops generated by the compiler are subject to the same verification requirements.

GNAT Pro HIE provides two additional restriction identifiers that address this issue by controlling the presence of implicit conditionals and loops:

```
pragma Restrictions (No_Implicit_Conditionals);  
pragma Restrictions (No_Implicit_Loops);
```

These are partition-wide restrictions that ensure that the generated code respectively contains no conditionals and no loops. This is achieved in one of two ways. Either the compiler generates alternative code to avoid the implicit construct (possibly at some sacrifice of efficiency) or, if it cannot find an equivalent code sequence, it rejects the program and flags the offending construct. In the latter situation, the programmer will need to revise the source program to avoid the implicit conditional or loop.

4 Summary and Conclusions

The avionics industry has witnessed a major shift towards COTS in recent years. Ada and RTOS suppliers have needed to enhance their technologies in order to assist this migration process.

This paper has outlined how Wind River and ACT have enhanced VxWorks and GNAT to meet the specific needs of the aerospace and defence market, and has demonstrated how COTS software technology can be used to build high integrity systems for safety critical applications.

References

1. RTCA, "DO-178B - Software Considerations in Airborne Systems and Equipment Certification", URL <http://www.rtca.org>
2. UK Ministry of Defence, "Requirements for Safety Related Software in Defence Equipment", Def Stan 00-55.
3. Dr. C. H. Pygott, "A Comparison of Avionics Standards", DERA/CIS3/TR990319/1.0, British Crown Copyright 2000.
4. "Guidance for Use of Ada in High Integrity Systems", ISO/IEC TR 15942.
5. Ted Baker and Tullio Vardanega, "Session Summary: Tasking Profiles", Ada Letters September-October 1997, Vol. XVII, Number 5, pages 5-7.
6. A. Burns and B. Dobbing, "The Ravenscar Tasking Profile for High Integrity Real-Time Programs", pp. 1-6 in Proceedings of ACM SigAda Annual Conference, ACM Press, Washington DC, U.S.A. (8-12 November 1998).
7. Alan Burns, "Guide for the use of the Ada Ravenscar Profile in high integrity systems – the work of the HRG", Ada User Journal, Vol. 22, Number 3, September 2001, pp182-187.
8. Wind River Datasheet, "Honeywell customer success story", <http://www.windriver.com>.
9. FAA Draft Notice N9110.RSC.
10. S. Tucker Taft et al.: Ada 95 Reference Manual - Language and Standard Libraries. International Standard ISO/IEC 8652:1995(E), Springer, LNCS 1246, ISBN 3-540-63144-5
11. J. Kemp, A. Wake, W. Williams, "The Development of the ASAAC Software Architecture", ERA Avionics Show 2000.
12. RTCA SC-182 "Avionics Computing Resource Minimum Operational Performance Specification (ACR MOPS)", <http://www.rtca.org>.
13. ARINC-653 Specification, <http://www.arinc.org>.
14. "VxWorks/Cert Subset Definition and Rationale v1.2a", Wind River Systems.
15. Paul Parkinson, "Hochverfügbar – Komplexe langlebige System mit Ada und VxWorks Entwickeln" (Developing High-Integrity Systems with VxWorks and Ada), pp 36-39, Elektronik Praxis, 2 Oktober 2001.
16. "High Availability for Embedded Systems" white paper, Wind River Systems. <http://www.windriver.com>.
17. "GENESYS – An Application of OO Technology to Aircraft Display Systems", Neil Davidson, BAE Avionics Ltd. Symposium on Reliable Object Oriented Programming, IEE, 24th October 2001
18. Roderick Chapman and Robert Dewar, "Re-engineering a Safety-Critical Application Using SPARK 95 and GNORT", Reliable Software Technologies, Ada-Europe'99, LNCS 1622, pp 40-51.
19. "GNAT Reference Manual", <http://www.gnat.com>.
20. John Barnes, "High Integrity Ada: The SPARK Approach", Addison Wesley, 1997