ADA USER JOURNAL

Volume 32
Number 4
December 2011

Contents

Editorial Policy for Ada User Journal

Publication

Ada User Journal — The Journal for the international Ada Community — is published by Ada-Europe. It appears four times a year, on the last days of March, June, September and December. Copy date is the last day of the month of publication.

Aims

Ada User Journal aims to inform readers of developments in the Ada programming language and its use, general Ada-related software engineering issues and Ada-related activities in Europe and other parts of the world. The language of the journal is English.

Although the title of the Journal refers to the Ada language, any related topics are welcome. In particular papers in any of the areas related to reliable software technologies.

The Journal publishes the following types of material:

- Refereed original articles on technical matters concerning Ada and related topics.
- News and miscellany of interest to the Ada community.
- Reprints of articles published elsewhere that deserve a wider audience.
- Commentaries on matters relating to Ada and software engineering.
- Announcements and reports of conferences and workshops.
- Reviews of publications in the field of software engineering.
- Announcements regarding standards concerning Ada.

Further details on our approach to these are given below.

Original Papers

Manuscripts should be submitted in accordance with the submission guidelines (below).

All original technical contributions are submitted to refereeing by at least two people. Names of referees will be kept confidential, but their comments will be relayed to the authors at the discretion of the Editor.

The first named author will receive a complimentary copy of the issue of the Journal in which their paper appears.

By submitting a manuscript, authors grant Ada-Europe an unlimited license to publish (and, if appropriate, republish) it, if and when the article is accepted for publication. We do not require that authors assign copyright to the Journal.

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

News and Product Announcements

Ada User Journal is one of the ways in which people find out what is going on in the Ada community. Since not all of our readers have access to resources such as the World Wide Web and Usenet, or have enough time to search through the information that can be found in those resources, we reprint or report on items that may be of interest to them.

Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.

We have a reciprocal approach in granting permission for other publications to reprint papers originally published in *Ada User Journal.*

Commentaries

We publish commentaries on Ada and software engineering topics. These may represent the views either of individuals or of organisations. Such articles can be of any length – inclusion is at the discretion of the Editor.

Opinions expressed within the *Ada User Journal* do not necessarily represent the views of the Editor, Ada-Europe or its directors.

Announcements and Reports

We are happy to publicise and report on events that may be of interest to our readers.

Reviews

Inclusion of any review in the Journal is at the discretion of the Editor.

A reviewer will be selected by the Editor to review any book or other publication sent to us. We are also prepared to print reviews submitted from elsewhere at the discretion of the Editor.

Submission Guidelines

All material for publication should be sent to the Editor, preferably in electronic format. The Editor will only accept typed manuscripts by prior arrangement.

Prospective authors are encouraged to contact the Editor by email to determine the best format for submission. Contact details can be found near the front of each edition.

Example papers conforming to formatting requirements as well as some word processor templates are available from the editor. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

I would like to start this issue, the last of the year 2011, by drawing your attention to a major Ada-related event due to occur in the New Year: the official release of the 2012 revision of the Ada language. The process leading to the submission of the proposed language revision to ISO has been put in motion by WG9 in November 2011. The plan is that WG9 members will vote on it during the month of March 2012 and, if all goes well, the revision document (termed Committee Draft in ISO jargon) will then be submitted to the higher tiers of ISO for the 7-month period of formal voting. Hopes thus are that Ada 2012 will become official in December 2012.

As noted in the Ada 2012 Rationale introductory chapter, published in the previous issue of the Journal, this revision contains developments in aspects and contracts for Ada entities, containers, flexible expressions, functions' parameter modes, multicore and multithreaded processing, as well as access types and dynamic storage management. And it is not a coincidence that the first technical chapter of the Rationale, which we publish in this issue, precisely deals with the first of these: aspects and contracts, one of the main additions appearing with the forthcoming revision.

Continuing with the technical contents of the issue (and with Ada 2012), we also publish an article by Kristoffer Gregertsen and Amund Skavhaug, from NTNU, Trondheim, Norway, on the design and implementation of the Ada 2012 execution time accounting for interrupts in a bare-board runtime environment. I would also like to draw your attention to the report on the 15th International Real-Time Ada Workshop (IRTAW-15), which was held in Fuente Dé, in the mountains nearby Santander, Spain. This report provides a short summary of the workshop's sessions and discussed topics, allowing readers to get some insight in the discussions and results of the workshop, and areas for future work.

In the issue, we also return the Ada Gems section, providing two gems by Pascal Obry, on the use of SOAP (Simple Object Access Protocol) in Ada. To finalize, the News, Calendar and Forthcoming Events sections complete the issue. The latter provides preliminary information on three events which will take place in 2012: the returning Ada Developer Room at the Free and Open source Software Developers' European Meeting (FOSDEM) next February in Brussels, Belgium; advance information concerning the 17th International Conference on Reliable Software Technologies – Ada-Europe 2012, to take place June 2012 in Stockholm, Sweden; and the ACM SIGAda's Annual International Conference, to take place December 2012 in Boston, Massachusetts, USA.

Our best wishes for 2012,

Luís Miguel Pinho
Porto
December 2011
Email: lmp@isep.ipp.pt

Quarterly News Digest

Marco Panunzio

University of Padua. Email: panunzio@math.unipd.it

Contents

Ada-related Events

[To give an idea about the many Ada-related events organized by local groups, some information is included here. If you are organizing such an event feel free to inform us as soon as possible. If you attended one please consider writing a small report for the Ada User Journal. —mp]

Call for Papers Ada-Europe 2012

From: Dirk Craeynest <dirk@cs.kuleuven.ac.be>
Date: Fri, 9 Sep 2011 20:32:00 +0100
Subject: CfP 17th Conf. Reliable Software Technologies, Ada-Europe 2012
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

CALL FOR PAPERS

17th International Conference on

Reliable Software Technologies - Ada-Europe 2012

11-15 June 2012, Stockholm, Sweden

http://www.ada-europe.org/conference2012

Organized by Ada-Europe,

in cooperation with ACM SIGAda (approval pending)

*** CfP in HTML/PDF on web site ***

General Information

The 17th International Conference on Reliable Software Technologies – Ada-Europe 2012 will take place in Stockholm, Sweden. Following its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical and industrial programs, along with parallel tutorials and workshops on Monday and Friday.

Schedule

28 November 2011: Submission of regular papers, tutorial and workshop proposals

12 January 2012: Submission of industrial presentation proposals

3 February 2012: Notification of acceptance to all authors

2 March 2012: Camera-ready version of regular papers required

11 May 2012: Industrial presentations, tutorial and workshop material required

Topics

The conference has successfully established itself as an international forum for providers, practitioners and researchers into reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies.

To mark the completion of the technical work for the Ada 2012 standard revision process, contributions that discuss the potential of the revised language are sought after. In parallel, facing the challenges presented to the development of reliable concurrent software, multicore programming models is added to the conference topics of interest.

[…]

Nominations for the 2011 SIGAda Awards

From: John McCormick <mccormick@cs.uni.edu>
Date: Wed, 17 Aug 2011 09:44:05 -0700
Subject: Call for SIGAda Award Nominations
Newsgroups: comp.lang.ada

Dear Members of the Ada Community:

On Thursday, 10 November 2011, the 2011 SIGAda Awards will be presented in a special morning plenary session at the SIGAda 2011 conference in Denver, Colorado. (See http://www.sigada.org/conf/sigada2011/ if you have somehow missed announcements of this year's annual SIGAda international conference.)

We welcome your nominations of deserving recipients.

The ACM SIGAda Awards recognize individuals and organizations who have made outstanding contributions to the Ada community and to SIGAda.

The two categories of awards are:

(1) Outstanding Ada Community Contribution Award

-- For broad, lasting contributions to Ada technology & usage.

(2) ACM SIGAda Distinguished Service Award

-- For exceptional contributions to SIGAda activities & products.

Please consider who should be nominated this year. You may nominate a person for either or both awards, and as many people as you think worthy. One or more awards will be made in both categories.

Please visit http://www.sigada.org/exec/awards/awards.html#Previous and peruse the names of past winners. This may help you think about the measure of accomplishment that is appropriate. You may be aware of people who have made substantial contributions that have not yet been acknowledged. Nominate them. Consider what you believe to be the best developments in the Ada community or SIGAda in the last year; the last 5 years; since Ada's inception. Who was responsible? Nominate them.

Please note that anyone who has received either of the two awards remains eligible for the other. Perhaps there is an outstanding SIGAda volunteer who has won our Distinguished Service Award and who has also made important contributions to the advance of Ada technology, or visa versa. Nominate him or her!

The nomination form is available on the SIGAda website at http://www.sigada.org/exec/awards/awards.html#Form.

Submit your nomination as an e-mail or e-mail attachment to SIGAda-Award@acm.org.

The ACM SIGAda Awards Committee, comprised of volunteers who have previously won an award, will determine this year's recipients from your nominations.

Call our attention to the people who are most deserving, by nominating them. And please nominate by Sunday September 25!

Your participation in the nominations process will help maintain the prestige and honor of these awards.

Thank you,

John McCormick

Past Chair ACM SIGAda

[information about the recipients at http://www.sigada.org/exec/awards/awards.html —lmp]

Ada Semantic Interface Specification (ASIS)

Building ASIS for GCC 4.6

From: "Forward In Code" blog
Date: Sat, 15 Oct 2011
Subject: Building ASIS
URL: http://forward-in-code.blogspot.com/2011/10/building-asis.html

This is a note on building ASIS for GCC 4.6.

"The Ada Semantic Interface Specification (ASIS) is an interface between an Ada environment as defined by ISO/IEC 8652 (the Ada Reference Manual) and any tool requiring information from this environment. An Ada environment includes valuable semantic and syntactic information. ASIS is an open and published callable interface which gives

CASE tool and application developers access to this information. ASIS has been designed to be independent of underlying Ada environment implementations, thus supporting portability of software engineering tools while relieving tool developers from having to understand the complexities of an Ada environment's proprietary internal representation."

AdaCore have implemented ASIS for GNAT. The "Ada environment" is a binary record of the compiler's internal state, output on request (the flag -gnatct) in files of type .adt (tree files).

Clearly it's vital that the ASIS library has the same view of the compiler's internal state as the compiler had when the .adt file was generated. This is organised by using consistent versions of the compiler's access mechanisms on both sides.

For unsupported customers, AdaCore releases ASIS updates in parallel with the GNAT GPL compiler toolset. The GNAT GPL release schedule (discussed as part of the Debian policy for Ada) doesn't match the FSF schedule, with the result that GNAT GPL 2010 is roughly the same from the Ada point of view as both GCC 4.5 and GCC 4.6.

It was possible to build an adequate ASIS for GCC 4.5 with very little alteration: at the time I wrote

> [The internal representation is] determined for a particular compiler release by particular compiler components, Sinfo and Snames, so to make ASIS work you need to include the appropriate files from your compiler. Sinfo is just the source files sinfo.ads, sinfo.adb. Snames is created from template files snames.ads-tmpl and snames-adb.tmpl.

As well as this, you'll need to copy gnatvsn.ads from your compiler, and edit gnatvsn.adb to match (for example, GNAT GPL 2010 source includes GNATPro as a build possibility, which isn't allowable in the FSF sources for GCC 4.5.0).

So, to adapt ASIS GPL 2010 for use with GCC 4.5.0, in asis-2010-src/gnat/:

replace sinfo.ad[bs] by gcc-4.5.0/gcc/ada/sinfo.ad[bs]

replace snames.*-tmpl by gcc-4.5.0/gcc/ada/snames.*-tmpl

replace gnatvsn.ads by gcc-4.5.0/gcc/ada/gnatvsn.ads

edit gnatvsn.adb to remove the 'gnatpro' choices, not in the .ads

However, "roughly the same" isn't adequate for GCC 4.6. The recipe above allowed the library to build, but nothing built with it would run (an inconsistency in the tree file format).

Ludovic Brenta, writing on comp.lang.ada, said

> Debian solves that problem by introducing libgnatvsn, a library compiled from GCC sources and containing gnatvsn.ads, sinfo.ad[bs], snames.ad[bs] and everything they depend on. libasis is compiled against libgnatvsn.

There's an alternative to creating a libgnatvsn.a or .dylib which merely depends on your having a compiler build tree around!

The way the ASIS GPL source distribution is structured is that the required compiler sources are in a subdirectory gnat/. The approach I've adopted is to replace this with a subdirectory gnatvsn/; I've modified the Makefile to copy the required sources from the compiler build and source trees (the Snames sources are built from templates during the compiler build).

To start with, I copied the same files that AdaCore supplied in gnat/. During the build it turned out that a few more are needed for GCC 4.6 (not surprising, since GNAT GPL 2010 is forked from GCC 4.3). The units are Aspects and Sem_Aux.

After having set up the compiler-derived sources, a couple of problems turned up while compiling ASIS:

the compiler doesn't understand Name_Implemented_By_Entry

the compiler doesn't understand Is_Overriding_Operation

so I patched these references out (I don't think it's going to affect users much).

This only leaves one problem; the compiler's Gnatvsn unit imports a symbol_version_string, which comes from the compiler's version.c.

Unfortunately, this file depends on a lot of macros defined as part of the compiler build; so the easiest way to get the right object is to copy the built version.o from the compiler build tree into libasis.a.

The patches are here [http://dl.dropbox.com/u/34783908/Ada/asis-gpl-2010-gcc-4.6.0.diff —mp].

As ever, use patch -p1 to apply.

Ada-related Resources

GTKAda tutorials

From: Sunny <daetalusun@gmail.com>
Date: Mon, 26 Sep 2011 17:10:08 -0700
Subject: Where can I find GTKADA tutorials?
Newsgroups: comp.lang.ada

Hello All!

Can anyone tell me where could get some GtkAda tutorials? I only have GtkAda User's Guide and GtkAda RM. But I want to get some tutorials about GtkAda in details, including samples and other things. Could you give me some links or send it to me if you have tutorials about this, please? […]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 27 Sep 2011 09:31:02 +0200
Subject: Re: Where can I find GTKADA tutorials?
Newsgroups: comp.lang.ada

[…]

Some simple samples are under

http://rosettacode.org/wiki/Rosetta_Code

in the corresponding GUI tasks.

GTK tutorials are here:

http://www.gtk.org/documentation.php

Note that GtkAda is thin bindings and to figure out what to do, you just read Gtk documentation.

The new version of GtkAda RM on AdaCore site is awful. Try to find the old one, there is not that many changes.

For complex stuff, e.g. tree view, custom renderers and stores, GIO interfacing, tasking, there is also some, but you should learn the basic stuff first.

AI library frameworks in Ada

From: Pablo Rego
Date: Fri, 21 Oct 2011
Subject: AI library framework in Ada
URL: http://stackoverflow.com/questions/7852149/ai-library-framework-in-ada

I'm looking for an Ada-constructed framework for AI. I think Ada would be perfect for implementing temporal and stochastic paradigms due to its tasking and real-time mechanisms, but did not find anyone who tried to make such a libraries. Actually I did not find strong implementations on other languages too. For C# I found http://www.c-sharpcorner.com/1/56/, and for C++ I found http://mind.sourceforge.net/cpp.html but both did not get much popularity. Maybe java has good AI libraries too, but I do not know. So, do you know an Ada implementation? Would it be useful for more anyone? If you know libraries from other languages, it would be useful to know and compare the implementation models in java, for example. Thanks.

From: Marc A. Criley
Date: Fri, 21 Oct 2011
Subject: AI library framework in Ada
URL: http://stackoverflow.com/questions/7852149/ai-library-framework-in-ada

Here's a few resources:

Book, rather old, though (1989): Artificial Intelligence With Ada

[http://rads.stackoverflow.com/amzn/click/0070033501 —mp]

Looks like some kind of university student dissertation:

MUTANTS: A generic genetic algorithm toolkit for Ada 95

[http://www.permutationcity.co.uk/projects/mutants/MutantsReport.pdf.gz —mp]

Dmitry Kazakov's AI stuff, mostly fuzzy logic. (Dmitry writes really nice software.)

[http://www.dmitry-kazakov.de/ai.htm —mp]

From: T.E.D.
Date: Fri, 21 Oct 2011
Subject: AI library framework in Ada
URL: http://stackoverflow.com/questions/7852149/ai-library-framework-in-ada

I once had a school AI project that used the CLIPS AI builder library.

[http://clipsrules.sourceforge.net/ —mp]

Since I avoid coding in C where I don't have to, I made an Ada Binding to it, which I believe is licensed without restriction.

[http://telepath.com/~dennison/Ted/AdaClips/AdaClips.html —mp]

If you want it, have at.

I used it to build an expert system capable of playing a user's opening moves in Empire. All the code is either in Ada, or Clips' expert system specification language.

[http://telepath.com/~dennison/Ted/Fodderbot/Fodderbot.html —mp]

Dynamic plugin loading in Ada

From: Thomas Løcke <tl@ada-dk.org>
Date: Fri, 26 Aug 2011
Subject: Dynamic Plug-in Loading with Ada
URL: http://ada-dk.org/?page=news&news_id=352

If you want/need to learn how to make use of dynamic plug-ins, then look no further! In the excellent article Dynamic Plug-in Loading with Ada authors Cyrille Comar and Pat Rogers, both of AdaCore fame, dive into the world of dynamic plug-ins, object-oriented programming and a crude simulation of instruments on an automobile dashboard.

> Maintenance of high-availability systems (e.g., servers) requires the ability to modify, enhance, or correct parts of the application without having to stop and re-link the entire system. This capability is relatively straight-forward with interpreted languages or virtual-machine based languages such as Java, in which new code is loaded upon demand.
>
> In languages typically implemented with static executable images this capability can be offered though dynamically loaded/linked libraries (“DLLs”).

However, in practice it is impractical to make full use of this capability because the protocol for invoking subprograms in a DLL is very low-level and unsafe. In the case of Ada, global coherency requirements and elaboration ordering constraints add an additional degree of complexity over less strict/safe languages. Object-oriented programming makes this approach practical by using dynamic dispatching to invoke dynamically loaded functions with a more robust, high-level protocol. In an OO paradigm, a “plug-in” contains new classes that enrich the class set of the original application. Calls to subprograms in the shared library (plug-in) are done implicitly through dynamic dispatching which is much simpler, transparent to the programmer, type-safe, and more robust. This application note shows how a statically-typed, statically-built, object-oriented language such as Ada can make full use of the notion of dynamic plug-ins á la Java without relying on a comparatively inefficient virtual machine. We build an extensible application and illustrate adding new functionality at run-time, without first stopping execution, using plug-ins.

It's a very informative read, with lots of source code coupled with well-written explanations. Definitely worth your time, if you're interested in Ada, OO and dynamic modules/plug-ins.

If you don't care for PDF's, there's a somewhat abbreviated version of the article available at Dr.Dobb's.

[read the article at http://www.adacore.com/wp-content/uploads/2005/04/dynamic_plugin_loading_with_ada.pdf or http://drdobbs.com/architecture-and-design/184407854 —mp]

Videos of industrial presentations at the Ada Connection 2011 conference

From: AdaCore's website
Date: Thu, 10 Nov 2011 [fetched]
Subject: The Ada Connection 2011
URL: http://www.adacore.com/home/ada_answers/lectures/ada-connection-2011/

A series of talks from the Ada Connection 2011 conference in Endinburgh, Scotland. The Ada Connection, which combines the 16th International Conference on Reliable Software Technologies — Ada-Europe 2011 — with Ada Conference UK 2011, sees a union of two Ada events that have both been very successful in their own right. The Ada-Europe series of conferences has become established as an international forum for providers, practitioners and researchers in all aspects of reliable software technologies.

Contents of Videos:

A new video will be added every Monday

- Design and Implementation of a Ravenscar Extension for Multiprocessors
- Implementing a Software Product Line for a complex Avionics System
- An Overview of DO-178C/ED-12C
- Ada based Automatic Code Generation Tools in DO178B context
- Detecting High-Level Synchronization Errors in Parallel Programs
- Real Time Longevity

[…]

Ada 2012 draft reference manual available

From: Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be>

Date: Thu, 10 Nov 2011 01:11:00 +0100
Subject: Ada 2012 - complete draft available (fwd)
Mailing list: ada-belgium-info.cs.kuleuven.be

[…]

Dear Ada-Belgium member,

FYI: the latest Ada 2012 draft reference manual is now available for your review. You can find the document at

http://www.ada-auth.org/standards/ada12.html.

As mentioned on that web-page:

"This is draft 14. This is the National Body review draft, and includes essentially all of the changes that will make up Ada 2012. All known issues with previous drafts have been addressed."

[…]

At the upcoming WG9 meeting tomorrow Thursday November 10, 2011, in Denver, CO, USA, we will discuss the review period within WG9 and plan the progress of the proposed new standard up to the next ISO level, i.e. SC22.

If you have any questions or comments, feel free to contact me.

Dirk Craeynest

ISO/IEC JTC1/SC22/WG9, Representative for Ada-Europe

ISO/IEC JTC1/SC22/WG9, former Head of Belgian Delegation, 2004-2010

Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/-Europe/SIGAda/WG9 mail)

[…]

Ada-related Tools

Dequesterity 1.0

From: Brad Moore <brad.moore@shaw.ca>
Date: Wed, 21 Sep 2011 23:26:37 -0600
Subject: Announce: Dequesterity - Ada 2005 Buffer container suite of generics
Newsgroups: comp.lang.ada

I am pleased to announce the initial release of Dequesterity.

Dequesterity is a set of Ada 2005 generics that provide various forms of general purpose buffer containers. Buffers may be used as deques, queues, ring buffers, stacks, double ended stacks, vectors, and similar abstractions.

The generics are combinable and pluggable such that lower level buffer implementations may be combined with higher level buffer generics to create a wide selection of buffer types with specific sets of functionality.

Lower level buffer implementations include bounded and unbounded buffer forms.

Higher level buffer implementations add concurrency support, and streaming capabilities, including Ravenscar-compliant buffer forms.

A Passive Buffer provides capabilities for deadlock detection, as well as seamlessly managing oversized requests (Read and Write requests that are larger than the buffer). The oversized requests are blocked until successful, and the transfer occurs automatically in the background without requiring any additional tasks.

Buffer instances may be streamed, or may be accessed remotely using the Distributed Systems Annex.

The Stream Buffer forms allow heterogeneous objects to be read and written to the buffer. A Ravenscar Stream Buffer allows a producer and a consumer task to stream heterogeneous objects.

Most buffers can store their state persistently. Some buffer implementations operate entirely on secondary (file based) storage.

The buffers may be instantiated with user defined types, and indefinite buffer forms also exist.

The interface to the buffers is modeled after the Ada 2005 container library.

Some might recall papers presented at SIGAda 2008 discussing the buffers. I finally got around to creating a release for them.

Any comments on the generics would be greatly appreciated.

Please send comments to brad.moore@shaw.ca

0.0 DOWNLOADING
==============

The latest stable release and older releases may be downloaded from;

https://sourceforge.net/projects/dequesterity/files/

For those who want the current development versions of the source they can download using git (http://git-scm.com/) by issuing the following commands

```
mkdir sandbox
cd sandbox
git clone
git://dequesterity.git.sourceforge.net/
    gitroot/dequesterity/dequesterity
```

The current development version typically will correspond to the latest stable release, but may at times be unstable when new features are being worked on.

Simple components for Ada v3.12

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 22 Oct 2011 16:55:19 +0200
Subject: ANN: Simple components for Ada v3.12
Newsgroups: comp.lang.ada

The library provides implementations of smart pointers, directed graphs, sets, maps, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support.

New version provides implementations of streams.

1. The package Block_Streams provides a stream built upon a stream of blocks transported over another stream. The stream can be used to store binary data in files access using Stream_IO or for sending data over a socket.
2. The package Storage_Streams provides a memory-resident stream. The memory is allocated by blocks.

[see also "Simple components for Ada v3.10" in AUJ 31-4 (Dec 2010), p.229 —mp]

Bare metal GNAT compiler targeting ARM

From: "Archeia" blog
Date: Thu, 29 Sep 2011
Subject: Bare metal ARM GNAT compiler built
URL: http://www.archeia.com/bare-metal-arm-gnat-compiler-built.html

After a multitude of different builds and a few modifications to the GNAT runtime, I have finally managed to build GCC-4.6.1 C and GNAT compilers for bare metal.

The build utilises Newlib as the libc interface that GNAT's RTS builds upon, I've disabled sockets, files and a few other things we don't need. I've added:

- system-bare-armel.ads as the bare metal system package (this can be used for any platform with a few modifications),

-mlib-specific-bare.adb, to enable the building of static libs within the arm-none-eabi-* tools from project files. This has also been tested

with my minimal runtime using gnat.gpr, which I was told by Arno Charlet that it wouldn't work.

[http://gcc.gnu.org/bugzilla/show_bug.cgi?id=47717#c6 —mp]

This is really good progress and also gives me some insider knowledge of GNAT, which isn't pretty!

I think I will use this new information to also build a GNAT to target the chipKIT Uno32 which is based on a PIC32 (MIPS core).

Support for tasking in AVR-Ada

From: Pablo Rego <pvrego@gmail.com>
Date: Mon, 24 Oct 2011 11:14:23 -0700
Subject: AVR-Ada Tasking support
Newsgroups: comp.lang.ada

Does someone know why tasking isn't supported by avr-ada? Is it yet under development? I read in some place in sourceforge project that there is the intention to turn avr-ada in a run-time system, it would be very good.

From: KK6GM <mjsilva@scriptoriumdesigns.com>
Date: Tue, 25 Oct 2011 14:53:19 -0700
Subject: Re: AVR-Ada Tasking support
Newsgroups: comp.lang.ada

That would be a fascinating development if it happened. I have no idea what subset e.g. Ravenscar? could be reasonably ported to AVR, but I would sure use it. With the Arduino phenomenon AVR has a lot of exposure these days. It would be a good place for Ada to get some notice.

Ada w/ simple tasking on AVR and ARM Cortex M3. Is that hoping for too much?

From: Pablo Rego <pvrego@gmail.com>
Date: Wed, 26 Oct 2011 04:06:58 -0700
Subject: Re: AVR-Ada Tasking support
Newsgroups: comp.lang.ada

[…]

That's the point. I receive tons of posts from Instructables, and several other DIY sites with applications for Arduino, Boarduino, and other AVR models every day, and almost all that I look inside could be improved if developers could use Ada instead of C from scratch.

I got to build AVR-Ada recently for an 8-bit AVR, but sadly discovered that the tasking feature was not enabled yet. Anyway, it looks that we can program AVR in Ada also using RTEMS (which I began to try), but AVR-Ada looks easier to coding, so I'd prefer to use AVR-Ada (at least until I don't get to build with RTEMS)

[…]

From: Tero Koskinen <tkoskine@kapsi.fi>
Date: Wed, 9 Nov 2011 18:11:42 +0200
Subject: Re: AVR-Ada Tasking support
Newsgroups: comp.lang.ada

[…]

The development version of AVR-Ada includes bindings to avr-threads:

http://avr-ada.git.sourceforge.net/git/gitweb.cgi?p=avr-ada/avr-ada;a=blob;f=avr/avr_lib/avr-threads.ads;h=635c5ea92aba2621cc4d7c98085033645b7ed6a6;hb=HEAD

It isn't as easy to use as native Ada tasks, but at least you don't need to implement tasking by yourself.

[see also "AVR-Ada and GCC 4.5.0" in AUJ 32-1 (Mar 2011), p.9 —mp]

VTKAda 5.9

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Wed, 31 Aug 2011 18:50:59 -0700
Subject: Announce: VTKADA 5.9
Newsgroups: comp.lang.ada

I'm pleased to announce

VTKAda version 5.9 release 1 free edition (work in progress) VTKAda is and Ada-95(2005,2012) port to VTK (Visualization Toolkit by Kitware, Inc) and Qt4 application and UI framework by Nokia.

[…]

VTKAda is a powerful 2D, 3D rendering and imaging system and works inside Qt4 applications.

[…]

VTKAda and QtAda for Windows and Linux (Unix) free edition are available from

http://users1.jabry.com/adastudio/index.html

[…]

[see also "VTKAda 5 release 4" in AUJ 32-1 (Mar 2011), p.12 —mp]

AWS on Mac OS X

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: Wed, 05 Oct 2011 13:42:45 +0200
Subject: AWS on Mac OS X?
Newsgroups: comp.lang.ada

Is there a special reason that AdaCore doesn't distribute AWS for Mac OS X? I would expect AWS to be rather portable, so it worries me a bit, that I can't find AWS in the list of downloads for Mac OS X on the libre.adacore.com website.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 5 Oct 2011 13:16:04 -0700
Subject: Re: AWS on Mac OS X?
Newsgroups: comp.lang.ada

[…]

Yes, there are two reasons:

1. git clone as shown on http://libre.adacore.com/libre/tools/aws/
2. make setup build install

Or something like that. It worked for me.

I have to admit that it was much more involving on Windows, as it required the installation of Cygwin. Then I understand that already compiled binary packages provide a lot of added value. But interestingly, they are not available on Windows, either. Or at least I was not able to find them.

From: Simon Wright <simon@pushface.org>
Date: Thu, 06 Oct 2011 12:54:03 +0100
Subject: Re: AWS on Mac OS X?
Newsgroups: comp.lang.ada

[…] you can download the source from the Linux or Windows variants (neither seems to supply binary, anyway!)

It builds on Lion with GNAT GPL 2011 and (after a battle which I can tell you about if you need to know) with GCC 4.6.0; 'make check' fails in both cases (in gnatcheck, I think, complaining about "implicit IN mode in parameter specification" -- I would rather complain about *explicit* IN mode but there you go).

[…]

QtAda 3.2.0 for GNAT GPL 2011

From: Vadim Godunko <vgodunko@gmail.com>
Date: Sat, 13 Aug 2011 00:08:19 -0700
Subject: Announce: QtAda for GNAT GPL 2011
Newsgroups: comp.lang.ada

QtAda 3.2.0 preview for GNAT GPL 2011 is available for download.

Source code:

http://download.qtada.com/qtada-gpl-3.2.0-20110812-3857.tar.gz

Prebuild Microsoft Windows package:

http://download.qtada.com/qtada-gpl-3.2.0-20110812-3857-qt4.7.3-1.exe

[see also "QtAda 3.1.0" in AUJ 31-4 (Dec 2010), p.230 —mp]

TclAdaShell 20110925

From: Simon Wright <simon@pushface.org>
Date: Sun, 25 Sep 2011 12:30:57 +0100
Subject: ANN: tcladashell 20110925
Newsgroups: comp.lang.ada

This maintenance release of TclAdaShell [1] has the following changes:

The ClientData generics in Tcl.Ada had comments stating that the size of the ClientData formal type must be equal to the size of a C pointer. These have been replaced by assertions that the size of ClientData must not be greater than that of System.Address.

The top-level makefile now supports an 'install' target which on GNAT-based systems other than Debian installs TASH alongside your compiler (so you don't need to set ADA_PROJECT_PATH).

The setup.tcl script recognises gnatgcc, if present, as the compiler to use for the C compilations required to build the library.

The setup.tcl script supports the flag "--nogui", meaning "perform the setup immediately".

The GPR files have been improved; the result is that the Tcl and Tk libraries will be linked automatically.

[1] https://sourceforge.net/projects/tcladashell/files/source/20110925/

[see also "TclAdaShell 20090611" in AUJ 30-3 (Sep 2009), p.146 —mp]

GtkAda contributions v2.10

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 6 Nov 2011 18:21:11 +0100
Subject: ANN: GtkAda contributions v2.10
Newsgroups: comp.lang.ada

This version is compatible with the newest version 2.18 of GtkAda.

Further changes are:

- The procedure Set was added to the package Gtk.Handlers.References for explicit unsetting references;
- The procedures Has_Tooltip, Set_Has_Tooltip, Set_Tip were added to Gtk.Missed;
- Gtk.Handlers.Generic_Callback was added to support signal handlers with return values handled as GValue;
- The package Gtk.Recent_Manager was renamed to Gtk.Recent_Manager_Alt to keep it compatible to GtkAda 2.18.0. For earlier versions of GtkAda, renaming packages are provided for backward compatibility;
- The Gtk.Object.Checked_Destroy procedure was added to safely destroy floating widgets;
- The package Gtk.Recent_Manager_Keys now provides simplified means to store and restore values by key, store and restore contents of combo boxes.

[see also "GtkAda contributions v2.9" in AUJ 30-1 (Mar 2011), p.12 —mp]

SOCI 3.1.0

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 11 Oct 2011 07:02:27 -0700
Subject: SOCI 3.1.0 released, SOCI-Ada merged
Newsgroups: comp.lang.ada

I am pleased to announce that the 3.1.0 release of SOCI is available for download.

SOCI is a database access library for C++ that is recognized for its ease of use and natural API.

The SOCI-Ada project existed as a separate package and provided the Ada binding to the SOCI library with the advantage of reusing the constantly-growing set of backends developed withing the context of the main project.

The SOCI-Ada project is going to be discontinued as it was effectively merged with the main SOCI project. That is, Ada programmers that want to access databases using SOCI can now use a single and consistent software package.

The home page of the SOCI project is:

http://soci.sourceforge.net/

The documentation related to the SOCI-Ada part is available at:

http://soci.sourceforge.net/doc/languages/ada/index.html

Don't hesitate to contact me in case of any questions related to the build process. It has been tested with recent GNAT versions and appropriate project files are provided, but some customization might be necessary depending on how GNAT itself was installed and how it relates to the C++ compiler on the given target system.

[see also "SOCI-Ada — Database Access Library" in AUJ 29-3 (Sep 2008), p.153 —mp]

GNATColl and PostgreSQL stored procedures and non-public schemas

From: Thomas Løcke <tl@ada-dk.org>
Date: Thu, 03 Nov 2011 20:34:50 +0100
Subject: GNATColl and support for PostgreSQL stored procedures and non-public schemas
Newsgroups: comp.lang.ada

[…]

Are there any plans for adding support for non-public schemas to the gnatcoll_db2ada tool? Right now we have to resort to embedded SQL instead of using the excellent GNATCOLL.SQL Ada style SQL interface.

[…]

And how about stored procedures? We use those A LOT in our application but we haven't been able to figure out how to make use of them without having to go back to plain old embedded SQL, which of course is sub-optimal.

[…]

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Fri, 4 Nov 2011 01:36:06 -0700
Subject: Re: GNATColl and support for PostgreSQL stored procedures and non-public schemas
Newsgroups: comp.lang.ada

[…]

No such plan at this stage. Presumably, adding those should not be too difficult: in the text file, the name of the tables would be "schema.name". At this point, gnatcoll_db2ada needs to issue a "CREATE SCHEMA schema" command. The second part of it is to replace "." by some other substring for the Ada identifiers that are generated to represent the table.

[…]

For aggregate functions, you can simply define new constants similar to GNATCOLL.SQL.Func_Count. For other types of stored procedures, you could copy the implementation of GNATCOLL.SQL.Lower, for instance, and perhaps even make it more general so that it takes the name of the procedure in parameter.

Fortran to Ada 95 converter 1.2

From: Oliver M. Kellogg <okellogg@users.sourceforge.net>
Date: Mon, 31 Oct 2011 03:42:47 -0700
Subject: Fortran to Ada95 converter update
Newsgroups: comp.lang.ada

The f2a.pl semi-automatic Fortran to Ada 95 converter perl script has been updated due to user feedback.

The new version is available at

http://www.okellogg.de/x.html

http://www.okellogg.de/for2ada95-1.2.tar.gz

Here are the changes of version 1.2:

- Added keywords REWIND and PRINT to @tbd.
- Translate STOP to (commented) System.OS_Lib.OS_Exit. The call is commented because of its GNAT dependency.
- Use artificial name "Main" for main procedure if the PROGRAM statement is missing in the input code.
- Print line number in input file on error message.
- New variables $linebuf_save, $line_pending_save, and $linenum_save do the necessary saving of global state before processing of INCLUDE file.
- Print success status and file names generated upon completion of conversion.

Update on the support for Ada on Android platforms

From: John Marino <dragonlace.cla@marino.st>
Date: Sat, 20 Aug 2011
Subject: Zero Testsuite Failures for GNATDroid
URL: http://www.dragonlace.net/posts/Zero_Testsuite_Failures_for_GNATDroid/

I finally acquired an Android device, a nice ASUS Transformer TF101 equipped with an NVIDIA Tegra2 dual-core CPU. After building GNATDroid-ARMv7, I confirmed that I could compile Ada programs on FreeBSD and execute them on the Android tablet.

After some trial and error, I modified the ACATS test suite to execute the tests remotely on the transformer. After the first run, 57 tests failed with the same error. It turns out that the default location for temporary files (/tmp) doesn't exist on Android. I patched GNATDroid to first attempt to create temporary files at $ANDROID_DATA/local/tmp (/data/local/tmp) which normally requires a rooted device, and then try $EXTERNAL_STORAGE/ (/sdcard) which would require that the user permissions can write to that area.

With that update, the temporary files could be created and the GNATDroid passed the ACATS testsuite without a single failure. Modifying Dejagnu test harness is a little trickier, so the gnat.dg testsuite still hasn't been run.

In any case, the confidence in this cross-compiler is now quite high and these ports can be officially submitted to FreeBSD.

The tarball and signature file referenced in the previous post have been updated to include the patch for adaint.c which controls the temporary file creation.

If you have already built GNATDroid, you may wish to deinstall it, re-extract the files, and then rebuild it.

From: John Marino <dragonlace.cla@marino.st>
Date: Wed, 7 Sep 2011
Subject: GNATDroid incorporated into FreeBSD
URL: http://www.dragonlace.net/posts/GNATDroid_incorporated_into_FreeBSD/

After the GNATDroid compiler passed the ACATS testsuite flawlessly, it was submitted officially to the FreeBSD ports tree. It can sometimes take months for new ports to make it through the submission process, but the GNATDroid ports got approved and committed within a week. We are very grateful to FreeBSD committer Frederic Culot for the special attention he affords these Ada ports, and we can't express our appreciation enough.

So now FreeBSD users can obtain the ports through the normal method (e.g. portsnap fetch update) and install the cross-compiler like any other FreeBSD software (e.g. cd /usr/ports/lang/gnatdroid-armv7 && make install).

Now its up to you guys to utilize the strength of Ada on the Android platform.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Fri, 9 Sep 2011 13:29:06 +0000
Subject: Re: GNATDroid (Ada cross-compiler for ARM/Android) incorporated into FreeBSD ports tree
Newsgroups: comp.lang.ada

[…]

I ran "portsnap fetch" and saw these just before you posted!

GNATDroid installs cleanly here, and builds a working "hello world" executable. Tests here so far are limited to copying the executable onto a USB drive, then onto the tablet, and running it within the terminal emulator. Now on to more interesting examples…

From: John Marino <dragonlace.cla@marino.st>
Date: Fri, 9 Sep 2011 10:26:52 -0700
Subject: Re: GNATDroid (Ada cross-compiler for ARM/Android) incorporated into FreeBSD ports tree
Newsgroups: comp.lang.ada

[…]

I used the SSHDroid application to set up an SSH Daemon on my tablet, then used SCP to copy the Ada binaries over. You can even execute them remotely from the host machine using SSH. This is the technique I used to run through the ACATS testsuite. There's some lag, but much better than breaking out a USB stick.

[see also "Ada for Android" in AUJ 32-3 (Sep 2011), p.136 —mp]

Ada on Solaris

From: John Marino <dragonlace.cla@marino.st>
Date: Wed, 5 Oct 2011
Subject: GNAT-AUX on Solaris
URL: http://www.dragonlace.net/posts/GNAT-AUX_on_Solaris/

Unfortunately, OpenSolaris died when Oracle took over Sun Microsystems.

However, out of the ashes rose the Illumos project [http://www.illumos.org/ —mp], which took over powering platforms like Nexenta [http://www.nexenta.org/ —mp] and spawned new distributions such as OpenIndiana. [http://openindiana.org/ —mp]

The Illumos-powered platforms gained Pkgsrc through the Illumos Pkgsrc Project, which boasted an impressive 6600 built packages soon after getting launched.

GNAT-AUX was already available on Pkgsrc, but it only works on platforms in which a bootstrap compiler is provided. Before this week, this covered exactly four platforms:

x86 NetBSD

x86 DragonFly

x86_64 NetBSD

x86_64 DragonFly

I had previously built a flawless GNAT-AUX on an obsolete OpenSolaris version SXCE 130. It was fairly trivial to built a static bootstrap compiler on the latest version of OpenIndiana (oi_151a).

The lang/gnat-aux package has been updated to support the x86 Solaris target, and also updates all supported platforms to provide working runtime symbolic traceback. It just missed the 2011Q3 branch, so it's only available in the pkgsrc trunk.

Ahven 2.0 and 2.1

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: 24 Sep 2011 04:54:51 GMT
Subject: Announce: Ahven 2.0 and Ahven 2.1
Newsgroups: comp.lang.ada

Hi,

I released two versions of Ahven today, Ahven 2.0 and Ahven 2.1.

Ahven 2.0 introduces two new features: timeouts and test skipping; and Ahven 2.1 fixes a bug in the skipped test reporting.

You can get the source code from SourceForge:

https://sourceforge.net/projects/ahven/files/

Changelog:

2011-09-24 Ahven 2.1

Bugs fixed

- Ahven.Text_Runner did not report skipped tests correctly.

  This is now fixed.

Internal

- Function Ahven.Results.Skipped_Count was added.

2011-09-23 Ahven 2.0

Changes

- Tests can be now given a timeout value. If a test is not executed in the given time, it is stopped and a timeout failure is reported. See '-t' option of the test runners.

The timeout feature depends on the possibility to abort a task after a certain amount of time. If the task abortion is not possible, the current test will continue running even after the given timeout.

- A test can be now skipped programmatically by calling procedure Skip("Message"). A skipped test are considered to be equal to passed tests, but depending on the test runner, they can have extra "SKIP" information attached.
- README is now provided in reStructured text format, just like the manual.

Bugs fixed

- Ahven can be compiled on Fedora systems by installing package "libgnat-static". Note: This was not a bug in Ahven but a configuration issue on Fedora.

About Ahven:

Ahven is a simple unit test library (or a framework) for Ada programming language. It is loosely modelled after

JUnit and some ideas are taken from AUnit.

Ahven is free software distributed under permissive ISC license and should work with any Ada 95 or 2005 compiler.

Homepage:

http://ahven.stronglytyped.org/

[see also "Ahven 1.9" in AUJ 32-2 (Jun 2011), p.75 —mp]

Deepend 2.6

From: Brad Moore <brad.moore@shaw.ca>
Date: Sun, 06 Nov 2011 21:33:41 -0700
Subject: Announce: Deepend 2.6 for GNAT and ICC Ada 2005 compilers
Newsgroups: comp.lang.ada

I am pleased to announce the availability of Deepend 2.6.

Deepend is an efficient and safer form of storage management for Ada 2005 that can outperform garbage collection schemes.

Since the initial (previous) announcement of Deepend on comp.lang.ada, there have been a number of improvements.

Some of these include:

1) When Deepend was first announced, it was a binding to the Apache run time pool library. Since then, the Apache library has been removed as a dependency, and Deepend is now 100% pure Ada. Testing has shown that the new version of Deepend runs noticeably faster than the earlier version that called out to the Apache library.

2) Deepend and the Irvine ICC Ada 2005 compiler

Deepend has been compiled and tested using the Irvine ICC Ada 2005 compiler, running on Windows and purportedly on Linux. See http://www.irvine.com for more information about their compiler

3) Deepend on an Android Samsung Galaxy S II smart phone

Deepend has been compiled and tested using GNAT AUX on an Android Samsung Galaxy S II Smart phone.

See http://www.dragonlace.net for more information about this compiler

4) Deepend and the GNAT 2011 GPL compiler

Deepend has been compiled and tested using GNAT 2010 and 2011 GPL versions of the compiler on both Windows and Linux. See http://libre.adacore.com/libre for more information about this compiler.

5) Deepend Aligned with the Ada 2012 subpools proposal.

Deepend provides two storage management options,

- Basic_Dynamic_Pools
- Dynamic_Pools

The Basic_Dynamic_Pools package is forward compatible with the Ada 2012 proposal for Storage_Pools, since it only allows allocations via the existing "new" operator. This facility relies on access type finalization to free all the objects from a pool.

Dynamic_Pools provides the capabilities of Basic_Dynamic_Pools, but in addition allows the creation of subpools, and allocations can be made from subpools. Subpools can be deallocated, which deallocates all objects allocated from the subpool. The Dynamic_Pools package is designed to closely align with the Ada 2012 proposal, except that it works for Ada 2005, and doesn't support deallocation of fat pointers, or controlled types. When Ada 2012 is available, the interfaces may change to match the proposal, and capabilities offered by Ada 2012.

Deepend is a dynamic storage pool with Subpool capabilities for Ada 2005 where all the objects in a subpool can be reclaimed all at once, instead of requiring each object to be individually reclaimed one at a time. A Dynamic Pool may have any number of subpools. If subpools are not reclaimed prior to finalization of the pool, then they are finalized when the pool is finalized.

Rather than deallocate items individually which is error prone and subceptable to memory leaks and other memory issues, a subpool can be freed all at once automatically when the pool object goes out of scope.

With this Storage pool, Unchecked_Deallocation is implemented as a No-Op (null procedure), because it is not needed or intended to be used.

Subpool based storage management provides a safer means of memory management, which can outperform other mechanisms for storage reclamation including garbage collection.

You can get the source code of Deepend from SourceForge:

https://sourceforge.net/projects/deepend/files/

If performance is a desired goal, you may also want to check out Paraffin, which provides Ada 2005 generics to add Parallelism to loops and recursive algorithms. Paraffin and Deepend complement each other for obtaining faster execution times.

for Paraffin,

see https://sourceforge.net/projects/paraffin/files/

From: Brad Moore <brad.moore@shaw.ca>
Date: Mon, 07 Nov 2011 10:37:18 -0700
Subject: Re: Announce: Deepend 2.6 for GNAT and ICC Ada 2005 compilers
Newsgroups: comp.lang.ada

A minor correction to the announcement.

I said:

> The Dynamic_Pools package is designed to closely align with the Ada 2012 proposal, except that it works for Ada 2005, and doesn't support deallocation of fat pointers, or controlled types.

Actually, the limitation is that fat pointers (access to objects of unconstrained types) cannot be allocated to subpools, but they can be allocated to a deepend pool using the "new" operator. Controlled types can be allocated to subpools, but it is erroneous to cause the finalization of these objects to occur before they would have otherwise been finalized, such as when the access type is finalized. It is otherwise OK to allocate controlled types to subpools, but recommended that they instead be allocated to the pool, using the "new" operator. Similarly, allocated task objects cannot be finalized earlier than when they would have ordinarily been finalized.

Ada 2012 will provide new syntax for the "new" operator to allow objects of unconstrained types to be allocated to subpools, as well as expose machinery that will allow objects of controlled types to be finalized when a subpool is finalized. The limitation for allocated task objects will remain however.

[see also "Storage pool with bindings to Apache Runtime Pools library" in AUJ 32-2 (Jun 2011), p.73 and "Paraffin" in AUJ 32-1 (Mar 2011) —mp]

Ada-related Products

AdaCore — GPS 5.1

From: AdaCore Press Center
Date: Tue, 27 Sep 2011
Subject: AdaCore Releases Major New Version of GNAT Programming Studio
URL: http://www.adacore.com/2011/09/27/gps5-1/

GPS 5.1 Integrated Development Environment brings new C/C++ features, improved support for CodePeer, and more powerful source editing.

BOSTON, Mass., NEW YORK and PARIS, September 27, 2011 – Embedded Systems Conference – AdaCore, a leading supplier of Ada language tools and support services, today announced the upcoming release of GNAT Programming Studio (GPS) 5.1. This new major version of AdaCore's graphical Integrated Development Environment (IDE), to be available in October, offers extended feature support for C and C++, improved integration with CodePeer (automated code reviewer and validator), more powerful source editing, and enhanced GUI performance. GPS is provided with

GNAT Pro on most platforms, for both native and embedded software development.

"This new version of GPS strengthens support for tools and processes that are important for full life cycle application development," said Arnaud Charlet, GPS Project Manager at AdaCore. "Large applications are almost always developed using multiple languages, so we have extended GPS's facilities in this area. Large or long-lived applications need analysis tools and configuration management system integration, so we have also improved GPS's support here. This new version is simply more powerful in areas that our customers have asked for."

Enhancements in GPS 5.1 include:

- Improved support for C and C++:
  - o Smart completion for C and C++ using -fdump-xref info
  - o Ada-to-C source navigation
- Improved CodePeer support :
  - o Availability of score card feature
  - o Improved filtering
  - o Locations view now synched with CodePeer report
  - o Ability to specify alternate database/output directories
  - o Availability of race condition report
- New facility for handling VCS menus:
  - o All VCS menus are now handled in a centralized place allowing customization of the layout of all VCS menus
- Availability of additional automatic code fixes
- Enhanced documentation generation:
  - o Ability to export browser contents to PDF
- Improved GUI integration and performance:
  - o Enhancement of multiple document interface (MDI), search support, and code browsers.

GPS 5.1 is compatible with GNAT Pro versions 3.16a1 up to 6.4. As with all GNAT Pro components, GPS is distributed with full source code and is backed by AdaCore's rapid and expert online support.

About GNAT Programming Studio (GPS)

GPS is a powerful Integrated Development Environment (IDE) written in Ada using the GtkAda toolkit. GPS's extensive source-code navigation and analysis tools can generate a broad range of useful information, including call graphs, source dependencies, project organization, and complexity metrics. It also supports configuration management through an interface to third-party Version Control Systems, and is available on a variety of platforms. GPS is highly extensible; a simple scripting approach enables additional tool integration. It is also customizable, allowing programmers to specialize various aspects of the program's appearance in the editor for a user-specified look and feel.

Pricing and Availability

GPS 5.1 is included with the GNAT Pro Ada Development Environment as well as the SPARK Pro and CodePeer Pro toolsets, and customers can download it via the GNAT Tracker tool.

Webinar

A webinar focusing on the new features of the GPS 5.1 release will be presented later this year. For schedule and other information, or to register, please visit GNAT Pro Webinars

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial software solutions for Ada, a modern programming language designed for large, long-lived applications where safety, security, and reliability are critical. AdaCore's flagship product is the GNAT Pro development environment, which comes with expert on-line support and is available on more platforms than any other Ada technology. AdaCore has an extensive worldwide customer base; see http://www.adacore.com/home/company/customers/ for further information.

Ada and GNAT Pro continue to see growing usage in high-integrity and safety-certified applications, including commercial aircraft avionics, military systems, air traffic management/control, railway systems and medical devices, and in security-sensitive domains such as financial services.

AdaCore has North American headquarters in New York and European headquarters in Paris. www.adacore.com

Press Contacts

press@adacore.com

[…]

Inspirel — YAMI4 1.4.0

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 14 Sep 2011 14:41:45 -0700
Subject: YAMI4 1.4.0 released
Newsgroups: comp.lang.ada

I am pleased to announce that the 1.4.0 version of YAMI4 is available for download:

http://www.inspirel.com/yami4

As the most significant contribution, this new release brings a GUI management console that allows to browse and manage the name servers, message brokers and individual agents in a bigger distributed system.

The GUI console, called YAMI4 dashboard, is based on HTML and can be used with any web browser.

An example screenshot showing the output of ping operation and traffic statistics of simple data publisher server is shown here:

http://www.inspirel.com/yami4/dashboard.png

Ada programmers will find it interesting that the dashboard was implemented in terms of AWS. Its ready to use binary versions, together with all other central services, are available for Linux and Windows.

Other library improvements for Ada and C++ include the possibility to monitor internal events with standard implementation of simple statistics monitor that can be inspected remotely.

Note: even though the YAMI4 library was tested with a range of GNAT compiler versions, the statistics monitor requires GNAT 2011. Please contact Inspirel for assistance if porting to older compilers is needed.

All comments are welcome.

Vector Software — VectorCAST 5.3

From: Vector Software Press Release
Date: Wed, 14 Sep 2011
Subject: Vector Software announces enhanced functionality in 5.3 release of VectorCAST
URL: http://www.vectorcast.com/news/2011/press-release-vectorcast-5.3.php

Newest version of VectorCAST includes open support for multiple requirements management tools

Providence, RI – 9/14/2011 - Vector Software, Inc., the leading provider of dynamic software test tools for safety-critical embedded applications, announced today the release of VectorCAST version 5.3. This latest release of VectorCAST features a significant redesign to VectorCAST/Requirements Gateway. Previous releases of VectorCAST/Requirements Gateway supported only IBM® Rational® DOORS®. The re-design allows the Requirements Gateway tool the ability to capture and manage requirements from any management tool.

In addition, VectorCAST 5.3 includes several new features and an expanded set of enhancements to existing functionality.

VectorCAST 5.3 new capabilities and enhancements

- New Graphical Control Flow Editor

- New Test Case Tree and Test Case Tree Editor functionality
- New tools for easier editing of the Link Options
- Coverage Data Storage and Reporting re-architected using SQL
- Aggregate Coverage Data and Viewer added to VectorCAST/Manage
- Updated Rhapsody Integration using the Rhapsody Java API
- Flexible DSP Bios Support for TI Code Composer

VectorCAST 5.3 is immediately available to customers worldwide. For additional details about the 5.3 release of VectorCAST or to register for the What's New in VectorCAST 5.3 webinar, please visit: www.vectorcast.com/vectorcast-5.3

About Vector Software, Inc.

Founded in 1989, Vector Software, Inc. is the leading independent provider of automated software testing tools for developers of safety critical embedded applications. Vector Software's VectorCAST line of products, automate and manage the complex tasks associated with unit, integration, and system level testing. The VectorCAST tools support the C, C++, and Ada programming languages.

Vector Software's Product Family

VectorCAST/C++

VectorCAST/Ada

VectorCAST/RSP

VectorCAST/Cover

VectorCAST/Manage

VectorCAST/Requirements Gateway

Modified Condition / Decision Coverage (MC/DC) module

IEC 61508 and ISO 26262 Certification Kits

DO-178B and FDA Qualification Packages

Vector Software — Support for RTEMS in VectorCAST

From: Vector Software Press Release
Date: Mon, 3 Oct 2011
Subject: Vector Software expands support for the RTEMS Real-Time Operating System
URL: http://www.vectorcast.com/news/2011/press-release-rtems-support-vectorcast.php

Latest VectorCAST integrated solution targets European Space Agency aerospace and defense applications

Providence, RI – 10/3/2011 - Vector Software, the leading provider of dynamic software test tools for embedded systems, has integrated the VectorCAST tool suite with the Real-Time Executive for Multiprocessor Systems (RTEMS). Managed by OAR Corporation, RTEMS is a full featured RTOS (Real-Time Operating System) that supports a variety of open API and interface standards. RTEMS is specifically designed for deeply embedded systems and used on several European Space Agency (ESA) projects.

"We are delighted to announce support for the Real-Time Executive for Multiprocessor Systems Operating System", said William McCaffrey, chief operating officer for Vector Software, "This latest integration offers our ESA program customers many improvements and enhancements."

[…]

Ada and GNU/Linux

Matreshka included in Fedora 15

From: Vadim Godunko <vgodunko@gmail.com>
Date: Wed, 7 Sep 2011 00:33:59 -0700
Subject: Announce: Matreshka in Fedora
Newsgroups: comp.lang.ada

Matreshka was included into Fedora 15 release as part of Ada Developer Tools.

Matreshka is a set of libraries to help to develop general purpose Ada applications. It includes Unicode based support for localization and globalization, XML reader and writer, FastCGI and generic SQL database access. For more information please visit:

http://forge.ada-ru.org/matreshka/wiki

[see also "Matreshka 0.1.1" in AUJ 32‑3 (Sep 2011), p.136 —mp]

Support for multilib in GNAT on Debian

From: awdorrin <awdorrin@gmail.com>
Date: Thu, 3 Nov 2011 09:24:53 -0700
Subject: Gnat on Debian 6.0.3 building 32-bit executables?
Newsgroups: comp.lang.ada

I realize this may not be the best group for asking this question as it may be more of an OS question than an Ada question - but since its related to building Ada source code - I'm hoping it's ok I post this here.

I just reinstalled my system using Debian 6.0.3 for amd64/x86_64.

I need to compile my code into 32-bit. I see that the GCC compiler supports multilib - but I can't find the corresponding files for GNAT.

Is there a way to install the i386 files for GNAT on the amd64/x86_64 version of Debian, or would I be better off reinstalling Debian for i386?

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 03 Nov 2011 21:54:15 +0100
Subject: Re: Gnat on Debian 6.0.3 building 32-bit executables?
Newsgroups: comp.lang.ada

[…]

> I need to compile my code into 32-bit. I see that the GCC compiler supports multilib - but I can't find the corresponding files for gnat.

Actually, support for multilib in Ada isn't _quite_ there yet in 6.0.3, we're working on it for Debian 7 "Wheezy".

> Is there a way to install the i386 files for GNAT on the amd64/x86_64 version of Debian, or would I be better off reinstalling Debian for i386?

You're better off installing Debian for i386 in a chroot; for some details see http://lists.debian.org/debian-ada/2010/02/msg00003.html

References to Publications

High-integrity object-oriented programming with Ada

From: Thomas Løcke <tl@ada-dk.org>
Date: Wed, 24 Aug 2011
Subject: High-integrity object-oriented programming with Ada - Part 3
URL: http://ada-dk.org/?page=news&news_id=349

A few weeks ago I wrote about the High-integrity object-oriented programming with Ada articles by Benjamin Brosgol.

While I was away on vacation, part 3 was made available:

High-integrity object-oriented programming with Ada - Part 3

[http://www.eetimes.com/design/military-aerospace-design/4218480/High-integrity-object-oriented-programming-with-Ada---Part-3 —mp]

Be sure to check out these articles if you're interested in how Ada does OOP.

[see also "High-integrity object-oriented programming with Ada" in AUJ 32.3 (Sep 2011), p.139 —mp]

"Why Hi-Lite Ada?"

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Sat, 1 Oct 2011 11:17:00 CEST
Subject: "Why Hi-Lite Ada?" (paper)
Newsgroups: comp.lang.ada

An up-to-date paper about Ada, oriented towards Design By Contract (™) and proofs to be runtime-error-free. Noticeably, the paper is hosted at research.microsoft.com. It goes beyond just SPARK, by introducing some other ways to prove correctness of Ada programs (sometimesit is better to get multiple choices), including Alt-Ergo and the Why system. The paper was written in the context of a so called “First International Workshop On Intermediate Verification Languages”.

Date: 2011

Length: 13 pages

Authors: Jérôme Guitton, Johannes Kanig and Yannick Moy

Document: http://research.microsoft.com/en-us/um/people/moskal/boogie2011/boogie2011_pg27.pdf

"Should we put up with software that doesn't work?"

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 10 Oct 2011
Subject: Should we put up with software that doesn't work?
URL: http://ada-dk.org/?page=news&news_id=365

Robert Dewar from AdaCore begins his

"Should we put up with software that doesn't work?" article with a statement:

> We are used to software that dismally fails. What is surprising is that we accept this as reasonable. It is time to stand up and say we are not going to put up with this anymore. There is no excuse for junk software.

I could not agree more. It's scary that the software business as a whole have managed to convince the world that it's OK to deliver products that fail to function as advertized, and it's even more scary that users have accepted this as normal. It's a troubling development that buyers of software can expect less quality from expensive software than from cheap goods from other industries. Companies that manufacture and sell toasters are more liable in the world of 2011, than companies that manufacture million dollar software.

If software development is to be taken serious as both a science and a profession deserving of the term "engineering", then we need to change our ways. A consequence of this change might be that the current model of selling boxed software will have to end. The constant need for new versions and features push the limits of our ability to produce quality products. If instead software is sold on a subscription basis or as a service, then stability and polish becomes more important than going from version 3 to 4.

But obviously this all starts with the user. As long as the users silently accepts the current state of affairs, nothing is going to change.

But luckily we have guys like Dewar, who are at least trying, even though it does seem like a pretty small candle in a very dark and giant room. The article is very much worth reading, especially if you're in any way interested in software, which I suspect you are, since you're here.

[read the article at http://www.mil-embedded.com/articles/id/?5343 —mp]

Gem #113: Visitor Pattern in Ada

From: AdaCore's Website
Date: Mon, 7 Nov 2011
Subject: Gem #113: Visitor Pattern in Ada
URL: http://www.adacore.com/2011/11/07/gem-113-visitor-pattern-in-ada/

Author: Emmanuel Briot, AdaCore

Abstract: The visitor pattern is a design pattern that provides a way to execute specific methods on an object (the visitor) depending on the type of another object. Since the exact subprogram called depends on both types of the objects, this pattern is often called double dispatching.

[…]

[read the rest of the Ada Gem at the URL above. This Ada Gem was referred to in the news item "On multiple dispatch in Ada" in AUJ 32.3 (Sep 2011), p.151, but was still unpublished at that time —mp]

Ada Inside

Use of Ada for Digicomp's military and defense applications

From: AdaCore Press Center
Date: Tue, 27 Sep 2011
Subject: Digicomp Shows Continuing Success with Ada and GNAT Pro
URL: http://www.adacore.com/2011/09/27/digicomp-success/

BOSTON, Mass., NEW YORK and PARIS, September 27, 2011 – Embedded Systems Conference – AdaCore, provider of tools and expertise for the mission-critical, safety-critical, and security-critical software communities, today announced that Digicomp Research has reaffirmed their commitment to Ada based on their continuing success with the language, by renewing their long-standing subscription for the GNAT Pro Ada development environment. Digicomp, a system engineering and software development company specializing in military and defense applications, has been an AdaCore customer for more than a decade, using GNAT Pro to successfully implement and deploy a variety of mission-critical systems on Sparc Solaris, x86 Solaris, and Linux platforms.

[…]

Digicomp started using Ada in 1990 when the company was awarded a contract to build a new radar tracking, data fusion, and command and control system for use at Tyndall AFB (Florida). They were free to choose the programming language for the project, and they selected Ada for its expressiveness, its error detection, and its support for building software components and subsystems that can be combined without unstated coupling of modules.

Since that time, the company has successfully used the Ada language for multiple programs based on its proven ability to support development of reliable software in a cost effective manner. Core characteristics of the language allow for coding errors to be detected early in the software life cycle, when they can be corrected with the least cost.

Current Digicomp projects using Ada and GNAT Pro include: the Mode 4 Interrogation Support rack-mount computer system; the Air Surveillance and Control System, which is used for weapons control, bombing range safety, and surveillance; and the Operational Flight Program (OFP) for the Symbol Generator Unit (SGU) on the MH53-J helicopter.

“The Ada run-time software for the MH53-J includes multiple tasks to perform input and output from several sensors, to communicate with other aircraft systems over the MIL-STD-1553B data bus, and to update the symbology display at a 10 Hz frame rate,” Dan DeJohn continued. “Ada’s good information hiding facilitated writing an emulator for the final display hardware so that we could design, implement, and demonstrate the symbol generation prior to availability of the final system hardware. This inherent capability was invaluable as it allowed us to deliver on time despite delays in obtaining the final hardware.”

[…]

Indirect Information on Ada Usage

[Extracts from and translations of job-ads and other postings illustrating Ada usage around the world. —mp]

Job offer [Belgium]: Senior Ada 95 Developer

The successful candidate will be an expert in the following:

- Ada 95 programming
- Object-oriented analysis, design and programming
- HP-UX, Linux (or similar)
- Communication protocols such as TCP/IP
- Usage of development tools such as Unix scripting, Emacs, test tools, ClearCase, etc
- Design methodologies such as HOOD, BOOCH, UML

- Good knowledge of relational databases (e.g. Oracle)
- Good knowledge of tools such as Rational Rose
- Ability to produce technical documentation for computer systems
- Experience with the following is desired: MOTIF, WINDOWS, GTK
- Strong algorithmic knowledge and ability to abstract and factorise is essential
- Development experience with C++ would be an asset but is not mandatory

[…]

Job offer [Belgium]: Senior Ada 95 Developer

Mission & responsibilities:

Technical support in analysis, developing, installing, testing, tuning, upgrading and maintaining systems/applications to meet agreed business needs.

- Detailed Software design
- Implementation (design, code & test)
- Participate in test automation preparation and implementation
- Writing of documentation mainly of technical nature
- Test and debug computer program

Profile:

- English : fluent in reading, writing and speaking
- Master degree in Computer Science or Software Engineering
- Thorough experience in Ada 95 or 2005 software design and development of mission critical systems
- Experience with software design of high availability systems
- Excellent Ada 95 or 2005 programming skills (minimum 5 years of experience)
- Strong algorithmic thinking skill
- Unix or Linux: minimum experience 3 years, including knowledge of a scripting language
- SQL and database programming skills
- Ability to absorb large amounts of complex information (being able to work on and maintain a huge code base)

Knowledge of the Air Traffic Management domain is an asset.

Job offer [United Kingdom]: Software Engineer

Software Engineer Key Responsibilities:

- Development of software using Visual Studio, Ada and C and potentially other programming languages
- Verification and Test of embedded real-time software
- Provision of software consultancy and advice to clients
- Perform and deliver allocated project tasks, to quality, time and budget requirements, as directed by project manager and/or team leader
- Diligent and accurate work ethic, recognising that many of the applications we work on are safety-related

[…]

Software Engineer Engineer Qualifications and Experience:

- Numerate degree (Computer Science or Mathematics preferred)
- Good Knowledge of a variety of programming languages, including C (or C++, C#) or Ada or .NET / Visual Studio or Java
- Experience of PC based software development or embedded, real time systems
- Enthusiastic proponent of full software life-cycle best practice (requirements capture, design, documentation, testing, etc.)

Desirable Requirements for the role

- Experience of safety-related software
- Experience of embedded, real time systems
- We will also be interested in hearing from any candidates with experience of Formal Methods or Compiler Development.

Successful applicants will be required to be security cleared to at least SC level without restrictions prior to appointment.

[…]

Job offer [United Kingdom]: Software Engineer

Software Engineer - Design, Development, Embedded Software, Avionics, C, C++, Assembler, Ada, Aerospace.

Role Activities

Software engineering

Design and development of real time embedded software for Avionics products

Experience & Knowledge

Essential:

Appropriate Engineering degree.

Experience of software design life cycle within aerospace or similar controlled industry

Experience of design and implementation of real time embedded software

Software design

Software implementation (C, C++, Assembler, Ada)

Experience with target hardware (C167,C269, PowerPC, Coldfire, PIC Microchip, TMS320)

Desirable:

Software requirements analysis

Software V&V

UML (Artisan preferably), Teamwork, DOORS

Formal configuration control (e.g. PCVS)

Development within DO-178B standard

Job offer [United Kingdom]: Principal Software Engineer - Real-time embedded software

Principal Software Engineer : An Opportunity has arisen for a Senior Software Engineer […] to lead software engineering development projects. Managing teams of 7-10 Software Engineers, you will have relevant software engineering competence (see below), but just as important is the ability to plan, schedule and cost up small projects.

Principal Software Engineer Key Responsibilities:

- Lead teams in the development of existing radar and support software to add new functionality or correct defects, in accordance with the relevant software development processes
- Liaise with Systems and Hardware engineering to determine new software requirements and assess system performance
- Integrate with the target hardware and perform unit and integration testing
- Update software requirements, design and test documentation
- Participate in formal software verification and validation activities

Principal Software Engineer Experience:

- Team / Project Leadership of software development projects - Strong supervisory skills
- Real-time embedded software development experience
- C, C++, Ada 83/95
- VxWorks RTOS, Workbench/Tornado; Teamwork - RTSA/SD; OOD/UML - Rhapsody/Rational Rose
- Dimensions configuration management

It would be advantageous to have experience of full lifecycle application and of hardware / software interfacing and digital signal processing.

Job offer [United Kingdom]: Software Engineer

As Software Engineer you will be responsible for developing and maintaining system solutions that provide business process efficiencies, ensuring that all developments are aligned to strategic goals, remain within cost, and follow the agreed design and principles. The developer will be involved in the creation of the different designs throughout the lifecycle of the project and being part of the decision making process.

Main duties

- Develop PC (Windows/Linux) and embedded software across the full product life-cycle – design, implementation, test and support
- Review business requirements and/or specifications
- Design functional system areas
- Perform coding to written technical specifications
- Execute unit and system testing
- Create and maintain technical documentation
- Report progress to the Technical Director and/or Project Manager

Requirements

- Qualified to degree or 3+ years of experience in a similar role
- Strong software engineering background
- Electronics or system design knowledge
- Experience of software design, code and/or verification along with associated tools such as Ada or C/C++
- Embedded experience - Linux or Windows CE

Job offer [United States of America]: Senior Software Engineer

Job Family: Engineering

Reports to: Lead Engineer, Technical Project Manager, Programs Manager

Works with: Managers, individual employees and clients.

[…]

Job Summary

This position is responsible for a variety of intermediate to advanced engineering assignments. The candidate will demonstrate a thorough understanding of complete software lifecycle, will require minimal instruction, and will be active in informal mentoring of Software Engineers and Technicians.

The successful candidate must meet the following basic requirements:

- Experience in disciplined software development using C, C++, or Ada
- Possess strong inter-personal and communication skills

The successful candidate will possess:

- Experience with the embedded real-time software development under DO-178B
- Experience with formal verification under DO-178B
- Experience with Model Based Development tools
- Experience with software development for an embedded Linux OS
- Experience in working with a variety of embedded processors, including PowerPC
- Experience with scripting languages (Python, Perl, etc)
- Experience with development and debugging tools (oscilloscopes, logic analyzer, multi-meters, etc.)
- Broad knowledge of avionic systems
- Ability to perform analysis of requirements, design, development, verification, and documentation of moderate to complex software applications.
- Ability to breakdown software requirements into solid design and into solid test cases
- Ability to apply working knowledge in two or more programming languages
- Ability to follow good software engineering processes

The successful candidate will also possess:

- Desire to participate in process improvement

[…]

- Good working knowledge of ISO/SEI CMMI processes and procedures

[…]

Education:

- MS in Software Engineering, Computer Science or related field and 5+ years of software engineering experience in real time embedded systems, or
- BS in Software Engineering, Computer Science or related field and 7+ years of software engineering experience in real time embedded systems, or
- AS in Software Engineering, Computer Science or related field and 10+ years of software engineering experience in real time embedded systems,

Ada in Context

List of errors to be detected by an Ada compiler

From: Baptiste Fouques <bateast@bat.fr.eu.org>
Date: Thu, 18 Aug 2011 09:07:41 -0700
Subject: list of errors whose detection is required by RM
Newsgroups: comp.lang.ada

Hi all,

I am looking for a referenced list of errors whose detection is required by the Ada Standard. (either 95 or 2005).

The RM, §1.1.3 is clear on the point that compliant compiler is required to detect every error specified in the standard (RM, that is). The next chapter in the RM gives a classification of errors, mainly by the detection point, and the expected behaviour after the detection.

Then, all over the RM chapters, there is the specification of expected checks at any relevant point. By the way, those checks are not linked to the error classification.

I am looking for a single list of those checks.

Given this list, it is easy to demonstrate that no other tool than a compliant compiler is required to prove the absence of given type of errors (depending on the list). I can't find such a list. If you know where I can find this list, please give me the link…

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 18 Aug 2011 15:41:32 -0500
Subject: Re: list of errors whose detection is required by RM
Newsgroups: comp.lang.ada

[…]

So far as I know, no such list exists. It would be helpful in verifying the coverage (or lack thereof) of the ACATS test suite. […]

Probably the closest thing that exists is the Test Objective spreadsheets that I created for the Ada 2005 test suite, in order to gauge coverage. Those list all of the objectives for particular sections, which include checks (both compile-time and run-time) that need to be tested. But those only cover a small fraction of the language (about 12% of the core) - there hasn't been enough time or money to do more. You can find those objectives at http://www.ada-auth.org/acats.html -- look for the link to "Test Objective Files" at the very bottom of the page. [There are newer versions of these files that have never been posted - ask if you want the newer versions.]

Note that what you are asking for is a *very* long list. For Ada 2005, I have objectives created for 36 clauses out of 292 core clauses. This resulted in 875 total objectives. Extrapolating over the entire standard, that would result in 7100 objectives for the core alone (for Ada 2012, there are 468 clauses, giving an estimate of 11,300 objectives for the entire standard). Not all of these objectives relate to "checks" of course, but a large fraction do (most of rest relate to run-time semantics, for instance checking that the correct limb of a case statement is selected).

On elaboration rules

From: Pablo Rego <pvrego@gmail.com>
Date: Sat, 3 Sep 2011 09:57:16 -0700
Subject: Elaborate_All on child package
Newsgroups: comp.lang.ada

Hi,

I'm trying to create a child package from a package which have a class, but it needs the pragma Elaborate_All so

```
package Forum_Test is
  type Small_Class;
  type Small_Class_Acc is
      access all Small_Class;
  task type My_Task_Type
        (This_Small_Class :
              access Small_Class);
  type Small_Class is tagged limited
    record
      My_Task : My_Task_Type
                (Small_Class'Access);
    end record;

  function Construct return
               Small_Class_Acc;
end Forum_Test;

package Forum_Test.Childp is
  Small_Obj : Small_Class_Acc :=
                          Construct;
end Forum_Test.Childp;
```

and I got the messages

```
forum_test-childp.ads:2:35: info: call to
"Construct" during elaboration
forum_test-childp.ads:2:35: info: implicit
pragma Elaborate_All for "Forum_Test"
generated
forum_test-childp.ads:2:35: warning: call
to "Construct" in elaboration code
requires pragma Elaborate_All on
"Forum_Test"
```

So I included a pragma Elaborate_All in the beginning of the file, but got the message

```
forum_test-childp.ads:1:23: argument of
pragma "Elaborate_All" is not withed
unit
```

And finally I 'withed' the child package and it worked as well.

However, should it be done this way? Due to it is already "withed" to the parent package, so why have I make a new with? I mean: why is the following code incorrect:

```
pragma Elaborate_All (Forum_Test);
package Forum_Test.Childp is <....>
```

and why is the following code correct?

```
with Forum_Test;
pragma Elaborate_All (Forum_Test);
package Forum_Test.Childp is <....>
```

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Sun, 4 Sep 2011 22:21:58 -0700
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

[…]

The simple (and unsatisfying) answer: Because the RM says so.

So your problem is based on lack of understanding of elaboration. A program begins execution by elaborating all compilation units. There is no order prescribed for this except that what is mentioned in a context clause must be elaborated before.

So applied to your example, elaboration order can be:

1. Forum_Test'Spec
2. Forum_Test'Body
3. Forum_Test.Childp'Spec

With this sequence, your program would have worked. However your compiler, GNAT, chose another sequence:

1. Forum_Test'Spec
2. Forum_Test.Childp'Spec
3. Forum_Test'Body

When Childp calls Construct, you get an elaboration check that fails.

There are several ways to force certain elaboration orders. One is to use pragma Elaborate_Body in every spec whenever the spec defines a function - this forces the body to be elaborated directly after the spec. Sometimes, this is not possible.

Other pragmas are Preelaborate, Elaborate_All, Elaborate, Pure. See RM 10.2.1 Elaboration Control.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Mon, 05 Sep 2011 11:28:33 +0200
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

[…]

Side note: in order to write better Ada programs when using GNAT, run the compiler with -gnatl -gnatE every now and then.

Do so and have GNAT tell you about parts of your program that depend on elaboration order. (The final executable may not need the switches.)

From: Egil Høvik <egilhovik@hotmail.com>
Date: Tue, 6 Sep 2011 00:05:13 -0700
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

> However, should it be done this way? Due to it is already "withed" to the parent package, so why have I make a new with? I mean: why is the following code incorrect:

> pragma Elaborate_All (Forum_Test);

Forum_Test is unknown. It is not defined in this scope.

> package Forum_Test.Childp is <....>

Implicit with happens here, after Elaborate_All

> and why is the following code correct?

> with Forum_Test;

> pragma Elaborate_All (Forum_Test);

> package Forum_Test.Childp is <....>

Here, Forum_Test is known to the pragma, due to the explicit with.

Would you expect this to compile?

```
pragma Elaborate_All(Forum_Test);
with Forum_Test;
```

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Tue, 06 Sep 2011 08:51:43 -0400
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

[…]

Right. The visibility rules in context clauses are special.

They have to be, because the context clause are determining what's visible elsewhere. Why Jean Ichbiah chose this exact design, I don't know. I would have put the 'with' clauses inside the package.

[…]

> Implicit with happens here, after Elaborate_All

Nitpick: There is no such thing as an "implicit with" in Ada.

The semantics of parent/child units are defined in terms of the child being *inside* the parent. That is, Forum_Test is visible in Childp because Childp is inside Forum_Test, not because of an "implicit with".

Inside Childp, things declared in Forum_Test are directly visible.

If the semantics were defined in terms of "implicit with", then you would have to say "use Forum_Test;" to make those things directly visible. Also, things declared in the private part of Forum_Test are directly visible in (parts of) Childp; they wouldn't be visible at all in the "implicit with" semantics.

[…]

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Tue, 06 Sep 2011 15:46:08 +0200
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

[…]

If the 'with' clauses would be inside the package, you would have to change also the location of the generic formals for generic packages, wouldn't you? The declarations of the formals should be able to refer to entities in 'withed' packages.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Tue, 06 Sep 2011 10:23:54 -0400
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

[…]

You wouldn't have to, but you'd want to.

Note that Ada has always allowed forward references:

```
generic
  type Formal_Type is (<>);
  with procedure P (
      X : My_Generic.Formal_Type);
      -- This is legal!
package My_Generic is
end My_Generic;
```

which is a bit strange, given that Ada doesn't allow forward references in general.

Anyway, generic formal parameters are analogous to a procedure's formal parameters -- in both cases, they ought to come after the name of the thing being declared.

From: Christoph Grein
<christoph.grein@eurocopter.com>
Date: Tue, 6 Sep 2011 22:03:35 -0700
Subject: Re: Elaborate_All on child package
Newsgroups: comp.lang.ada

[…]

>> Why Jean Ichbiah chose this exact design, I don't know. I would have put the 'with' clauses inside the package.

I guess he wanted them to stand out and not be hidden somewhere inside.

[…]

> Anyway, generic formal parameters are analogous to a procedure's formal parameters -- in both cases, they ought to come after the name of the thing being declared.

Perhaps. I guess this special syntax was used because of the different kind of parameters. In Ada 83, there were no subprogram parameters, so since for generics, there are these kind of formals, Ichbiah chose special syntax. Today, with subprogram parameters existing (but still no type parameters), syntax decisions could have been different.

(I gather, with all the syntax and semantics discussions in this group, we would have at least 100 different Adas if we were to start from scratch.)

On the pragma Pure declaration for the Ada package

From: Yannick Duchêne
<yannick_duchene@yahoo.fr>
Date: Sat, 01 Oct 2011 03:50:36 +0200
Subject: pragma Pure (Ada)
Newsgroups: comp.lang.ada

In “A Brief Introduction to Ada 2012” (a great paper from John Barnes) ->

http://www2.adacore.com/wp-content/uploads/2006/03/Ada2012_Rational_Introducion.pdf

[The Ada 2012 Rationale introduction is also available in AUJ 32-3 (Sep 2011) —lmp]

On page 12, you may read

> Ada 95 introduced the package Ada thus
>
> package Ada is
>
> pragma Pure(Ada);
>
> end Ada;
>
> However, a close reading of the RM revealed that poor Ada is illegal since the pragma Pure is not in one of the allowed places for a pragma.

Does that mean that this was really illegal from strict lawyers point of view ? So GNAT was hacked ?

Notice how John Barnes was pleasantly teasing with the wording, as he wrote “poor Ada is illegal” instead of “Pure(Ada) is illegal”

From: Adam Beneschan
<adam@irvine.com>
Date: Fri, 30 Sep 2011 20:09:19 -0700
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

[…]

It's not fair to say GNAT was hacked. Everybody knew that this code was supposed to be legal. It's just it was illegal according to a literal reading of the RM rules that nobody noticed until I think I discovered it while I was trying to look over the rules carefully to answer a different question. So the RM rules were clearly worded wrong and had to be changed.

Different behaviour of tagged and non-tagged records in Ada 2005

From: Yannick Duchêne
<yannick_duchene@yahoo.fr>
Date: Sat, 01 Oct 2011 04:28:01 +0200
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

[…]

[still refering to “A Brief Introduction to Ada 2012” by John Barnes —mp]

On page 14

> However, the behaviour of components which are records is different in Ada 2005 according to whether they are tagged or not. If a component is tagged then the primitive operation is used (which might have been redefined), whereas for an untagged type, predefined equality is used even though it might have been overridden. This is a bit surprising and so has been changed in Ada 2012 so that all record types behave the same way and use the primitive operation.

Not talking for me, but potentially for others: this may possibly introduce surprising behaviors in ancient applications, isn't it ?

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 7 Oct 2011 20:08:40 -0500
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

[…]

Right, although in most cases the change will fix bugs rather than create them. [Someone at AdaCore ran a testing version of GNAT with added code to identify such cases -- all of the code that was found either was test programs created specifically to check for the Ada 83/95 behavior, or cases where the programmer expected "=" to compose (but it didn't). They found no examples of cases where code actually expected non-composition of "=".]

There also are some new legality rules, but those of course could only cause an old program to be rejected. (Which would easily be fixed by moving the offending "=" declaration to some visible place.) So these are much less concerning.

We spent a lot of effort trying to figure out a way to keep compatibility for old code, but in this case it always came back to the fact that composition of "=" is what is really intended, and any hack to support both makes little sense (no one would intentionally want "=" to not compose).

Thus we eventually decided that it was better to just make the change.

From: Adam Beneschan
<adam@irvine.com>
Date: Fri, 7 Oct 2011 18:37:53 -0700
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

>> Programmers have always moaned about the need for many explicit conversions in Ada. Accordingly, implicit conversions from anonymous access types to named access types are now permitted provided the explicit conversion is legal. The idea is that the need for an explicit conversion with access types should only arise if the conversion could fail. A curious consequence of this change is that a preference rule is needed for the equality of anonymous access types.

> “a preference rule is needed for the equality of anonymous access types” ?
>
> What does that mean ?

If you have two objects of an anonymous access type that point to the same type, in Ada 2005 you can test them for equality:

```
X : access My_Record;
Y : access My_Record;
if X = Y then …
```

This calls a function "=" that is defined in the language and takes "universal access types" as parameters.

If you define a named access type:

```
type My_Record_Acc is
  access My_Record;
```

this defines an implicit "=" operator:

```
function "=" (
    Left, Right : My_Record_Acc)
  return Boolean …
```

[and you can override it with your own operator if you really want to]. In Ada 2005, this didn't pose a problem. But in Ada 2012, if an anonymous access type (access My_Record) could be implicitly converted to a named access type (My_Record_Acc), then in this:

```
if X = Y then …
```

the compiler couldn't tell whether "=" means the function on "universal access", or the function on My_Record_Acc (since X and Y can now be implicitly converted to My_Record_Acc). This is ambiguous, and ambiguous function calls are normally an error, but making it an error would make some legal Ada 2005 programs illegal in Ada 2012. So a special rule had to be added to make the language *prefer* the "universal access" equality function over any other function. Hope this helps.

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Sun, 09 Oct 2011 00:54:28 +0200
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

[…]

Yes Adam, that helps, but it raises another question: the choice of the anonymous access type operator is surprising to me. Why was the choice of the more specific operator rejected ? I guess this may be because the anonymous access type may not always be a valid parameter for the more specific redefined operator of the named access type (ex. different storage pool), but the choice of the least specific operator, is counterintuitive, and unintuitive things may lead to bad surprised and unexpected behavior (an error at compile time is always preferable to an unexpected behavior).

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sun, 09 Oct 2011 08:34:51 +0200
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

[…]

There can be several different such "more specific" operators, when there are possible implicit conversions of the anonymous access type to several named access type. Which "more specific" operator should be chosen?

[…]

The choice of the least specific operator minimizes the number of implicit type conversions, which in my view is good. But I would, perhaps, like to have a compiler switch to warn me when this preference rule is applied, especially if the non-preferred implicit conversions could reach an operator that is not the predefined operator for its type.

From: Adam Beneschan <adam@irvine.com>
Date: Mon, 10 Oct 2011 08:06:26 -0700
Subject: Re: pragma Pure (Ada)
Newsgroups: comp.lang.ada

[…]

> I guess this may be because the anonymous access type may not always be a valid parameter for the more specific redefined operator of the named access type (ex. different storage pool), but the choice of the least specific operator, is counterintuitive,

Actually, I think the opposite is true. Say you have a package that defines a record type Rec1 which defines an element of a linked list, and it has a Link field:

```
Link : access Rec1;
```

Now, in another package P2, you have some reason for defining your own access type that accesses Rec1. Maybe it's declared as a private type:

```
type Employee is private;
```

but you decide to implement it as an access-to-Rec1, in the private part:

```
type Employee is access all Rec1;
```

If we followed the rule you say is "intuitive", then any time in the body of P2, you wrote

```
if A.Link = B.Link then ...
```

the Link fields would suddenly be treated as having type Employee (because the "=" operator on Employee would be used), even though Link wasn't declared as having type Employee and the Employee type wasn't even visible at the point the Link was declared. This, to me, is counter-intuitive. So I think you've gotten it backwards.

The example is somewhat contrived; it's hard for me to think of a good real-life example. In practice, "=" won't be redefined that often, which means it really doesn't matter which one the compiler picks. You're going to be comparing two addresses for equality, no matter which one is picked. But there have to be some rules to allow the compiler to pick one. And, as Niklas pointed out, if there are multiple named access types visible that access the same designated type, using the "more specific" operator would still produce ambiguous errors, which is not desired.

Calling Ada procedures from C

From: awdorrin <awdorrin@gmail.com>
Date: Tue, 1 Nov 2011 11:46:07 -0700
Subject: Calling Ada from C (linux/gnat 4.3.2)
Newsgroups: comp.lang.ada

I am trying to port a program originally written on VxWork to Linux.

The main program is written in C and spawns new threads and in the new threads calls are made to Ada procedures.

In the original program, written with GreenHills AdaMulti, it appears that there was a call being made that I am assuming was initializing the Ada Runtime's task stack/control block (rts_init_task()) - however this is just a guess.

In the version of the program I am migrating, I was seeing things that made me believe that the Ada thread's did not have their stacks setup properly, so I added the -fstack-check flag to the build.

Now, the C program creates the new thread, which calls the Ada procedure - and the moment the thread has an opportunity to run - the application exits with 'raised STORAGE_ERROR : stack overflow detected'

I cannot get GDB to provide me a backtrace, since the program exits. If I try to break on the Ada procedure name, I get a break, but trying to 'step' or 'next' keeps me in the main thread until a 'sleep' call lets the other thread run - at which point it raises the exception.

I am assuming that the Ada runtime is initializing and the STORAGE_ERROR is raised before it gets to execute any of the code in my Ada procedure.

I have been trying to search to find what the proper way would be to call an Ada procedure from a C pthread, but have had zero luck. I also have been unable to determine what the equivalent of the 'rts_init_task' would be with GNAT.

Any have any ideas?

From: Jeffrey Carter <jrcarter@acm.org>
Date: Tue, 01 Nov 2011 13:44:25 -0700
Subject: Re: Calling Ada from C (linux/gnat 4.3.2)
Newsgroups: comp.lang.ada

[…]

Does your C main program call adainit and adafinal [ARM B.1 (39)]?

From: Simon Wright <simon@pushface.org>
Date: Tue, 01 Nov 2011 23:38:04 +0000
Subject: Re: Calling Ada from C (linux/gnat 4.3.2)
Newsgroups: comp.lang.ada

[…]

The info on-line is sparse; if you google 'gnat foreign threads' you'll find [1], which tells you to look at the documentation of GNAT.Threads (g-thread.ads). What it wants you to do is to read the comments in that file.

In the past I've had success calling Register_Thread from the called Ada procedure before it does anything at all, but that's not what the comments seem to be saying. For example, I might have said

```
procedure P;
pragma export (C, P, "my_ada_proc");
procedure P is
  Id : constant System.Address :=
      GNAT.Threads.Register_Thread;
  …
begin
  …
```

Jeffrey is correct that you need to call adainit() (and you should call adafinal() before program exit or unloading a shared library).

[1] http://gcc.gnu.org/onlinedocs/gcc-4.6.0/gnat_rm/GNAT_002eThreads-_0028g_002dthread_002eads_0029.html

From: anon@att.net
Date: Wed, 2 Nov 2011 04:17:42 +0000
Subject: Re: Calling Ada from C (linux/gnat 4.3.2)
Newsgroups: comp.lang.ada

Within C use normal external calling

```
extern <name> <arg list>
```

And in Ada use a specification file to declare routine.

```
procedure <name> ( arg list ) ;
  pragma Export ( C, <name> ) ;
```

or you could use

```
pragma Export
  ( C, <name>, "<linked name>" ) ;
```

In Ada the routine may or may not be in a package.

[…]

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 02 Nov 2011 12:05:38 -0400
Subject: Re: Calling Ada from C (linux/gnat 4.3.2)
Newsgroups: comp.lang.ada

[…]

I suggest you rewrite the C main in Ada, and avoid all of these problems.

Having a C main is probably a good idea in VxWorks, but totally unnecessary in Linux.

Getting "C threads" and Ada tasks to intermix properly is not trivial.

Ada Variant Record and C++ union

From: Guarita
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

I am designing a communication middleware for use in an application which has a module in Ada and many modules in C++ which communicates passing parameters (scalar values) and structures. The application runs in MS Windows XP and Windows 7, the C++ part is being developed in MSVC++ 2008, the Ada part is being developed using GPS/GNAT. Ada version is 1995 but we're in the middle of a compiler migration (newer version of GPS/GNAT) with the possibility of using newer Ada spec.

The middleware is being written in C++ and I would like to use a union type containing all types that are passed between modules so I won't need to specify one put/get function for each type that is used on the system.

The question is, are C++ unions binary compatible to Ada variant records?

In other words, if I pass a C++ union to Ada code will it be able to read it as a Variant record? (and vice-versa)

I think that for this to be possible some adjustments will be necessary… (Eg.: C++ unions do not contain a member which describes its content while Ada variant records do)

From: Marc A. Criley
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

Possibly.

Ada 2005 provides the Unchecked_Union pragma which allows a program to "[specify] an interface correspondence between a given discriminated type and some C union. The pragma specifies that the associated type shall be given a representation that leaves no space for its discriminant(s)."

From my reading of the RM section, one declares an Ada type with the discriminant(s) needed to define a variant record, but no storage space is allocated for the discriminant(s). I take it this means on the Ada side that the discriminant cannot subsequently be referenced. (There are other restrictions as well, like all the fields must be C-compatible, see RM B.3.3 for more info.)

I've never used this pragma, and I can't help but think that it will require some experimentation to get it to (hopefully) work with your system. Good luck!

From: Simon Wright
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

Some of the experimentation will be to find out whether you need to tell GNAT to lay the records out the way MSVC++ expects, and if so how. GNAT understands GCC's conventions, of course; so long as you restrict yourself to C unions and straightforward types it shouldn't be too bad.

From: MSalters
Date: Fri, 4 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

GCC on Windows conforms to the platform C ABI, so GNAT on Windows should be ABI compatible with MSVC++.

From: NWS
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

[…]

Ada is compatible with C/C++ Unions. See here for how to do it [http://www.ghs.com/download/whitepapers/ada_c++.pdf —mp]

In particular Page 5 shows how to do it with Unions & Tags. It should be the same for using Discriminant records.

(Caveat: it is probably not the compiler you are using, but I would be very surprised if yours didn't behave the same way!)

From: Marc A. Criley
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

I think you're a little optimistic about the expectation that one can expect different Ada/C/C++ compilers to handle this with no great concern, but it is a helpful guide to the problem.

From: Marc A. Criley
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

> My Optimism is based on the fact that Ada compilers should be validated, so they should be able to do the same things (even if not equally well!)

It's not the Ada side I'm concerned about, it's the field layout and linking minutia amongst different compilers that colors me skeptical (of optimism :-).

I'm no stranger to bindings, even sharing data structures between GNAT and GCC requires care.

From: T.E.D.
Date: Thu, 3 Nov 2011
Subject: Can an Ada Variant Record be binary compatible to a C++ union?
URL: http://stackoverflow.com/questions/7994386/can-an-ada-variant-record-be-binary-compatible-to-a-c-union

As MSalters mentioned, it won't work unless the C union for some reason contains a field designating the variant. As this isn't required in C, it won't often work. However, since you control the implementation of that C type, you can make it work. Just be sure to have a field right before the union that designates which union is being used.

To make it fully binary-compatible with your C union-bearing struct, you will probably need to go with a simple Ada record type, along with a record representation clause to make sure the fields are laid out in the same places your C compiler happens to put them. And yes, that does leave you vulnerable to C compiler changes causing layout changes. You can try to protect against that with bitfields in your C code, but they aren't powerful enough to really lay things out the way Ada record rep clauses can. That's one of the reasons we prefer to use Ada for low-level work.

I should mention that, when last I checked, the Windows version of GNAT was not linker-compatible with VisualStudio binaries. The only way I know of to get those two compilers to work together is to put the entire interface in a DLL.

Otherwise, you will probably either need to use GCC to build your C++ system, or use some other Ada compiler, like ObjectAda.

Support for green threads / FSU threads in GNAT

From: Simon Wright <simon@pushface.org>
Date: Sat, 10 Sep 2011 16:19:00 +0100
Subject: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

It used to be that you could configure GNAT to use a tasking RTS that didn't use threads; tasks were scheduled entirely within the RTS. I think these may be called 'green threads' [1]. Of course, this meant that blocking on I/O would block the whole program, but for some purposes it might be ideal.

I don't see any trace of this in FSF GCC; has it gone for good?

[1] http://en.wikipedia.org/wiki/Green_threads

From: John B. Matthews
Date: Sun, 11 Sep 2011 05:22:32 -0400
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

> No, it used the FSU threads

FSU threads were used to implement Ada tasking in Tenon's MachTen and CodeBuilder products for Mac OS 9.

http://www.tenon.com/products/machten/

From: anon@att.net
Date: Sun, 11 Sep 2011 09:49:51 +0000
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

Unless 2011 has changed the default run-time library, When GNAT is installed the RTL is rts-native. Native uses the thread package of the underlying OS.

"FSU threads library", "pthreads library" (Solaris only), "Zero-Cost Exceptions" ("ZCX"), and "setjmp/longjmp" ("SJLJ"), are optional if compiled, by using the "--RTS=" command line option.

[…]

From: Simon Wright <simon@pushface.org>
Date: Sun, 11 Sep 2011 11:36:11 +0100
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

> No, it used the FSU threads

Thanks, all, for that.

>From the Changelog for FSF GCC:

2005-02-09

* gnat_ugn.texi: Remove all mentions of FSU threads, which are no longer supported.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 12 Sep 2011 00:19:00 -0700
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

The FSU threads library was a kind of green threads. The GNAT User's Guide of GCC 3.4 used to say: "The FSU threads package operates with all Ada tasks appearing to the system to be a single thread."

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 12 Sep 2011 02:26:23 -0700
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

> Plus, between 2004 .. 2010 for Linux (binary package) from Libre.Adacore.com only contains native, siji, and maRTE libraries only. No default FSU or ZCX RTL. If still available you will have to download source and compile your own version to get FSU threads.

Presumably, by "siji" you meant "sjlj" which stands for "setjump/longjump exception handling". GCC presently comes with two run-time systems:

- zero-cost exception handling aka ZCX aka "native", which is the default on most architectures. The phrase "zero-cost" really means "zero distributed cost" which means you do not pay a performance penalty unless and until you raise an exception. Raising an exception is however costly.
- sjlj aka setjump/longjump, which is the alternative, in which every frame that might possibly raise an exception calls setjmp(3), thus incurring distributed cost, and raising an exception calls longjmp(3), which is cheap compared to the raising of an exception with ZCX. sjlj is the only supported run-time system on a few, non-mainstream architectures.

Both run-time systems now use native threads and both support Annex D (real-time systems) insofar as the underlying kernel does. The old FSU threads were intended to provide better support for Annex D but that was rendered unnecessary by the advances in Linux, Solaris and other kernels.

It is true that, a few years ago, Glade used to require the sjlj run-time system to implement exception handling across partitions in in distributed programs (Annex E). I know because I packaged it for Debian. However, PolyORB has not required sjlj since it introduced support for Annex E in version 2.2.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 12 Sep 2011 02:49:51 -0700
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

I forgot to add that, as Pascal said, ZCX and SJLJ have nothing to do with tasking, so bringing that up in a discussion about FSU vs. native threads as though it had an influence is indeed misleading. My previous post was to clarify the present state of GNAT and the underlying concepts.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Mon, 12 Sep 2011 09:01:17 -0400
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

> - sjlj aka setjump/longjump, which is the alternative, in which every frame that might possibly *raise* an exception calls setjmp(3) […]

You meant "handle".

> incurring distributed cost, and raising an exception calls longjmp(3), which is cheap compared to the raising of an exception with ZCX. sjlj is the only supported run-time system on a few, non-mainstream architectures.

Right. ZCX is the "right" way to implement exception handling, because raising exceptions is rare, whereas exception handlers (and finalizable objects) are not-so-rare, so you want to pay the cost on the raise.

On Erlang-like data passing in Ada

From: Simon Wright <simon@pushface.org>
Date: Tue, 13 Sep 2011 09:30:50 +0100
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

> It used to be that you could configure GNAT to use a tasking RTS that didn't use threads; tasks were scheduled entirely within the RTS. I think these may be called 'green threads'[1]. Of course, this meant that blocking on I/O would block the whole program, but for some purposes it might be ideal.
>
> […]
>
> [1] http://en.wikipedia.org/wiki/Green_threads

My prompt for asking this was a colleague who was used to Erlang and was complaining that GNAT's use of OS threads meant he would have to change his design mindset to not use thousands of tasks (Erlang processes).

I believe Scala is similar.

I believe that Erlang allows you to classify some threads as maybe-io-bound.

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Tue, 13 Sep 2011 06:57:05 -0400
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

If one is in control of the run time library and if the underlying OS supports asynchronous I/O, then I believe it is possible to write a user mode thread library that works "nicely" even in the face of I/O operations. When calling an I/O operation that might block the library uses asynchronous I/O so that the single kernel thread can be scheduled onto a different user thread while the I/O completes.

> I believe Scala is similar.

In Scala you can create "thread based" actors that consume a single thread each or "event based" actors that can all share a single thread. I haven't experimented with this but my guess is that if you do a blocking operation while handling an event you may well tie up all event based actors. I imagine the thread based actors would continue to work, however.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Tue, 13 Sep 2011 11:39:16 +0200
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

Just an idea: using mostly functions together with protected channel objects should allow to use Ada in a way one might be used to when passing Erlang data around.

You write to some entry/procedure of a PO (pass data down the channel) and read from some entry/function of the PO (read from a channel). Add barriers as needed.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 13 Sep 2011 14:18:38 +0200
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

Unfortunately this does not work this way for many reasons (I have evaluated this design for our middleware and quickly dropped the idea).

Protected objects are not tagged, you need inheritance to provide typed channels. You meed multiple dispatch to handle channel-type + value-type hierarchies. You need entries returning indefinite values. You need MI to have handles to the channels/devices implementing the interface of a protected object.

> You write to some entry/procedure of a PO (pass data down the channel) and read from some entry/function of the PO (read from a channel). Add barriers as needed.

Of course this can only be the transport layer. In our design at the application layer the channels are multiplexed into typed named channels, so that you can exchange data as if you had one channel per each variable and in full duplex mode of course.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de
Date: Tue, 13 Sep 2011 18:35:09 +0200
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

>> Protected objects are not tagged, …

> Protected types are tagged if they are derived from synchronized interfaces.

Effectively not, for design purpose they should be usable as parent types to derive from, protected operations has to be primitive.

An inability to push implementations down the hierarchy (which is also the case for Ada's MI) poses a huge problem for the designer and for the end users.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 13 Sep 2011 22:35:03 +0200
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

> Is there an AI on "limited holders"?

Holder is useless without delegation, interface inheritance, MI, otherwise it quickly becomes an endless swamp of generic instantiations. Note also classical MD case: channel-type x value-type (<=> handle-type).

BTW, protected objects are unsuitable for distributed interfaces anyway.

You need a background task to prevent blocking upon I/O. The usual technique of re-queueing does not help here. The interfaces must be tasks, rather than objects.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Tue, 13 Sep 2011 23:22:52 +0200
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

To be fair, writing Erlang is perhaps associated with a more "pragmatic" attitude towards static typing, which means you will be tracing and debugging anyway when looking for things frequently detected by Ada compilers before running the program.

With this in mind, an owner of a protected channel object (Ada) can "receive" (access to) Any'Class objects and trigger dispatching calls, primitive subprograms of the received objects, where Erlang would perform a case distinction.

Can "agents" then simply share a physical task by being selected for acting, perhaps triggered by messages sent (rendezvous if ready), or by some simple scheduler task selecting them in a round robin fashion, or in a way that resembles reacting to HTTP requests in AWS?

Yes, when some agent needs to both deliver a message and be sure the message is sent, then it may wait in the channel's queue forever until delivery is signaled. If the system allows messages to be dropped, then barriers can reflect this permission. How would tasks be more helpful?

Thus, reducing the Channel PO to a very basic thing, and, unfortunately, exhibiting all the pointers inherent in most functional programming languages,

```
package Sys is
   type Any is abstract tagged
         limited null record;
   -- ... parent/interface of every
      message type
   type Box is tagged private;
   -- holds a value of type `Any'Class`;
      see `Ref` and `Deref`

   -- functions for wrapping and
      unwrapping:
   function Ref (Item : Any'Class)
     return Box;
   function Deref (This : Box)
     return access constant Any'Class;

private
   type Poly_Cell is
     access constant Any'Class;
   type Box is tagged
      record
        Storage :  Poly_Cell;
      end record;
end Sys;

package Sys.Messages is
   type Vector is array
      (Natural range <>) of Box;
   -- a channel object's container of
      boxes; message box
   protected type Channel
      (Capacity : Natural) is
     entry Send (Object : in Box);
     -- ! operation
     entry Receive (Object : out Box);
     -- pattern matching will correspond
        to dispatching based on what is
        in `Object`
   private
     Queue : Vector (1 .. Capacity);
     Front : Natural := 0;
     Rear : Natural := 0;
   end Channel;
end Sys.Messages;

[…]

package body Sys.Messages is
   protected body Channel is
     entry Send (Object : Box)
       when Rear < Capacity is
     begin
       Rear := Rear + 1;
       Queue (Rear) := Object;
     end Send;

     entry Receive (Object : out Box)
       when Front < Rear is
     begin
       Front := Front + 1;
       Object := Queue (Front);
       if Front = Rear then
         Front := 0; Rear := 0;
       end if;
     end Receive;
   end Channel;

   function Ref (Item : Any'Class)
     return Box is
   begin
     return Box'(Storage =>
          Item'Unchecked_Access);
   end Ref;

   function Deref (This : Box) return
      access constant Any'Class is
   begin
     return This.Storage;
   end Deref;

end Sys.Messages;
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 14 Sep 2011 09:58:54 +0200
Subject: Re: Does GNAT support a thread-free RTS?
Newsgroups: comp.lang.ada

[…]

> With this in mind, an owner of a protected channel object (Ada) can "receive" (access to) Any'Class objects and trigger dispatching calls, primitive subprograms of the received objects, where Erlang would perform a case distinction.

However implemented, this call can only initiate I/O.

> Can "agents" then simply share a physical task by being selected for acting, perhaps triggered by messages sent (rendezvous if ready), or by some simple scheduler task selecting them in a round robin fashion, or in a way that resembles reacting to HTTP requests in AWS?

Yes, they do. In my design, there is a queue to the I/O task (of a "device"), to which the operation is queued. Note how this resembles the behavior of a entry task, being unable to become one.

> Yes, when some agent needs to both deliver a message and be sure the message is sent, then it may wait in the channel's queue forever until delivery is signaled. If the system allows messages to be dropped, then barriers can reflect this permission. How would tasks be more helpful?

Yes there is much stuff coming with. You want to be able to wait for a completion. You want to be able to cancel the operation pending. You want all temporary objects freed. You want references between objects and their marshaled parts etc.

To much disappointment, though this appears very similar to how Ada tasks works, it is impossible to promote as a task or reuse task mechanics (queues for example).

>[…]

> package Sys is

>

> type Any is abstract tagged limited null record;

> -- ... parent/interface of every message type

Nope, you want to send/receive types, e.g. Integer, not messages. For "messages" there is already Stream_Array, uninteresting.

> type Box is tagged private;

> -- holds a value of type `Any'Class`; see `Ref` and `Deref`

>

> -- functions for wrapping and unwrapping:

> function Ref (Item : Any'Class) return Box;

> function Deref (This : Box) return access constant Any'Class;

You don't need that, it has values semantics, because things are marshaled. I have reference-counted handles to the devices and variables ("registers"), but that is mostly because of the language. From the application point of view, the scopes of devices and variables are static. Counters are used because there is no way to express their dependencies in a static manner.

> protected type Channel (Capacity : Natural) is

>

> entry Send (Object : in Box);

> -- ! operation

That is a "device" interface (the transport layer [*]). Actually Send is a primitive operation of a register, rather than the device (the application layer). When you do Send, you queue to the device an I/O request with a reference to the register. You also return back the request object to the caller, in order to be able to wait for it or cancel it. You may have several requests pending on a variable, if the underlying protocol support this.

This is how it looks like in my design:

```
Device : EtherCAT_Device_Handle :=
            Create (Name, Adapter);
Output : Middleware.Output.
     Integers_16.Register_Handle :=
                              ...;
...
```

```
Output.Send (123);  -- Asynchronous
                       send
Output.Wait;  -- Wait for completion
Output.Write (456);  -- Synchronous
                       send and wait
```

And do not forget the opposite direction:

```
Input.Request;  -- Asynchronous
                   request
X := Input.Wait;  -- Wait for completion
Y := Input.Read;  -- Synchronous
                     request and wait
```

* It cannot be a protected object either, because you want reusable implementations of the devices. E.g. I have abstract half-duplex device driver with abstract primitive operations. Protected objects are not composable upon inheritance. The object you had in mind is still there, but in the form of a specialized lock object hidden deep inside.

On fixed point and floating point types

From: Rasika Srinivasan <rasikasrinivasan@gmail.com>
Date: Thu, 29 Sep 2011 03:25:31 -0700
Subject: fixed point vs floating point
Newsgroups: comp.lang.ada

[…]

I am investigating the applicability of fixed point to a numerical problem. I would like to develop the algorithm as a generic and test with different floating and fixed point types to decide which one to go with.

Questions:

- Ada.Numerics family is pretty much floating point only - is this correct?
- Can we design a generic (function or procedure) that can accept either fixed point or floating point data types at the same time excluding other types?

[…]

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Thu, 29 Sep 2011 03:49:48 -0700
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

Unfortunately, fixed and floating point are separate categories of real types, so there is no generic formal that can serve both.

You have to make the type private and supply all numeric operations like this:

```
generic
  type Real is private;
  with function "+" (Left, right: Real)
return Real;
  ...
package Numerics is
```

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 30 Sep 2011 06:17:28 -0400
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

[…]

> I am investigating the applicability of fixed point to a numerical problem. […]

What sort of criteria are you using to make the decision?

If it's just speed, then the answer will depend more on the hardware and the level of compiler optimization than on this choice.

The major algorithmic difference between fixed and floating is the handling of small differences; floating point allows arbitrarily small differences (down to the exponent limit, of course), while fixed point has a fixed small difference.

So the choice should be determined by the application, not by experiment.

The only place I have found fixed point to be useful is for time; everything else ends up needing to be scaled, so it might as well be floating point from the beginning.

The other thing that can determine the choice is the hardware; if you have no floating point hardware, you will most likely need fixed point. But even then, it depends on your speed requirement. You can do floating point in software; it's just slower than fixed point on the same hardware.

[…]

From: Tom Moran <tmoran@acm.org>
Date: Fri, 30 Sep 2011 16:25:22 +0000
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

> The only place I have found fixed point to be useful is for time; everything else ends up needing to be scaled, so it might as well be floating point from the beginning.

Also for matching instrument or control values, formatting output, saving memory, interfacing to C stuff, or future proofing.

In embedded devices measurements usually come in implicitly scaled integers, not float, as do output control values.

If Degrees is fixed point, Degrees'image is much more readable than if it's in floating point.

Usually real world physical values don't need 32 or more bits of float for either their range or precision. If memory size (or IO time) is an issue, they can be stored in much smaller fixed point format.

Very often values passed to C et al. are scaled, e.g. durations are milliseconds or seconds or hundredths of seconds, represented as integers, angles are tenths of a degree integers, and so forth. Trying to do calculations remembering the proper scaling is error-prone, but the compiler will do it correctly if you use fixed point.

Intensities (e.g. color, sound) are always fractions, but they are usually represented as if they were integers ranging from 0..15, or 0..255, or 0..65535. Code like

```
Is_Bright := (Color > 128);
```

is much more tedious and error-prone to change then

```
Is_Bright := (Color > 0.5);
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 30 Sep 2011 18:52:23 +0200
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

[…]

Right, but arithmetic of color-models intensities is not linear, so although a fixed point type would be far more convenient for color stimuli, it still would require redefinition of the operations.

An addition to your list: screen units (horizontal, vertical coordinates).

Traditionally rendering frameworks are using floating point for them, but I think that fixed point could be more suitable with regard of anti-aliasing issues etc.

To the OP: Integer type is a special case of decimal fixed point. So I don't understand your desire to single out signed integer types then. However, for the modular ones, it would indeed make sense.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 01 Oct 2011 07:09:41 -0400
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

[…]

> In embedded devices measurements usually come in implicitly scaled integers, not float, as do output control values.

Well, yes. I do declare fixed point types that match hardware values.

But they immediately get turned into float (or time fixed point); they are not used in computations.

> If Degrees is fixed point, Degrees'image is much more readable than if it's in floating point.

Put (item, fore, aft, exp) gives the same control.

> Usually real world physical values don't need 32 or more bits of float for either their range or precision. If memory size (or IO time) is an issue, they can be stored in much smaller fixed point format.

Yes; these are reasonable criteria.

[…]

From: Rasika Srinivasan <rasikasrinivasan@gmail.com>
Date: Sat, 1 Oct 2011 06:37:25 -0700
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

In embedded platforms, it is not often the case we have a floating point processor (or it may come at a price which we cannot afford!) and they have to be emulated. Fixed point arithmetic may do the job in certain class of problems. In my class of problems, I am not sure the fixed point arithmetic will be sufficient. The experiments are to understand how the fixed point solutions may diverge from the floating point solutions.

But the basic answer appears to be that the Ada generics makes it a bit harder to do this - but unfortunately make Ada.Numerics.* also not a viable option for fixed point data types.

[…]

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 02 Oct 2011 10:19:06 -0400
Subject: Re: fixed point vs floating point
Newsgroups: comp.lang.ada

[…]

It is not likely that experiment will answer that question, unless the only issue is speed.

If you need to worry about data range and precision, analysis of the inputs and algorithms is necessary. It might be possible to develop a truly representative set of data for testing this, but that requires the same analysis!

Speed has to be measured, on a representative set of data; it's somewhat easier to develop a data set that covers all speed issues.

On Ada 2012 iterators [1]

From: comp.lang.php <kst@ecs.soton.ac.uk>
Date: Tue, 8 Nov 2011 02:05:33 -0800
Subject: Iterators in Ada2012
Newsgroups: comp.lang.ada

I have an instantiation of the package Ada.Containers.Ordered_Maps (Integer, Float). Is there a neat way of iterating over the values similar to the construction:

```
for x of a_Set loop … end loop;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 8 Nov 2011 15:25:19 -0600
Subject: Re: Iterators in Ada2012
Newsgroups: comp.lang.ada

[…]

Exactly that should work (but only in Ada 2012, and only in a reasonably complete implementation). Don't know if GNAT has all of the needed features (my understanding was that a few were still missing, but that's a few months old now).

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Tue, 08 Nov 2011 14:13:03 +0100
Subject: Re: Iterators in Ada2012
Newsgroups: comp.lang.ada

[…]

Fear of nesting a procedure? :-)

I had thought that the new syntax of AI 139 would be for all containers.

Strangely, the fashion seems to be moving elsewhere:

list comprehensions, Map + Reduce (US Patent #7,650,331), Scala, … all functional, hence little sympathy for for-loops.

Sequence comprehensions are now recommended when writing Python, so I had thought there is all the more to say in favor of a simple automatic

```
Container.Iterate (Process =>
   Meaningful_Name'Access).
```

But OTOH, Guido van Rossum suggested to simply write a for-loop in place of a traditional reduce procedure. With Ada, the two can be combined…

From: R. Tyler Croy <tyler@linux.com>
Date: Wed, 9 Nov 2011 04:12:29 +0000
Subject: Re: Iterators in Ada2012
Newsgroups: comp.lang.ada

[…]

The reasons things like list comprehensions and generator expressions are recommended in Python is because they're faster in the CPython implementation of the language since they get to native C "faster" (with less interpretation and bytecode).

Additionally, generator expressions: `(x for x in iterable)` are lazily evaluated so you don't have to actually have your working set in memory all at once.

An Ada "built-in" equivalent of a generator expression I actually don't know of, I suppose you could hide that behind your Container.Iterate procedure?

From: Simon Wright <simon@pushface.org>
Date: Wed, 09 Nov 2011 09:02:38 +0000
Subject: Re: Iterators in Ada2012
Newsgroups: comp.lang.ada

[…]

I had no idea that list comprehensions were faster (I have no idea whether Mac OS X's native Python is a CPython implementation!), I used them because they are so expressive!

From: Georg Bauhaus <rm-host.bauhaus@maps.futureapps.de>
Date: Wed, 09 Nov 2011 11:55:52 +0100
Subject: Re: Iterators in Ada2012
Newsgroups: comp.lang.ada

[…]

> Additionally, generator expressiones: `(x for x in iterable)` are lazily evaluated so you don't have to actually have your working set in memory all at once.

Maybe it is worth mentioning---though in this case savings come from the other end where collections exist---that Iterate may not have to inspect every element. Exceptions such as StopIteration in the following example, can make the process stop early, as needed:

```
with Ada.Containers.Ordered_Maps;
with Ada.Text_IO;

procedure Iter is
   type Location is digits 8;
   type Coords is record
      Sun : Location;
      Snow : Location;
   end record;
   type Place is (Nowhere, Springfield,
        Shangri_La, Meryton, Zamunda);
   package Maps is new
        Ada.Containers.Ordered_Maps
             (Key_Type => Place,
              Element_Type => Coords);
   StopIteration : exception;
   Whereabouts : Maps.Map;
   First_Above_Equator : Place;

   procedure Is_Above (Key : Place;
                  Element : Coords) is
      -- When place `Key` is above the
         equator, stores it in
         `First_Above_Equator` and
         raises `StopIteration`
   begin
      if Element.Snow > 0.0 then
         First_Above_Equator := Key;
         raise StopIteration;
      end if;
   end Is_Above;

   procedure Stops_Early
         (Position : Maps.Cursor) is
      -- inspects key/element via
         `Is_Above`
   begin
      Maps.Query_Element
         (Position, Is_Above'Access);
   end Stops_Early;

begin
   First_Above_Equator := Nowhere;
   Whereabouts.Insert (Meryton,
              Coords'(Sun => 0.0,
                      Snow => 51.0));
   -- ...
   Whereabouts.Iterate
        (Stops_Early'Access);
```

```
exception
   when StopIteration =>
     Ada.Text_IO.Put_Line
           (Place'Image
              (First_Above_Equator));
end Iter;
```

> An Ada "built-in" equivalent of a generator expression I actually don't know of,

Me neither; though Ada.Numerics.*Random are examples of how to model producing elements on demand. I imagine one can do similar things with task objects in order to emulate a Python style yield. The result, then, is a generating mechanism that does not require a working set in memory. I guess one could make the mechanism more generically useful.

```
task body Generator is
   Current : Value_Type;
begin
   loop
     Current := Produce_Next;
     accept Next (Result : out
                        Value_Type) do
        Result := Current;
     end Next;
   end loop;
end Generator;
```

And wrap it in an Ada.Container like package.

On Ada 2012 iterators [2]

From: David Sauvage <david.sauvage@adalabs.com>
Date: Thu, 29 Sep 2011 05:31:06 -0700
Subject: Ada 2012 Iterators limitations & proposition
Newsgroups: comp.lang.ada

Ada 2012 Iterators are very convenient by avoiding the user creating a procedure that will be used via quote access to iterate.

The problem (may be a feature for some) is that, by hiding the indexing it only gives the user an access to the Element of the container/array, but no access to the corresponding key/index.

I think it would be interesting to be able to have a read-only access the key/index. This is the only reason why I can't use these new features every time I want it.

The only challenge is to declare the key and the element variable instead of only the element, here is what I would propose to avoid any language impact ;

Allow the user to specify 2 variables, separated by a comma.

If only one variable is present, it will be affected the element, If two variables are present, the first is the key/index, the second is the element.

The user could specify null instead of a variable name, if he only wants the key/index for example.

Specifying null for key/index and element would not be legal.

```
-- access only index/key
  for Key, null of Data.Processors loop
    Process (Key);
  end loop;

-- access both index/key & element
  for Key, Element of Data.Processors
       loop
    Process (Key, Element);
  end loop;

-- access to element only (2)
  for null, Element of Data.Processors
       loop
    Process (Element);
  end loop;

-- access to element only (2), for
  compatibility
  for Element of Data.Processors loop
    Process (Element);
  end loop;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 29 Sep 2011 20:44:18 -0500
Subject: Re: Ada 2012 Iterators limitations & proposition
Newsgroups: comp.lang.ada

[…]

> The problem (may be a feature for some) is that, by hiding the indexing it only gives the user an access to the Element of the container/array, but no access to the corresponding key/index.

This is not true. You have access to either the index or the element (but not both).

I did not think (and still do not think) that the element version is necessary or all that useful. (It's "cool", though.) Most of the time you would want to use the index (cursor) form.

When you write something like:

```
for Index in My_Container.Iterator
   loop
```

you have easy access to the elements using the indexing forms, so there is little need for direct access to the element.

Specifically, you could write a loop to bump a count in every element of a container as follows:

```
for Index in My_Container.Iterator
     loop
   My_Container(Index).Count :=
        My_Container(Index).Count + 1;
end loop;
```

essentially the same thing you would have written if My_Container would have been an array.

In this case, you don't need the index for anything else, so you could use the element form as well:

```
for Element of My_Container loop
   Element.Count :=
        Element.Count + 1;
end loop;
```

These have the same semantics and will generate the same code.

> I think it would be interesting to be able to have a read-only access the key/index. […]

You have it, just use the index form, not the element form. (I think that using the index form requires calling the Iterator function for the appropriate container; the element form does this automatically -- but that allows the index form to support alternative iterators, like the form with the optional "Start" parameter that is available for the lists, and the Iterate_Subtree that's available for the trees.)

[…]

> The user could specify null instead of a variable name, if he only wants the key/index for example. […]

This form is not necessary, as it already exists. (It just uses the familiar "in" syntax, not "of".)

> -- access both index/key & element

> for Key, Element of Data.Processors loop

> Process (Key, Element);

> end loop;

[…] I think this would be very confusing to the reader.

Data(Key) and Element would both represent the same element, and there would no reason to choose one over the other. Aliasing of names is usually something to avoid.

Access to functions with in out parameters: implementation status

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 12 Oct 2011 06:25:38 -0400
Subject: Re: Ada 2012 and type access to in out functions
Newsgroups: comp.lang.ada

> Using GNAT GPL 2011, I try to define a type that is an access to an in out function, and the following example code [1] is illegal for the compiler.

> Is it an Ada 2012 restriction on using in out functions, a GNAT GPL 2011 limitation or a bug ?

Same error in GNAT 6.4.2. I'd guess it's a GNAT bug; they just haven't implemented that part of Ada 2012 yet.

On task discriminants and invariant expressions in SPARK

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Mon, 31 Oct 2011 13:38:21 +0100
Subject: [Q] task discriminants and invariant expressions in SPARK
Newsgroups: comp.lang.ada

In a subprogram (named Test) local to a task, the body is a case distinction that depends only on the value of the task's discriminant. SPARK reports a flow error because referring to the discriminant yields an invariant expression. I am trying to understand why this is a flow error. Or maybe, why this flow is an error. I see that for invariant expression of a case statement, only one of the branches of the case statement will be computed per task subtype, and that the discriminant will have a statically know value. (I couldn't think of an easy workaround but that may be just me.)

Additionally, will the same flow error be reported for case expressions or---by extension of the argument---for expression functions?

```
package Tsk
--# own task Busy : Business;
--#     protected Data : Binary;
is

   type Binary is mod 2**8;
   pragma Atomic (Binary);

   type Name is (Fred, Barney);

   task type Business (Id : Name)
   --# global in out Data;
   --# derives Data from Data;
   -- , Id (constant, not entire_variable)
   is
      pragma Priority (5);
   end Business;

end Tsk;

package body Tsk is

   Data : Binary := 2#0#;

   subtype Freds_Business is Business
            (Id => Fred);
   Busy : Freds_Business;

   task body Business is

      Current_Data : Binary;

      procedure Test
      --# global out Data;
      --# derives Data from ;
      is
      begin
         case Id is
            when Fred =>
               Data := 2#1#;
            when Barney =>
               Data := 2#101#;
         end case;
      end Test;
   begin
      loop
         Current_Data := Data;
         if Current_Data = 2#0# then
            Test;
         end if;
      end loop;
   end Business;
end Tsk;
```

Examining the body of package Tsk …

```
18    case Id is
           ^
!!!   Flow Error   : 22: Value of
expression is invariant.
```

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Mon, 31 Oct 2011 14:04:14 +0100
Subject: Re: [Q] task discriminants and invariant expressions in SPARK
Newsgroups: comp.lang.ada

[…]

Apparently, the following seem to be O.K., no more flow error, but is it cheating?

```
procedure Test (V : Name) …
… case V is …

Test (Id);
```

Just one indirection…

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Thu, 3 Nov 2011 02:34:08 -0700
Subject: Re: task discriminants and invariant expressions in SPARK
Newsgroups: comp.lang.ada

[…]

I can't find anything relevant in the documentation - there's no list of known deficiences in the GPL release note.

It is probably worth reporting this even if you are not a supported customer - use spark@adacore.com as the address (and include "SPARK" in the subject line otherwise it gets dumped in the spam bucket).

[…]

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Thu, 03 Nov 2011 13:11:51 +0100
Subject: Re: task discriminants and invariant expressions in SPARK
Newsgroups: comp.lang.ada

[…]

Thanks for having a look. A report is now sent.

I have also tried a variation that uses a protected object in place of an atomic object. Same diagnostic message.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Thu, 03 Nov 2011 15:15:06 +0100
Subject: Re: task discriminants and invariant expressions in SPARK
Newsgroups: comp.lang.ada

[…]

I've got a reply! Both, discriminant-is-constant and discriminant-turned-variable through making it a parameter would be expected in the current RavenSPARK model, and deliberately. Good to know.

Middleware for Ada and C++ integration

From: leandrohbatista <leandrohbatista@gmail.com>
Date: Fri, 7 Oct 2011 09:22:31 -0700
Subject: Middleware options for Ada and visual C++ integration
Newsgroups: comp.lang.ada

[…]

I have a simulation written in Ada95 and a visual scenario (IHM) built in visual C++. I'd like to integrate these two programs.

I was wondering to use MS COM, but it's no longer supported in Windows 7…

So, which middleware or inter process communication do you suggest or have already used in this kind of application?

[…]

From: leandrohbatista <leandrohbatista@gmail.com>
Date: Fri, 7 Oct 2011 10:29:53 -0700
Subject: Re: Middleware options for Ada and visual C++ integration
Newsgroups: comp.lang.ada

[…] porting the HMI to Ada it's not a choice yet.

In fact, I'd like to setup a COM server in my Ada simulation and run the HMI in another PC for example using TCP/IP.

So, I'm looking for a middleware that could interface Ada and C++.

I started reading about DDS (data distribution service) and I found RTI DDS solution (http://www.rti.com/products/dds/index.html).

Before choosing a middleware, I'd like to hear from Ada community, what solutions are you using as inter process communications?

DDS, CORBA, PolyORB, DCOM, .NET, or are you developing your own solution (DYO)?

From: Per Sandberg <per.sandberg@bredband.net>
Date: Fri, 07 Oct 2011 19:42:58 +0200
Subject: Re: Middleware options for Ada and visual C++ integration
Newsgroups: comp.lang.ada

It all depends on budget and data volumes and rates.

Some options may be:

SOAP and Web-services using AWS and some MSVC framework did that some years ago fairly straight forward.

0MQ as messaging infrastructure and XML or JSON as data-carrier this is more bleeding edge, and I guess you will be surprised over the power in this stack.

Both technologies are fairly platform independent.

From: Per Sandberg <per.sandberg@bredband.net>
Date: Fri, 07 Oct 2011 20:19:08 +0200
Subject: Re: Middleware options for Ada and visual C++ integration
Newsgroups: comp.lang.ada

[…]

The solutions I know about are:

WEB services with SOAP:

Fairly straight forward and from the Ada side all is available in AWS.

CORBA: with PolyORB and TAU elegant.

But there seems to be some interoperability problems that may be related to sloppy coding in one of the applications.

DDS: using RTIDDS lots of stuff to learn. Requires some budget and is pure PUB/SUB.

0MQ/JSON: kind of DYO but elegant.

The above middleware choices are platform neutral and there may be more but these are the ones i got firs hand experience with.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 7 Oct 2011 21:54:11 +0200
Subject: Re: Middleware options for Ada and visual C++ integration
Newsgroups: comp.lang.ada

[…]

We (cbb software GmbH) have a commercial middleware for distributed process automation and control. It has Ada, C++ and other interfaces.

http://www.cbb-software.com/technicalinformation/labmapmiddlewareautomation/index.html

> In fact, I'd like to setup a COM server in my Ada simulation and run the HMI in another PC for example using TCP/IP.

Yes, this is a common case. Usually our customers have HMI designed in "strange ways", much worse than C++, e.g. VisualBasic, LabView, DiaDem etc.

We leave it be, replacing the data acquisition/exchange layer with our middleware.

> So, I'm looking for a middleware that could interface Ada and C++.

You can do that, however it's not very usual that the components communicate directly with each other. More frequently you would have something like model-view-controller, which is rather "vertical communication" (application to hardware) than "horizontal" (application to application) one. In effect applications do interact, but indirectly like in the MVC.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sat, 8 Oct 2011 08:29:30 -0700
Subject: Re: Middleware options for Ada and visual C++ integration
Newsgroups: comp.lang.ada

[…]

You might want to have a look at this:

http://www.inspirel.com/yami4/

YAMI4 is a messaging solution for distributed systems that natively supports (among others) Ada and C++. Visual C++ is specifically one of the target platforms.

The advantage of YAMI4 in your particular case might be that it is very lightweight in terms of binary size and run-time footprint and that it can be used with very little impact on your existing codebase.

Conference Calendar

Dirk Craeynest

K.U.Leuven. Email: Dirk.Craeynest@cs.kuleuven.be

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with ☺ denote events with close relation to Ada.

The information in this section is extracted from the on-line *Conferences and events for the international Ada community* at: http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

2012

☺ January 25-27 — 39th ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2012), Philadelphia, PA, USA. Topics include: all aspects of programming languages and systems, with emphasis on how principles underpin practice.

- January 23-24 — ACM SIGPLAN **Workshop on Partial Evaluation and Program Manipulation** (PEPM'2012). Topics include: Program and model manipulation techniques (such as: partial evaluation, slicing, symbolic execution, refactoring, ...); Program analysis techniques that are used to drive program/model manipulation (such as: abstract interpretation, termination checking, type systems, ...); Techniques that treat programs/models as data objects (including: metaprogramming, generative programming, model-driven program generation and transformation, ...); etc. Application of the above techniques including case studies of program manipulation in real-world (industrial, open-source) projects and software development processes, descriptions of robust tools capable of effectively handling realistic applications, benchmarking.
- January 28 — 7th ACM SIGPLAN **Workshop on Types in Language Design and Implementation** (TLDI'2012). Topics include: Type-based language support for safety and security; Types for interoperability; Type-based program analysis, transformation, and optimization; Dependent types and type-based proof assistants; Types for security protocols, concurrency, and distributed computing; Type-based specifications of data structures and program invariants; Type-based memory management; Proof-carrying code and certifying compilation; etc.

Jan 30 – Feb 03 — 10th **Australasian Symposium on Parallel and Distributed Computing** (AusPDC'2012), Perth, Australia. Topics include: Multicore; GPUs and other forms of special purpose processors; Middleware and tools; Parallel programming models, languages and compilers; Runtime systems; Reliability, security and dependability; Applications; etc.

♦ February 04 — **Ada at the Free and Open-Source Software Developers' European Meeting** (FOSDEM'2012), Brussels, Belgium. FOSDEM 2012 is a two-day event (Sat-Sun 04-05 February). This years' edition includes again an Ada Developer Room, organized by Ada-Belgium in cooperation with Ada-Europe, which will be held on Saturday 4 February.

Febuary 15-17 — 20th Euromicro **International Conference on Parallel, Distributed and Network-Based Computing** (PDP'2012), Garching near Munich, Germany. Topics include: Parallel Computing, Models and Tools, Advanced and Applications, etc.

February 16-17 — 4th **International Symposium on Engineering Secure Software and Systems** (ESSoS'2012), Eindhoven, The Netherlands. Topics include: security architecture and design for software and systems; specification formalisms for security artifacts; verification techniques for security properties; systematic support for security best practices; programming paradigms for security; processes for the development of secure software and systems; trade-off between security and other non-functional requirements; support for assurance, certification and accreditation; etc.

February 22-25 5th **India Software Engineering Conference** (ISEC'2012), Kanpur, India. Topics include: Testing and Static Analysis, Specification and Verification, Model Driven Software Engineering, Software Architecture and Design, Tools and Environments, Development Paradigms and Processes, Maintenance and Evolution, Quality Management, Component Based Software Engineering, Object-Oriented Analysis and Design, Distributed Software Development, Case Studies and Industrial Experience, Software Engineering Education, Mining Software Repositories, etc.

☺ Feb 29 – Mar 03 43rd ACM **Technical Symposium on Computer Science Education** (SIGCSE'2012), Raleigh, North Carolina, USA.

☺ March 03 3rd **Workshop on Determinism and Correctness in Parallel Programming** (WODET'2012), London, UK. Topics include: open questions on deterministic multiprocessing in programming languages, compilers, operating systems, runtime systems and architecture; language extensions for disciplined parallel programming models (deterministic, data race-free, etc.); architecture, operating system, runtime system and compiler support for parallel program correctness; concurrency debugging techniques; concurrency bug avoidance techniques; real-world experience with safe parallel programming models, systems, or tools; etc. Deadline for submissions: January 6, 2012.

Mar 24 – Apr 01 **European Joint Conferences on Theory and Practice of Software** (ETAPS'2012), Tallinn, Estonia. Events include: CC, International Conference on Compiler Construction; ESOP, European Symposium on Programming; FASE, Fundamental Approaches to Software Engineering; FOSSACS, Foundations of Software Science and Computation Structures; TACAS, Tools and Algorithms for the Construction and Analysis of Systems.

Mar 31 9th **International Workshop on Formal Engineering approaches to Software Components and Architectures** (FESCA'2012). Topics include: Modelling formalisms for the analysis of concurrent, embedded or model-driven systems assembled of components; Interface compliance (interface-to-interface and interface-to implementation) and contractual use of components; Techniques for prediction and formal verification of system properties, including static and dynamic analysis; Industrial case studies and experience reports; etc.

Mar 31–Apr 1 12th **International Workshop on Language Descriptions, Tools and Applications** (LDTA'2012). Topics include: software based on grammars in some form, typically language processing applications such as parsers, program analyzers, optimizers and translators; parser generation, attribute grammar systems, term/graph rewriting systems, and other grammar-related meta-programming tools, techniques, and formalisms; program analysis, transformation, generation, and verification, reverse engineering and re-engineering, refactoring and other source-to-source transformations, language definition and language prototyping, and debugging, profiling, IDE support, etc.

March 25-29 27th ACM **Symposium on Applied Computing** (SAC'2012), Riva del Garda, Trento, Italy.

☺ Mar 25-29 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2012). Topics include: Language design and implementation; Type systems, static analysis, formal methods; Integration with other paradigms; Aspects, components, and modularity; Distributed, concurrent or parallel systems; Interoperability, versioning and software adaptation; etc.

☺ Mar 25-29 **Track on Software Engineering** (SE'2012), Topics include: technologies, theories, and tools used for producing highly dependable software more effectively and efficiently; such as Safety, Security; Dependability and Reliability; Fault Tolerance and Availability; Architecture, Framework, and Design Patterns; Standards; Maintenance and Reverse Engineering; Quality Assurance; Verification, Validation, and Analysis; Formal Methods and Theories; Component-Based Development and Reuse; Empirical Studies, and Industrial Best Practices; Applications and Tools; Distributed, Embedded, Real-Time, Highly Dependable Systems; etc.

☺ Mar 25-29 **Track on Programming Languages** (PL'2012). Topics include: Compiling Techniques, Formal Semantics and Syntax, Garbage Collection, Language Design and Implementation, Languages for Modeling, Model-Driven Development, New Programming Language Ideas and Concepts, Practical Experiences with Programming Languages, Program Analysis and Verification, Programming Languages from All Paradigms, etc.

March 25-30 11th **International Conference on Aspect-Oriented Software Development** (AOSD'2012), Potsdam, Germany. Topics include: Complex systems; Software design and engineering; Programming languages (language design, compilation and interpretation, verification and static program analysis, ...); Varieties of modularity (model-driven development, generative programming, software product lines, contracts and components, ...); Tools (evolution and reverse engineering, crosscutting views, refactoring, ...); Applications (distributed and concurrent systems, middleware, ...); etc. Deadline for submissions: January 9, 2012 (demonstrations), January 13, 2012 (workshop papers).

March 27-30 16th **European Conference on Software Maintenance and Reengineering** (CSMR'2012), Szeged, Hungary. Topics include: the development of maintainable systems, and the evolution, migration and reengineering of the existing ones.

April 03-05 4th **NASA Formal Methods Symposium** (NFM'2012), Norfolk, Virginia, USA. Topics include: identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems; formal verification, including theorem proving, model checking, and static analysis; model-based development; techniques and algorithms for scaling formal methods, such as abstraction and symbolic methods, parallel and distributed techniques, ...; code generation from formally verified models; significant applications of formal methods to aerospace systems; etc.

☺ April 11-13 15th IEEE **International Symposium on Object/component/service-oriented Real-time distributed Computing** (ISORC'2012), Shenzhen, China. Topics include: Programming and system engineering (languages, model-driven development of high integrity applications, specification, design, verification, validation, maintenance, ...); System software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...); Applications (embedded systems (automotive, avionics, consumer electronics, etc), real-time object-oriented simulations, ...); System evaluation (timeliness, worst-case execution time, dependability, end-to-end QoS, fault detection and recovery time, ...); etc.

April 11-13 19th Annual IEEE **International Conference and Workshops on the Engineering of Computer Based Systems** (ECBS'2012), Novi Sad, Serbia. Topics include: Dependability, Safety, and Security; Distributed Systems Design & Architecture; ECBS Infrastructure (Tools, Platforms); Embedded Real-Time Software Systems; Model-based System Development; Verification & Validation; Reengineering & Reuse; Evolution & Change; etc. Deadline for early registration: February 22, 2012.

April 17-19 25th **Conference on Software Engineering Education and Training** (CSEET'2012), Nanjing, China. Topics include: Technology Transfer; Student projects and internships; Industry-academia collaboration models; Software engineering professionalism; Education & training for "real-world" Software Engineering practices; Evaluation of SE Curricula: Are We Still Relevant?; Training models in industry; Systems and Software Engineering; Teaching the Business of Software Engineering; etc. Deadline for submissions: January 10, 2012 (short papers, work in progress papers, posters).

April 23-26 24th Annual **Systems and Software Technology Conference** (SSTC'2012), Salt Lake City, UT, USA.

☺ May 21-25 26th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2012), Shanghai, China. Topics include: all areas of parallel and distributed processing, such as Parallel and distributed algorithms; Applications of parallel and distributed computing; Parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, parallel programming paradigms, programming environments and tools, etc. Deadline for submissions: January 11, 2012 (PhD forum).

> ☺ May 25 **Workshop on Multithreaded Architectures and Applications** (MTAAP'2012). Topics include: programming frameworks for multithreading in the form of languages and libraries, compilers, analysis and debugging tools to increase the programming productivity. Deadline for submissions: January 9, 2012 (papers) .
>
> May 25 13th **International Workshop on Parallel and Distributed Scientific and Engineering Computing** (PDSEC-12). Topics include: parallel and distributed computing techniques and codes; practical experiences using various parallel and distributed systems; loop and task parallelism; scheduling; compiler issues for scientific and engineering computing; scientific and engineering computing on parallel computers, multicores, GPUs, FPGAs, ...; etc.

☺ May 29-31 50th **International Conference on Objects, Models, Components, Patterns** (TOOLS Europe'2012), Prague, Czech Republic. Topics include: Object technology, programming techniques, languages, tools;

Language implementation techniques, compilers, run-time systems; Distributed and concurrent object systems, multicore programming; Program verification and analysis techniques; Trusted, reliable and secure components; Component-based programming, modeling, tools; Model-driven development; Empirical studies on programming models and techniques; Domain specific languages and language design; Industrial-strength experience reports; Real-time object-oriented programming and design; etc. Deadline for submissions: January 6, 2012 (abstracts), January 13, 2012 (full papers).

☺ June 02-09 — 34th **International Conference on Software Engineering** (ICSE'2012), Zurich, Switzerland. Deadline for submissions: February 17, 2012 (workshop papers, posters, informal demonstrations).

☺ June 09 — 5th **Workshop on Exception Handling** (WEH'2012). Topics include: Exceptions in the software life-cycle (specifications, architectural design, modelling and programming, verification, debugging, testing, refactoring, variability management, static analysis, etc); Exception handling for and with new software artefacts (aspects, components, etc); Exception handling in today's applications (distributed, web-based, cloud, etc); Empirical studies of exception handling; Design patterns and anti-patterns, architectural styles, and good programming practice; etc. Deadline for submissions: February 17, 2012 (papers).

♦ June 11-15 — 17th **International Conference on Reliable Software Technologies - Ada-Europe'2012**, Stockholm, Sweden. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN. Deadline for submissions: January 12, 2012 (industrial presentations).

☺ June 11-16 — 26th **European Conference on Object-Oriented Programming** (ECOOP'2012), Beijing, China. Topics include: all areas of object technology and related software development technologies, such as Analysis and design methods; Concurrent, parallel, distributed, and real-time systems; Language design and implementation; Modularity, aspects, features, components, services; Software development environments and tools; Static and dynamic software analysis; Type systems, formal methods; Software evolution; etc.

June 13-15 — 37th **USENIX Annual Technical Conference** (USENIX ATC'2012), Boston, MA, USA. Topics include: Distributed and parallel systems; Embedded systems; Reliability, availability, and scalability; Security, privacy, and trust; etc. Deadline for submissions: January 10, 2012 (abstracts), January 17, 2012 (full papers).

June 18-22 — 9th **International Conference on Integrated Formal Methods** (iFM'2012), Pisa, Italy. Topics include: the combination of (formal and semi-formal) methods for system development, covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice. Deadline for submissions: January 14, 2012 (papers).

June 25-27 — 11th **International Conference on Mathematics of Program Construction** (MPC'2012), Madrid, Spain. Topics of interest range from algorithmics to support for program construction in programming languages and systems, such as type systems, program analysis and transformation, programming-language semantics, security, etc. Deadline for submissions: January 9, 2012 (abstracts), January 16, 2012 (full papers).

June 27-29 — 12th **International Conference on Application of Concurrency to System Design** (ACSD'2012), Hamburg, Germany. Topics include: (industrial) case studies of general interest, gaming applications, automotive systems, (bio-)medical applications, internet and grid computing, etc.; synthesis and control of concurrent systems, (compositional) modeling and design, (modular) synthesis and analysis, distributed simulation and implementation, ...; etc. Deadline for submissions: January 13, 2012 (abstracts), January 20, 2012 (papers).

July 01-03 — 24th **International Conference on Software Engineering and Knowledge Engineering** (SEKE'2012), Redwood City, California, USA. Topics included: Integrity, Security, and Fault Tolerance; Reliability; Component-Based Software Engineering; Embedded Software Engineering; Reverse Engineering; Programming Languages and Software Engineering; Program Understanding; Software Assurance; Software dependability; Software economics; Software Engineering Tools and Environments; Software Maintenance and Evolution; Software product lines; Software Quality; Software Reuse; Software Safety; Software Security; Software Engineering Case Study and Experience Reports; etc. Deadline for submissions: March 1, 2012 (papers). Deadline for early registration: May 10, 2012.

☺ July 09-11	**GNU Tools Cauldron 2012**, Prague, Czech Republic. Sponsored by: AdaCore, Google, IBM. Topics included: gathering of GNU tools developers. Deadline for submissions: January 31, 2012 (abstracts).
☺ July 10-13	10th IEEE **International Symposium on Parallel and Distributed Processing with Applications** (ISPA'2012), Madrid, Spain. Topics included: Parallel and Distributed Algorithms, and Applications; High-performance scientific and engineering computing; Middleware and tools; Reliability, fault tolerance, and security; Parallel/distributed system architectures; Tools/environments for parallel/distributed software development; Novel parallel programming paradigms; Compilers for parallel computers; Distributed systems and applications; etc. Deadline for submissions: January 15, 2012 (papers).
July 16-20	36th Annual **International Computer Software and Applications Conference** (COMPSAC'2012), Izmir, Turkey. Topics include: Software life cycle, evolution, and maintenance; Formal methods; Software architecture and design; Reliability, metrics, and fault tolerance; Security; Real-time and embedded systems; Education and learning; Applications; etc. Deadline for submissions: January 15, 2012 (abstracts), January 31, 2012 (full papers), March 15, 2012 (workshop papers), April 20, 2012 (fast abstracts, posters, doctoral symposium papers).
July 18-20	17th Annual IEEE **International Conference on the Engineering of Complex Computer Systems** (ICECCS'2012), Paris, France. Topics included: Verification and validation, Model-driven development, Reverse engineering and refactoring, Design by contract, Agile methods, Safety-critical & fault-tolerant architectures, Real-time and embedded systems, Tools and tool integration, Industrial case studies, etc. Deadline for submissions: February 1, 2012 (abstracts), February 15, 2012 (papers). Deadline for early registration: May 30, 2012.
August 27-31	18th **International Symposium on Formal Methods** (FM'2012), Paris, France. Theme: "Interdisciplinary Formal Methods". Topics include: Interdisciplinary formal methods (techniques, tools and experiences demonstrating formal methods in interdisciplinary frameworks); Formal methods in practice (industrial applications of formal methods, experience with introducing formal methods in industry, tool usage reports, etc); Tools for formal methods (advances in automated verification and model-checking, integration of tools, environments for formal methods, etc); Role of formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, method integration, qualitative or quantitative improvements); Theoretical foundations (all aspects of theory related to specification, verification, refinement, and static and dynamic analysis); Teaching formal methods (original contributions that provide insight, courses of action regarding the teaching of formal methods, teaching experiences, educational resources, integration of formal methods into the curriculum, etc). Deadline for submissions: March 5, 2012 (papers).
September 10-13	8th **International Conference on Open Source Systems** (OSS'2012), Hammamet, Tunisia. Theme: "Long-Term Sustainability with OSS". Deadline for submissions: March 9, 2012 (research papers, industry papers, formal tool demonstrations, lightning talks, posters, doctoral symposium), March 16, 2012 (workshops) May 25, 2012 (panels, tutorials). Deadline for early registration: June 15, 2012.
October 08-11	31st IEEE **International Symposium on Reliable Distributed Systems** (SRDS'2012), Irvine, California, USA. Topics include: distributed systems design, development and evaluation, with emphasis on reliability, availability, safety, security, trust and real time; high-confidence and safety-critical systems; distributed objects and middleware systems; formal methods and foundations for dependable distributed computing; evaluations of dependable distributed systems; etc. Deadline for submissions: March 9, 2012 (workshops), March 26, 2012 (abstracts), April 2, 2012 (papers), June 25, 2012 (workshop papers).
♦ November	ACM SIGAda **Annual International Conference on Ada and Related Technologies** (SIGAda'2012), Boston, Massachusetts, USA.
December 10	**Birthday of Lady Ada Lovelace**, born in 1815. Happy Programmers' Day!

Preliminary Call for Participation
Ada Developer Room at FOSDEM 2012
4 February 2012, Brussels, Belgium

Organized by Ada-Belgium
in cooperation with Ada-Europe

FOSDEM[1], the Free and Open source Software Developers' European Meeting, is a free and non-commercial two-day annual event organized in Brussels, Belgium. The 2012 edition will take place on Saturday 4 and Sunday 5 February, 2012. Ada-Belgium[2] organizes a series of presentations related to Ada and Free Software, to be held in a Developer Room on the first day of the event.

Preliminary overview:

- An introduction to Ada 2005 and Ada 2012, *by Jean-Pierre Rosen, Adalog*
- The contract model of Ada 2012, *by Jean-Pierre Rosen, Adalog*
- A historical perspective on GNAT: a successful FLOSS project, *by Robert Dewar, AdaCore*
- Lovelace: towards a full Ada OS, *by Xavier Grave, Centre National de la Recherche Scientifique*
- Multicore programming support in Ada, *(presenter to be confirmed)*
- Programming LEGO MINDSTORMS robots in Ada, *by Jose Ruiz, AdaCore*
- Ada in the on-line multi-user game Crimeville, *by Jacob Sparre Andersen, Research & Innovation*
- Programming Arduinos in Ada, *by Jacob Sparre Andersen, Research & Innovation*
- Ada on Rails, *by David Sauvage, AdaLabs Ltd.*
- PPETP: a P2P streaming protocol implemented in Ada, *by Riccardo Bernardini, University of Udine*
- SPARK: a free language and toolset for high-assurance software, *by engineer from Altran Praxis (to be confirmed)*

More details are available on the Ada at FOSDEM 2012 web-page, such as the full list with abstracts of presentations, biographies of speakers, and the concrete schedule. For the latest information, see:

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/12/120204-fosdem.html

The FOSDEM Team of Ada-Belgium

---

1 http://www.fosdem.org
2 http://www.cs.kuleuven.be/~dirk/ada-belgium

17^{th} International Conference on Reliable Software Technologies

Ada-Europe 2012

11-15 June 2012, Stockholm, Sweden

http://www.ada-europe.org/conference2012

Advance Information

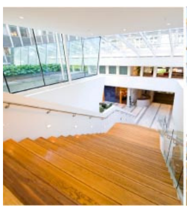
The 17^{th} International Conference on Reliable Software Technologies (Ada-Europe 2012) will take place in Stockholm, Sweden. This conference is the forthcoming edition in a series of annual international conferences, regularly held since the early 80's, under the auspices of, and organized by, Ada-Europe.

Ada-Europe 2012 provides a unique opportunity for interaction and collaboration between academics and industrial practitioners.

About the Venue

Stockholm, one of the most beautiful capitals in the world, is built on 14 islands around one of Europe's largest and best-preserved mediaeval city centres, located by the Baltic Sea coast. Stockholm is also Scandinavia's financial center with the largest gross regional product and highest amount of international companies.

In 2010, Stockholm was the first city to receive the European Green Capital award, an initiative of the EU commission, and is ranked fourth in the "Cities of opportunity" analysis, ranking first in intellectual capital and innovation health, safety and security demographics and livability.



The Ada-Europe conference will take place at Näringslivets Hus, a modern conference centre situated in the very heart of Stockholm, located near the Östermalmstorg metro station and close to the Gamla Stan historic district.

Program

Following tradition, the conference will span a full week, including a three-day technical program with the latest scientific advances in reliable software technologies and Ada. Attendees will have a varied choice of half-day and full-day tutorials that will be offered on Monday and Friday, either side of the central days of the conference. Tutorials consist of courses given by recognised experts in their respective fields, which deal with up-to-date technologies for the development of reliable software. Ada-Europe 2012 will also encompass panels and parallel industrial and vendor tracks.

The program of the conference will offer ample time for interaction and networking, with extensive lunch and coffee periods and a banquet being held on Wednesday, at Östermalms Saluhall, a marketplace food hall in a magnificent building from 1888.

Ada-Europe 2012 will build on the success of the 2011 event, in Edinburgh, UK, on June 20-24, which attracted over 130 delegates coming from Belgium, Brazil, Canada, Denmark, Egypt, Finland, France, Germany, Israel, Italy, Norway, Poland, Portugal, Russia, Slovakia, South Africa, Spain, Sweden, Switzerland, The Netherlands, UK and USA, representing more than 20 universities and 50 companies.

Further Information

The conference website at `http://www.ada-europe.org/conference2012` will provide full and up-to-date details of the program, venue and social program, accommodation and travel advice. For exhibiting and sponsoring details please contact the Conference Chair, Ahlan Marriott, at ahlan@ada-switzerland.ch.

Organization

Conference Chair

Ahlan Marriott
White Elephant GmbH, Switzerland
ahlan@ada-switzerland.ch

Program Co-Chairs

Mats Brorsson
KTH Royal Institute of Technology, Sweden
matsbror@kth.se

Luís Miguel Pinho
CISTER Research Centre/ISEP, Portugal
lmp@isep.ipp.pt

Tutorial Chair

Albert Llemosí
Universitat de les Illes Balears, Spain
albert.llemosi@uib.cat

Industrial Chair

Jørgen Bundgaard
Rovsing A/S, Denmark
jbg@rovsing.dk

Publicity Chair

Dirk Craeynest
Aubay Belgium & K.U.Leuven, Belgium
dirk.craeynest@cs.kuleuven.be

Local Chair

Rei Stråhle
Ada-Sweden
rei@ada-sweden.org

In-Cooperation

ACM SIGAda, SIGBED, SIGPLAN

Preliminary Call for Technical Contributions
SIGAda 2012

ACM Annual International Conference
on Ada and Related Technologies:
Engineering Safe, Secure, and Reliable Software

Boston, Massachusetts USA
Autumn 2012

Submission Deadline: **June 29, 2012**
Sponsored by ACM SIGAda (*ACM approval pending*)
http://www.acm.org/sigada/conf/sigada2012

SUMMARY: Reliability, safety, and security are among the most critical requirements of contemporary software. The application of software engineering methods, tools, and programming languages all interrelate to affect how and whether these requirements are met.

Such software is in operation in many application domains. Much has been accomplished in recent years, but much remains to be done. Our tools, methods, and languages must be continually refined; our management process must remain focused on the importance of reliability, safety, and security; our educational institutions must fully integrate these concerns into their curricula.

The conference will gather industrial and government experts, educators, software engineers, and researchers interested in developing, analyzing, and certifying reliable, safe, long-lived, secure software. We are soliciting technical papers and experience reports with a focus on, or comparison with, Ada.

We are especially interested in experience in integrating these concepts into the instructional process at all levels.

POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

- Ada 2012
- Ada and SPARK in universities
- Language selection for highly reliable systems
- Mixed-language development
- Ada and multicore
- Use of high reliability profiles such as Ravenscar
- Software safety standards such as DO-178B and DO-178C
- System of Systems
- Real-time networking/quality of service guarantees
- Analysis, testing, and validation
- Use of ASIS for new Ada tool development
- High-reliability development experience reports
- Static and dynamic analysis of code
- Integrating COTS software components
- System Architecture & Design including Service-Oriented Architecture and Agile Development
- Information Assurance
- Ada products certified against Common Criteria / Common Evaluation Methodology
- Distributed systems
- Fault tolerance and recovery
- Comparisons with other language technologies
- Cyber Security and Ada

KINDS OF TECHNICAL CONTRIBUTIONS:

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM *Ada Letters*. The Proceedings will be entered into the widely-consulted ACM Digital Library accessible online to university campuses, ACM's 100,000 members, and the software community.

EXTENDED ABSTRACTS discuss current work for which early submission of a full paper may be premature. If your abstract is accepted, you will be expected to produce a full paper, which will appear in the proceedings. Extended abstracts will be double-blind refereed. In 5 pages or less, clearly state the work's contribution, its relationship with previous work by you and others (with bibliographic references), results to date, and future directions.

EXPERIENCE REPORTS present timely results on the application of Ada and related technologies. Submit a 1-2 page description of the project and the key points of interest of project experiences. Descriptions will be published in the final program or proceedings, but a paper will not be required.

PANEL SESSIONS gather a group of experts on a particular topic who present their views and then exchange views with each other and the audience. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

WORKSHOPS are focused work sessions, which provide a forum for knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. A list of planned workshops and requirements for participation will be published in the Advance Program. Workshop proposals, up to 5 pages in length, will be selected by the Program Committee based on their applicability to the conference and potential for attracting participants.

TUTORIALS offer the flexibility to address a broad spectrum of topics relevant to Ada, and those enabling technologies which make the engineering of Ada applications more effective. Submissions will be evaluated based on relevance, suitability for presentation in tutorial format, and presenter's expertise. Tutorial proposals should include the expected level of experience of participants, an abstract or outline, the qualifications of the instructor(s), and the length of the tutorial (half-day or full-day).

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation to the Industrial Committee Chair by August 1st 2012. The authors of selected presentations shall prepare a final short abstract and submit it to the Committee Chair by October 1st, 2012, aiming at a 30-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the ACM Ada Letters.

HOW TO SUBMIT: For details on proposal deadlines and how to submit, please visit the conference website: www.acm.org/sigada/conf/sigada2012

FURTHER INFORMATION:

CONFERENCE GRANTS FOR EDUCATORS: The ACM SIGAda Conference Grants program is designed to help educators introduce, strengthen, and expand the use of Ada and related technologies in school, college, and university curricula. The Conference welcomes a grant application from anyone whose goals meet this description. The benefits include full conference registration with proceedings and registration costs for 2 days of conference tutorials/workshops. Partial travel funding is also available from AdaCore to faculty and students from GNAT Academic Program member institutions, which can be combined with conference grants. For more details visit the conference web site or contact Prof. Michael B. Feldman (MFeldman@gwu.edu)

OUTSTANDING STUDENT PAPER AWARD: An award will be given to the student author(s) of the paper selected by the program committee as the outstanding student contribution to the conference.

SPONSORS AND EXHIBITORS: For information about becoming a sponsor and/or exhibitor at SIGAda 2012, please visit the conference website: www.acm.org/sigada/conf/sigada2012

IMPORTANT INFORMATION FOR NON-US SUBMITTERS: International registrants should be particularly aware and careful about visa requirements, and should plan travel well in advance. Please visit the conference website for detailed information pertaining to visas.

ANY QUESTIONS?

Please contact the SIGAda Vice-Chair Conferences/Meetings, Alok Srivastava (Alok.Srivastava@auatac.com), the Conference Chair, Ben Brosgol (brosgol@adacore.com), or the SIGAda Chair, Ricky E. Sward (rsward@mitre.org).

Rationale for Ada 2012: 1 Contracts and aspects

John Barnes

John Barnes Informatics, 11 Albert Road, Caversham, Reading RG4 7AN, UK; Tel: +44 118 947 4125; email: jgpb@jbinfo.demon.co.uk

Abstract

This paper describes the mechanisms for including contracts in Ada 2012.

The main feature is that preconditions and postconditions can be given for subprograms. In addition, invariants can be given for types and predicates can be given for subtypes.

In attempting to find a satisfactory way of adding these features it was found expedient to introduce the concept of an aspect specification for describing properties of entities in general. It is thus convenient to describe aspect specifications in this paper.

Keywords: rationale, Ada 2012.

1 Overview of changes

The WG9 guidance document [1] identifies very large complex systems as a major application area for Ada. It further identifies four areas for improvements, one of which is

> Improving the ability to write and enforce contracts for Ada entities (for instance, via preconditions).

The idea of contracts has been a cornerstone of programming for many years. The very idea of specifying parameters for subroutines is a simple form of contract going back to languages such as Fortran over half a century ago.

More recently the idea of contracts has been brought to the fore by languages such as SPARK and Eiffel.

SPARK is, as many readers will be aware, a subset of Ada with annotations providing assertions regarding state embedded as Ada comments. The subset excludes features such as access types and dynamic dispatching but it does include Ravenscar tasking and generics. The subset was chosen to enable the contracts to be proved prior to execution. Thus SPARK is a very appropriate vehicle for real programs that just have to be correct because of concerns of safety and security.

Eiffel, on the other hand, is a language with a range of dynamic facilities much as in Ada and has found favour as a vehicle for education. Eiffel includes mechanisms describing contracts which are monitored on a dynamic basis at program execution.

The goal of this amendment to Ada is to incorporate matters such as pre- and postconditions but with the recognition that they are, like those in Eiffel, essentially for checking at runtime.

Adding pre- and postconditions and similar features has had quite a wide ranging impact on Ada and has required much more flexibility in many areas such as the form of expressions which will be addressed in later papers.

The following Ada issues cover the key changes and are described in detail in this paper:

145 Pre- and postconditions

146 Type invariants

153 Subtype predicates

183 Aspect specifications

191 Aliasing predicates

228 Default initial values for types

229 Specifiable aspects

230 Inheritance of null procedures with precondition

243 Clarification of categorization

247 Preconditions, postconditions, multiple inheritance and dispatching calls

250 Thoughts on type invariants

254 Do we really have contracts right?

267 Improvements for aspect specifications

These changes can be grouped as follows.

First we lay the syntactic foundations necessary to introduce features such as preconditions by discussing aspect specifications which essentially replace or provide an alternative to pragmas for specifying many features (183, 229, 243, 267).

Then we discuss the introduction of pre- and postconditions on subprograms including the problems introduced by multiple inheritance (145, 230, 247, 254).

Two other related topics are type invariants and subtype predicates which provide additional means of imposing restrictions on types (146, 153, 250).

Finally, two auxiliary features are the ability to provide default values for scalar types and array types (228) and means of checking that aliasing does not occur between two objects (191).

2 Aspect specifications

Although in a sense the introduction of aspect specifications is incidental to the main themes of Ada 2012 which are contracts, real-time, and containers, the clarity (and some might say upheaval) brought by aspect specifications merits their description first.

An early proposal to introduce preconditions was by the use of pragmas. Thus to give a precondition **not** Is_Full to the usual Push procedure acting on a stack S and a corresponding postcondition **not** Is_Empty, it was proposed that this should be written as

```
pragma Precondition(Push, not Is_Full(S));
pragma Postcondition(Push, not Is_Empty(S));
```

But this looks ugly and is verbose since it mentions Push in both pragmas. Moreover, potential problems with overloading means that it has to be clarified to which procedure Push they apply if there happen to be several. As a consequence it was decreed that the pragmas had to apply to the immediately preceding subprogram. Which of course is not the case with pragma Inline which with overloading applies to all subprograms with the given name. Other curiosities include the need to refer to the formal parameters of Push (such as S) so that the expression has to be resolved taking heed of these even though it is detached from the actual specification of Push.

Other pragmas proposed were Inherited_Precondition and Inherited_Postcondition for use with dispatching subprograms.

So it was a mess and an alternative was sought. The solution which evolved was to get away from wretched pragmas in such circumstances. Indeed, the Ada 83 Rationale [2] says "In addition, a program text can include elements that have no influence on the meaning of the program but are included as information and guidance for the human reader or for the compiler. These are: Comments; Pragmas..."

So pragmas were meant to have no effect on the meaning of the program. Typical pragmas in Ada 83 were List, Inline, Optimize and Suppress. But in later versions of Ada, pragmas are used for all sorts of things. The days when pragmas had no effect are long gone!

The basic need was to tie the pre- and postconditions syntactically to the specification of Push so that there could be no doubt as to which subprogram they applied; this would also remove the need to mention the name of the subprogram again. And so, as described in the introductory paper (in the previous issue of this esteemed journal) we now have

```
procedure Push(S: in out Stack; X: in Item)
  with
    Pre => not Is_Full(S),
    Post => not is Empty(S);
```

The syntax for aspect specification is

```
aspect_specification ::=
  with aspect_mark [ => expression] { ,
       aspect_mark [ => expression] }
```

and this can be used with a variety of structures, subprogram declaration being the example here.

Note especially the use of the reserved word **with**. Serious attempts were made to think of another word so as to avoid using **with** again but nothing better was suggested.

It might be thought that it would be confusing to use **with** which is firmly associated with context clauses. However, recall that **with** has also been used to introduce generic formal subprogram parameters without causing confusion since 1983. Thus

```
generic
  with function This ...
procedure That ...
```

Moreover, Ada 95 introduced the use of **with** for type extension as in

```
type Circle is new Object with
  record
    Radius: Float;
  end record;
```

So in Ada 95 there were already three distinct uses and a fourth one will surely do no harm. It's a versatile little word.

Any risk of confusion is easily avoided by using a sensible layout. Thus a **with** clause should start on a new line at the left and aligned with the following unit to which it applies. A formal generic parameter starting with **with** should be aligned with other formal parameters and indented after the word generic. In the case of type extension, **with** should be at the end of the line. Finally, in the case of aspect specifications, **with** should be at the beginning of a line and indented after the entity to which it applies.

Having introduced aspect specifications which are generally so much nicer than pragmas, it was decided to allow aspect specifications for all those situations where pragmas are used and an aspect specification makes sense. And then to make most of the pragmas obsolete.

Before looking at the old pragmas concerned in detail, two general points are worth noting.

The usual linear elaboration rules do not apply to the expression in an aspect specification. It is essentially sorted out at the freezing point of the entity to which the aspect applies. The reason for this was illustrated by an example in the Introduction which was

```
type Stack is private
  with
    Type_Invariant => Is_Unduplicated(Stack);
```

The problem here is that the function Is_Unduplicated cannot be declared before that of the type Stack and yet it is needed in the aspect specification of the declaration of Stack. So there is a circularity which is broken by saying that the elaboration of aspect specifications is deferred.

The other general point is that some aspects essentially take a Boolean value. For example the pragma Inline is replaced by the aspect Inline so that rather than writing

```
procedure Do_It( ... );
pragma Inline(Do_It);
```

we now write

```
procedure Do_It( ... )
  with Inline;
```

The aspect Inline has type Boolean and so we could write

```
procedure Do_It( ... )
  with Inline => True;
```

To have insisted on this would have been both pedantic and tedious and so in the case of a Boolean aspect there is a rule that says that => True can be omitted and True is then taken by default. Note however that omitting the whole aspect by just writing

```
procedure Do_It( ... );
```

results of course in the Inline aspect of Do_It being False.

A mad programmer could even use defaults for preconditions and postconditions. Thus writing

```
procedure Curious( ... )
  with Pre;
```

in which by default the precondition is taken to be True, results in the Curious procedure always being called.

We will now consider the fate of the various pragmas in Ada 2005. Some are replaced by aspect specifications and the pragmas made obsolete (of course, they can still be used, but should be discouraged in new programs). Some are paralleled by aspect specifications and the user left with the choice. Some are unchanged since for various reasons aspect specifications were inappropriate. Some pragmas are new to Ada 2012 and born obsolete.

The following are the obsolete pragmas with some examples of corresponding aspect specifications

The pragmas Inline, No_Return, and Pack are examples having Boolean aspect. We can now write

```
procedure Do_It( ... )
  with Inline;
procedure Fail( ... )
  with No_Return;
type T is ...
  with Pack;
```

Some thought was given as to whether the name of the Pack aspect should be Packing rather than Pack because this gave better resonance in English. But the possible confusion in having a different name to that of the pragma overrode the thought of niceties of (human) language.

Curiously enough the old pragmas Inline and No_Return could take several subprograms as argument but naturally the aspect specification is explicitly given to each one.

If several aspects are given to a procedure then we simply put them together thus

```
procedure Kill
  with Inline, No_Return;
```

rather than having to supply several pragmas (which careless program maintenance might have scattered around).

In the case of a procedure without a distinct specification, the aspect specification goes in the procedure body before **is** thus

```
procedure Do_It( ... )
  with Inline is
  ...
begin
  ...
end Do_It;
```

This arrangement is because the aspect specification is very much part of the specification of the subprogram. This will be familiar to users of SPARK where we might have

```
procedure Do_It( ... )
--# global in out Stuff;
is ...
```

If a subprogram has a distinct specification then we cannot give a language-defined aspect specification on the body; this avoids problems of conformance. If there is a stub but no specification then any aspect specification goes on the stub but not the body. Thus aspect specifications go on the first of specification, stub, and body but are never repeated. Note also that we can give aspect specifications on other forms of stubs and bodies such as package bodies, task bodies and entry bodies but none are defined by the language.

In the case of a stub, abstract subprogram, and null subprogram which never have bodies, the aspect specification goes after **is separate**, **is abstract** or **is null** thus

```
procedure Action(D: in Data) is separate
  with Convention => C;
procedure Enqueue( ... ) is abstract
  with Synchronization => By_Entry;
procedure Nothing is null
  with Something;
```

The above example of the use of Synchronization is from the package Synchronized_Queue_Interrfaces, a new child of Ada.Containers as mentioned in the Introduction.

The same style is followed by the newly introduced expression functions thus

```
function Inc (A: Integer) return Integer is (A + 1)
   with Inline;
```

Other examples of Boolean aspects are Atomic, Volatile, and Independent. We now write for example

```
Converged: Boolean := False
  with Atomic;
```

The aspects Atomic_Components, Volatile_Components and Independent_Components are similar.

The three pragmas Convention, Import and Export are replaced by five aspects, namely Import, Export, Convention, External_Name and Link_Name.

For example, rather than, (see [3] page 702)

```
type Response is access procedure (D: in Data);
pragma Convention(C, Response);
procedure Set_Click(P: in Response);
pragma Import(C, Set_Click);
procedure Action(D: in Data) is separate;
pragma Convention(C, Action);
```

we now more neatly write

```
type Response is access procedure (D: in Data)
  with Convention => C;
procedure Set_Click(P: in Response)
  with Import, Convention => C;
procedure Action(D: in Data) is separate
  with Convention => C;
```

Note that the aspects can be given in any order whereas in the case of pragmas, the parameters had to be in a particular order. We could have written **with** Import => True but that would have been pedantic. As another example (see the RM 7.4), instead of

```
CPU_Identifier: constant String(1 .. 8);
pragma Import(Assembler,
                    CPU_Identifier, Link_Name => "CPU_ID");
```

we now have

```
CPU_Identifier: constant String(1 .. 8)
  with Import, Convention => Assembler,
        Link_Name => "CPU_ID";
```

Observe that we always have to give the aspect name such as Convention whereas with pragmas Import and Export, the parameter name Convention was optional. Clearly it is better to have to give the name.

The pragma Controlled which it may be recalled told the system to keep its filthy garbage collector off my nice access type is plain obsolete and essentially abandoned. It is doubted whether it was ever used. The subclause of the RM (13.11.3) relating to this pragma is now used by a new pragma Default_Storage_Pools which will be discussed in a later paper.

The pragma Unchecked_Union is another example of a pragma replaced by a Boolean aspect. So we now write

```
type Number(Kind: Precision) is
  record
    ...
  end record
  with Unchecked_Union;
```

Many obsolete pragmas apply to tasks. The aspect Storage_Size takes an expression of any integer type. Thus in the case of a task type without a task definition part (and thus without **is** and matching **end**) we write

```
task type T
  with Storage_Size => 1000;
```

In the case of a task type with entries we write

```
task type T
  with Storage_Size => 1000 is
  entry E ...
  ...
end T;
```

The interrupt pragmas Attach_Handler and Interrupt_Handler now become

```
procedure P( ... )
  with Interrupt_Handler;
```

which specifies that the protected procedure P can be a handler and

```
procedure P( ... )
  with Attach_Handler => Some_Id;
```

which actually attaches P to the interrupt Some_Id.

The pragmas Priority and Interrupt_Priority are replaced by corresponding aspect specifications for example

```
task T
  with Interrupt_Priority => 31;
protected Object
  with Priority => 20 is                -- ceiling priority
```

Note that a protected type or singleton protected object always has **is** and the aspect specification goes before it.

Similarly, instead of using the pragma Relative_Deadline we can write

```
task T
  with Relative_Deadline => RD;
```

The final existing pragma that is now obsolete is the pragma Asynchronous used in the Distributed Systems Annex and which can be applied to a remote procedure or remote access type. It is replaced by the Boolean aspect Asynchronous.

That covers all the existing Ada 2005 pragmas that are now obsolete.

Two new pragmas in Ada 2012 are CPU and Dispatching_Domain but these are born obsolete. Thus we can write either of

```
task My Task is
  pragma CPU(10);
```

or

```
task My_Task
  with CPU => 10 is
```

and similarly

```
task Your_Task is
  pragma Dispatching_Domain(Your_Domain);
```

or

```
task Your_Task
  with Dispatching_Domain => Your_Domain is
```

The reason for introducing these pragmas is so that existing tasking programs with copious use of pragmas such as Priority can use the new facilities in a similar style. It was considered inelegant to write

```
task My_Task
  with CPU => 10 is
  pragma Priority(5);
```

and a burden to have to change programs to

```
task My_Task
  with CPU => 10, Priority => 5 is
```

So existing programs, can be updated to

```
task My_Task is
  pragma CPU(10);
  pragma Priority(5);
```

(One other pragma that was never born was Implemented which turned into the aspect Synchronization often used to ensure that an abstract procedure is actually implemented by an entry as illustrated earlier.)

A number of existing pragmas are paralleled by aspect specifications but the pragmas are not made obsolete. Examples are the pragmas relating to packages such as Pure, Preelaborate, Elaborate_Body and so on.

Thus we can write either of

```
package P is
  pragma Pure(P);
end P;
```

or

```
package P
  with Pure is
end P;
```

The author prefers the former but some avant garde programmers might like to use the latter.

Note that Preelaborable_Initialization is unusual in that it cannot be written as an aspect specification for reasons that need not bother us.

Finally, there are many pragmas that do not relate to any particular entity and so for which an aspect specification would be impossible.

These include Assert and Assertion_Policy, Suppress and Unsuppress, Page and List, Optimize and Restrictions.

As well as replacing pragmas, aspect specifications can be used instead of aspect clauses.

For example rather than

```
type Byte is range 0 .. 255;
```

followed (perhaps much later) by

```
for Byte'Size use 8;
```

we can now write

```
type Byte is range 0 .. 255
  with Size => 8;
```

Similarly

```
type My_Float is digits 20
  with Alignment => 16;
Loose_Bits: array (1 .. 10) of Boolean
  with Component_Size => 4;
type Cell_Ptr is access Cell
  with Storage_Size => 500 * Cell'Size / Storage_Unit,
       Storage_Pool => Cell_Ptr_Pool;
S: Status
  with Address => 8#100#;
type T is delta 0.1 range –1.0 .. +1.0
  with Small => 0.1;
```

But we cannot use this technique to replace an enumeration representation clause or record representation clause. Thus although we can write

```
type RR is
  record
    Code: Opcode;
    R1: Register;
    R2: Register;
  end record
with Alignment => 2, Bit_Order => High_Order_First;
```

the layout information has to be done by writing

```
for RR use
  record
    Code at 0 range 0 .. 7;
    R1 at 1 range 0 .. 3;
    R2 at 1 range 4 .. 7;
  end record;
```

It is interesting to note that attribute definition clauses and at clauses were not made redundant in the way that many pragmas were made redundant. This is because there are things that one can do with attribute definition clauses that cannot be done with aspect specifications. For example a visible type can be declared in a visible part and then details of its representation can be given in a private part. Thus we might have

```
package P is
  type T is ...
private
  Secret_Size: constant := 16;
  for T'Size use Secret_Size;
end P;
```

It's not that convincing because the user can use the attribute T'Size to find the Secret_Size anyway. But some existing programs are structured like that and hence the facility could hardly be made redundant.

The examples above have shown aspect specifications with the following constructions: subprogram declaration, subprogram body, stub, abstract subprogram declaration, null procedure declaration, full type declaration, private type declaration, object declaration, package declaration, task type declaration, single task declaration, and single protected declaration. In addition they can be used with subtype declaration, component declaration, private extension declaration, renaming declaration, protected type declaration, entry declaration, exception declaration, generic declaration, generic instantiation, and generic formal parameter declaration.

The appropriate layout should be obvious. In the case of a large structure such as a package specification and any body, the aspect specification goes before **is**. But when something is small and all in one piece such as a procedure specification, stub, null procedure, object declaration or generic instantiation any aspect specification goes at the end of the declaration; it is then more visible and less likely to interfere with the layout of the rest of the structure.

In some cases such as exception declarations there are no language defined aspects that apply but implementations might define them.

3 Preconditions and postconditions

We will look first at the simple case when inheritance is not involved and then look at more general cases.

To apply a precondition Before and/or a postcondition After to a procedure P we write

```
procedure P(P1: in T1; P2: in out T2; P3: out T3)
  with Pre => Before,
       Post => After;
```

where Before and After are expressions of a Boolean type (that is of type Boolean or a type derived from it).

The precondition Before and the postcondition After can involve the parameters P1 and P2 and P3 and any visible entities such as other variables, constants and functions. Note that Before can involve an **out** parameter such as P3 (if necessary it will be copied in to enable this).

The attribute X'Old will be found useful in postconditions; it denotes the value of X on entry to P. Old is typically applied to parameters of mode **in out** such as P2 but it can be applied to any visible entity such as a global variable. This can be useful for monitoring global variables which are updated by the call of P. But note that 'Old can only be used in postconditions and not in arbitrary text and it cannot be applied to objects of a limited type.

Perhaps surprisingly 'Old can also be applied to parameters of mode **out**. For example, in the case of a parameter of a record type that is updated as a whole, nevertheless we might want to check that a particular component has not changed. Thus in updating some personal details, such as address and occupation, we might want to ensure that the person's date of birth and sex are not tampered with by writing

```
Post => P.Sex = P.Sex'Old and P.Dob = P.Dob'Old
```

In the case of an array, we can write A(I)'Old which is the same as A'Old(I'Old) which means the original value of A(I). But A(I'Old) is different since it is the component of the final value of A but indexed by the old value of I.

Remember that the result of a function is an object and so 'Old can be applied to it. Note carefully the difference between F(X)'Old and F(X'Old). The former applies F to X on entry to the subprogram and saves it. The latter saves X and applies F to it when the postcondition is evaluated. These could be different because the function F might also involve global variables which have changed.

Generally 'Old can be applied to anything but there are restrictions on its use in certain conditional structures in which it can only be applied to statically determined objects. The details will be given in a later paper when we look at expressions in general.

(The collector of Ada curiosities might be amused to note that we can write

```
subtype dlo is Character;
```

and then in a postcondition we could have

```
dlo'('I')'old
```

which is palindromic. If the subtype were blo rather than dlo then the expression would be mirror reflective!

I am grateful to Jean-Pierre Rosen for this example.)

In the case of a postcondition applying to a function F, the result of the function is denoted by the attribute F'Result. Again this attribute can only be used in postconditions.

Some trivial examples of declarations of a procedure Pinc and function Finc to perform an increment are

```
procedure Pinc(X: in out Integer)
   with Post => X = X'Old+1;
function Finc(X: Integer) return Integer
  with Post => Finc'Result = X'Old+1;
```

Preconditions and postconditions are controlled by the pragma Assertion_Policy. They are enabled by

```
pragma Assertion_Policy(Check);
```

and disabled by using parameter Ignore. It is the value in effect at the point of the subprogram declaration that matters. So we cannot have a situation where the policy changes during the call so that preconditions are switched on but postconditions are off or vice versa.

And so the overall effect of calling P with checks enabled is roughly that, after evaluating any parameters at the point of call, it as if the body were

```
if not Before then                    -- check precondition
   raise Assertion_Error;
end if;
evaluate and store any 'Old stuff;
call actual body of P;
if not After then                -- check postcondition
   raise Assertion_Error;
end if;
copy back any by-copy parameters;
return to point of call;
```

The exceptions Assertion_Error are propagated and so raised at the point of call; they cannot be handled inside P. Of course, if the evaluation of Before or After themselves raise some exception then that will similarly be propagated to the point of call.

Note that conditions Pre and Post can also be applied to entries.

Before progressing to the problems of inheritance it is worth reconsidering the purpose of pre- and postconditions.

> A precondition Before is an obligation on the caller to ensure that it is true before the subprogram is called and it is a guarantee to the implementer of the body that it can be relied upon on entry to the body.
>
> A postcondition After is an obligation on the implementer of the body to ensure that it is true on return from the subprogram and it is a guarantee to the caller that it can be relied upon on return.

The symmetry is neatly illustrated by the diagram below

	Pre	Post
Call writer	obligation	guarantee
Body writer	guarantee	obligation

The simplest form of inheritance occurs with derived types that are not tagged. Suppose we declare the procedure Pinc as above with the postcondition shown and supply a body

```
procedure Pinc(X: in out Integer) is
begin
   X := X+1;
end Pinc;
```

and then declare a type

```
type Apples is new Integer;
```

then the procedure Pinc is inherited by the type Apples. So if we then write

```
No_Of_Apples: Apples;
...
Pinc(No_Of_Apples);
```

what actually happens is that the code of the procedure Pinc originally written for Integer is called and so the postcondition is inherited automatically.

If the user now wants to add a precondition to Pinc that the number of apples is not negative then a completely new subprogram has to be declared which overrides the old one thus

```
procedure Pinc(X: in out Apples)
   with Pre => X >= 0,
        Post => X = X'Old+1;
```

and a new body has to be supplied (which will of course in this curious case be essentially the same as the old one). So we cannot inherit an operation and change its conditions at the same time.

We now turn to tagged types and first continue to consider the specific conditions Pre and Post. As a perhaps familiar example, consider the hierarchy consisting of a type Object and then direct descendants Circle, Square and Triangle.

Suppose the type Object is

```
type Object is tagged
  record
    X_Coord, Y_Coord: Float;
  end record;
```

and we declare a function Area thus

```
function Area(O: Object) return Float
  with Pre => O.X_Coord > 0.0,
       Post => Area'Result = 0.0;
```

This imposes a requirement on the caller that the function is called only with objects with positive *x*-coordinate (for some obscure reason), and a requirement on the implementer of the body that the area is zero (raw objects are just points and have no area).

If we now declare a type Circle as

```
type Circle is new Object with
  record
    Radius: Float;
  end record;
```

and override the inherited function Area then the Pre and Post conditions on Area for Object are not inherited and we have to supply new ones, perhaps

```
function Area(C: Circle)
  with Pre => C.X_Coord - C.Radius > 0.0,
       Post => Area'Result > 3.1 * C.Radius**2 and
               Area'Result < 3.2 * C.Radius**2;
```

The conditions ensure that all of the circle is in the right half-plane and that the area is about right!

So the rules so far are exactly as for the untagged case. If an operation is not overridden then it inherits the conditions from its ancestor but if it is overridden then those conditions are lost and new ones have to be supplied. And if no new ones are supplied then they are by default taken to be True.

In conclusion, the conditions Pre and Post are very much part of the actual body. One consequence of this is that an abstract subprogram cannot have Pre and Post conditions because an abstract subprogram has no body.

We now turn to the class wide conditions Pre'Class and Post'Class which are subtly different. The first point is that the class wide ones apply to all descendants as well even if the operations are overridden. In the case of Post'Class if an overridden operation has no condition given then it is taken to be True (as in the case of Post). But in the case of Pre'Class, if an overridden operation has no condition given then it is only taken to be True if no other Pre'Class applies (no other is inherited). We will now look at the consequences of these rules.

It might be that we want certain conditions to hold throughout the hierarchy, perhaps that all objects concerned have a positive *x*-coordinate and nonnegative area. In that case we can use class wide conditions.

```
function Area(O: Object) return Float
  with Pre'Class => O.X_Coord > 0.0,
       Post 'Class => Area'Result >= 0.0;
```

Now when we declare Area for Circle, Pre'Class and Post'Class from Object will be inherited by the function Area for Circle. Note that within a class wide condition a formal parameter of type T is interpreted as of T'Class. Thus O is of type Object'Class and thus applies to Circle. The inherited postcondition is simply that the area is not negative and uses the attribute 'Result.

If we do not supply conditions for the overriding Area for Circle and simply write

```
overriding
function Area(C: Circle) return Float;
```

then the precondition inherited from Object still applies. In the case of the postcondition not only is the postcondition from Object inherited but there is also an implicit postcondition of True. So the applicable conditions for Area for Circle are

Pre'Class for Object
Post'Class for Object
True

Suppose on the other hand that we give explicit Pre'Class and Post'Class for Area for Circle thus

```
overriding
function Area(C: Circle return Float)
  with Pre'Class => .... ,
       Post'Class => ... ;
```

We then find that the applicable conditions for Area for Circle are

Pre'Class for Object
Pre'Class for Circle
Post'Class for Object
Post'Class for Circle

Incidentally, it makes a lot of sense to declare the type Object as abstract so that we cannot declare pointless objects. In that case Area might as well be abstract as well. Although we cannot give conditions Pre and Post for an abstract operation we can still give the class wide conditions Pre'Class and Post'Class.

If the hierarchy extends further, perhaps Equilateral_Triangle is derived from Triangle which itself is derived from Object, then we could add class wide conditions to Area for Triangle and these would also apply to Area for Equilateral_Triangle. And we might add specific conditions for Equilateral_Triangle as well. So we would then find that the following apply to Area for Equilateral_Triangle

Pre'Class for Object
Pre'Class for Triangle
Pre for Equilateral Triangle
Post'Class for Object
Post'Class for Triangle
Post for Equilateral_Triangle

The postconditions are quite straightforward, all apply and all must be true on return from the function Area. The compiler can see all these postconditions when the code for Area is compiled and so they are all checked in the body. Note that any default True makes no difference because B **and** True is the same as B.

However, the rules regarding preconditions are perhaps surprising. The specific precondition Pre for Equilateral_Triangle must be true (checked in the body) but so long as just one of the class wide preconditions Pre'Class for Object and Triangle is true then all is well. Note that class wide preconditions are checked at the point of call. Do not get confused over the use of the word apply. They all apply but only the ones seen at the point of call are actually checked.

The reason for this state of affairs concerns dispatching and especially redispatching. Consider the case of Ada airlines which has Basic, Nice and Posh passengers. Basic passengers just get a seat. Nice passengers also get a meal and Posh passengers also get a limo. The types Reservation, Nice_Reservation and Posh_Reservation form a hierarchy with Nice_Reservation being extended from Reservation and so on. The facilities are assigned when a reservation is made by calling an appropriate procedure Make thus

```
procedure Make(R: in out Reservation) is
begin
  Select_Seat(R);
end Make;

procedure Make(NR: in out Nice_Reservation) is
begin
  Make(Reservation(NR));
  Order_Meal(NR);
end Make;

procedure Make(PR: in out Posh_Reservation) is
  Make(Nice_Reservation(PR));
  Arrange_Limo(PR);
end Make;
```

Each Make calls its ancestor in order to avoid duplication of code and to ease maintenance.

A variation involving redispatching introduces two different procedures Order_Meal, one for Nice passengers and one for Posh passengers. We then need to ensure that Posh passengers also get a posh meal rather than a nice meal. We write

```
procedure Make(NR: in out Nice_Reservation) is
begin
  Make(Reservation(NR));
        -- now redispatch to appropriate Order_Meal
  Order_Meal(Nice_Reservation'Class(NR));
end Make;
```

Now suppose we have a precondition Pre'Class on Order_Meal for Nice passengers and one on Order_Meal for Posh passengers. The call of Order_Meal sees that it is for Nice_Reservation'Class and so the code includes a test of Pre'Class on Nice_Reservation. It does not necessarily know of the existence of the type Posh_Reservation and cannot check Pre'Class on that Order_Meal. At a later date we might add Supersonic passengers (RIP Concorde) and this can be done without recompiling the rest of the system so it certainly cannot do anything about checking Pre'Class on Order_Meal for Supersonic_Reservation which does not exist when the call is compiled. So when we eventually get to the body of one of the procedures Order_Meal all we know is that some Pre'Class on Order_Meal has been checked somewhere. And that is all that the writer of the code of Order_Meal can rely upon. Note that nowhere does the compiled code actually "or" a lot of preconditions together.

In summary, class wide preconditions are checked at the point of call. Class wide postconditions and both specific pre- and postconditions are checked in the actual body.

A small point to remember is that a class wide operation such as

```
procedure Do_It(X: in out T'Class);
```

is not a primitive operation of T and so although we can specify Pre and Post for Do_It we cannot specify Pre'Class and Post'Class for Do_It.

We now turn to the question of multiple inheritance and progenitors.

We noted above that the aspects Pre and Post cannot be specified for an abstract subprogram because it doesn't have a body. They cannot be given for a null procedure either, since we want all null procedures to be identical and do nothing and that includes no conditions.

In the case of multiple inheritance we have to consider the so-called Liskov Substitutability Principle (LSP). The usual consequence of LSP is that in the case of preconditions they are combined with "or" (thus weakening) and the rule for postconditions is that they are combined with "and" (thus strengthening). But the important thing is that a relevant concrete operation can be substituted for the corresponding operations of all its relevant ancestors.

In Ada, a type T can have one parent and several progenitors. Thus we might have

```
type T is new P and G1 and G2 with ...
```

where P is the parent and G1 and G2 are progenitors. Remember that a progenitor cannot have components and cannot have concrete operations (apart possibly for null procedures). So the operations of the progenitors have to be abstract or null and cannot have Pre and Post conditions. However, they can have Pre'Class and Post'Class conditions. It is possible that the same operation Op is primitive for more than one of these. Thus the progenitors G1 and G2 might both have an operation Op thus

```
procedure Op(X: G1) is abstract;
procedure Op(X: G2) is abstract;
```

If they are conforming (as they are in this case) then the one concrete operation Op of the type T derived from both G1 and G2 will implement both of these. (If they don't conform then they are simply overloadings and two operations of T are required). Hence the one Op for T can be substituted for the Op of both G1 and G2 and LSP is satisfied.

Now suppose both abstract operations have pre- and postconditions. Take postconditions first, we might have

```
procedure Op(X: G1) is abstract
  with Post'Class => After1;

procedure Op(X: G2) is abstract
  with Post'Class => After2;
```

Users of the Op of G1 will expect the postcondition After1 to be satisfied by any implementation of that Op. So if using the Op of T which implements the abstract Op of G1, it follows that Op of T must satisfy the postcondition After1. By a similar argument regarding G2, it must also satisfy the postcondition After2.

It thus follows that the effective postcondition on the concrete Op of T is as if we had written

```
procedure Op(X: T)
  with Post'Class => After1 and After2;
```

But of course we don't actually have to write that since we simply write

```
overriding
procedure OP(X: T);
```

and it automatically inherits both postconditions and the compiler inserts the appropriate code in the body. Remember that if we don't give a condition then it is True by default but anding in True makes no difference.

If we do provide another postcondition thus

```
overriding
procedure OP(X: T)
  with Post'Class => After_T;
```

then the overall class wide postcondition to be checked before returning will be After1 and After2 and After_T.

Now consider preconditions. Suppose the declarations of the two versions of Op are

```
procedure Op(X: G1) is abstract
  with Pre'Class => Before1;
```

```
procedure Op(X: G2) is abstract
  with Pre'Class => Before2;
```

Assuming that there is no corresponding Op for P, we must provide a concrete operation for T thus

```
overriding
procedure Op(X: T)
  with Pre'Class => Before_T;
```

This means that at a point of call of Op the precondition to be checked is Before_T **or** Before1 **or** Before2. As long as this is satisfied it does not matter that Before1 and Before2 might have been different.

If we do not provide an explicit Pre'Class then the condition to be checked at the point of call is Before1 **or** Before2.

An interesting case arises if a progenitor (say G1) and the parent have a conforming operation. Thus suppose P itself has the operation

```
procedure Op(X: P);
```

and moreover that the operation is not abstract. Then (ignoring preconditions for the moment) this Op for P is inherited by T and thus provides a satisfactory implementation of Op for G1 and all is well.

Now suppose that Op for P has a precondition thus

```
procedure OP(X: P)
  with Pre'Class => Before_P;
```

and that Before_P and Before1 are not the same. This is rather confusing if we do not provide an explicit overriding for Op. So in this case there is a rule that an explicit overriding is required for Op for T.

If Op for P is abstract then a concrete Op for T must be provided and the situation is just as in the case for the Op for G1 and G2.

If T itself is declared as abstract (and P is not abstract and Op for P is concrete) then the inherited Op for T is abstract.

(These rules are similar to those for functions returning a tagged type when the type is extended; it has to be overridden unless the type is abstract in which case the inherited operation is abstract.)

We finish this somewhat mechanical discussion of the rules by pointing out that if silly inappropriate preconditions are given then we will get a silly program.

At the end of the day, the real point is that programmers should not write preconditions that are not sensible and sensibly related to each other. Because of the generality, the compiler cannot tell so stupid things are hard to prohibit. There is no defence against stupid programmers.

A concrete example using simple numbers might help. Suppose we have a tagged type T1 and an operation Solve which takes a parameter of type T1 and perhaps finds the solution to an equation defined by the components of T1. Solve delivers the answer in a parameter A with a parameter D giving the number of significant digits required in the answer. Also we impose a precondition on the number of digits D thus

```
type T1 is tagged record ...
procedure Solve(X: in T1; A: out Float; D: in Integer)
  with Pre'Class => D < 5;
```

The intent here is that the version of Solve for the type T1 always works if the number of significant digits asked for is less than 5.

Now suppose we declare a type T2 derived from T1 and that we override the inherited Solve with a new version that works if the number of significant digits asked for is less than 10

```
type T2 is new T1 with ...
overriding
procedure Solve(X: in T2; A: out Float; D: in Integer)
  with Pre'Class => D < 10;
```

And so on with a type T3

```
type T3 is new T2 with ...
overriding
procedure Solve(X: in T3; A: out Float; D: in Integer)
  with Pre'Class => D < 15;
```

Thus we have a hierarchy of algorithms Solve with increasing capability.

Now suppose we have a dispatching call

```
An_X: T1'Class := ... ;
Solve(An_X, Answer, Digs);
```

this will dispatch to one of the Solve procedures but we do not know which one. The only precondition that applies is that on the Solve for T1 which is D < 5. That is fine because D < 5 implies D < 10 and D < 15 and so on. Thus the preconditions work because the hierarchy weakens them.

Similarly, if we have

```
An_X: T2'Class := ... ;
Solve(An_X, Answer, Digs);
```

then it will dispatch to a Solve for one of T2, T3, ..., but not to the Solve for T1. The applicable preconditions are D < 5 and D < 10 and these are notionally ored together which means D < 10 is actually required. To see this suppose we supply D = Digs = 7. Then D < 5 is False but D < 10 is True so by oring False and True we get True, so the call works.

On the other hand if we write

```
An_X: T2 := ... ;
Solve(An_X, Answer, Digs);
```

then no dispatching is involved and the Solve for T2 is called. But both class wide preconditions D < 5 and D < 10 apply and so again the resulting ored precondition that is required is D < 10.

Now it should be clear that if the preconditions do not form a weakening hierarchy then we will be in trouble. Thus if the preconditions were D < 15 for T1, D < 10 for T2, and D < 5 for T3, then dispatching from the root will only check D < 15. However, we could end up calling the Solve for T2 which expects the precondition D < 10 and this might not be satisfied.

Care is thus needed with preconditions that they are sensibly related.

4 Type invariants

Type invariants are designed for use with private types where we want some relationship to always hold between components of the type. Like pre- and postconditions there are both specific invariants that can be applied to any type and class wide invariants that can only be applied to tagged types.

One example mentioned above and discussed in the Introduction was a type Stack with specific invariant Is_Unduplicated. Thus we write

```
type Stack is private
  with Type_Invariant  => Is_Unduplicated(Stack);
```

After calls of Push and Pop and any other operations that manipulate the stack, the function Is_Unduplicated is called to ensure that there are no duplicates on the stack.

The monitoring is controlled by the pragma Assertion_Policy in the same way as pre- and postconditions. If an invariant fails (that is, has value False) then Assertion_Error is raised.

The invariant Is_Unduplicated is a curious example because it cannot be violated by Pop anyway since if there were no duplicates then removing the top item cannot make one appear.

Moreover, Push needs to ensure that the item to be added is not a duplicate of one on the stack already and so essentially much of the checking is repeated. Indeed, when writing Push we should be able to assume that no items are already duplicated and hence all we need to do is check that the new item to be added is not equal to one of the existing items (so n comparisons). However, a general function Is_Unduplicated will need to compare all pairs and thus require a double loop (so $n(n+1)/2$ comparisons).

The reader is invited to meditate over this conundrum. One's first reaction might be that this is a bad example. However, one way to ensure reliability is to introduce redundancy. Thus if the encoding of Is_Unduplicated and Push are done independently then there is an increased probability that any error will be detected.

The aspect Type_Invariant requires an expression of a Boolean type. The mad programmer could therefore also write

```
type Stack is private
  with Type_Invariant;
```

which would thus be True by default and so useless! Actually it might not be entirely useless since it might act as a placeholder for an invariant to be defined later and meanwhile the program will compile and execute.

Type invariants are useful whenever a type is more than just the sum of its components. Note carefully that the invariant may not hold when an object is being manipulated by a subprogram having access to the full type. In the case of Push and Pop and the invariant Is_Unduplicated this will not happen but consider the following simple example.

Suppose we have a type Point which describes the position of an object in a plane. It might simply be

```
type Point is
  record
    X, Y: Float;
  end record;
```

Now suppose we want to ensure that all points are within a unit circle. We could ensure that a point lies within a square by means of range constraints by writing

```
type Point is
  record
    X, Y: Float range –1.0 .. +1.0;
  end record;
```

but we need to ensure that X**2 + Y**2 is not greater than 1.0, and that cannot be done by individual constraints. So we might declare a type Disc_Pt with an invariant as follows

```
package Places is

  type Disc_Pt is private
    with Type_Invariant => Check_In(Disc_Pt);
  function Check_In(D: Disc_Pt) return Boolean;
  ...                    -- various operations on disc points
private

  type Disc_Pt is
    record
      X, Y: Float range –1.0 .. +1.0;
    end record;
  function Check_In(D: Disc_Pt) return Boolean is
    (D.X**2 + D.Y**2 <= 1.0)
      with Inline;

end Places;
```

Note that we have used an expression function for Check_In. Expression functions were outlined in the Introduction and will be discussed in detail in the next paper. They are very useful for small functions in situations like this and typically will be given the aspect Inline as shown.

Now suppose that we wish to make available to the user a procedure Flip that reflects a Disc_Pt in the line $x = y$, or in other words interchanges its X and Y components. The body might be

```
procedure Flip(D: in out Disc_Pt) is
  T: Float;                    -- temporary
begin
  T := D.X;  D.X := D.Y;  D.Y := T;
end Flip;
```

This works just fine but note that just before the assignment to D.Y, it is quite likely that the invariant does not hold. If the original value of D was (0.1, 0.8) then at the intermediate stage it will be (0.8, 0.8) and so well outside the unit circle.

So there is a general principle that an intermediate value not visible externally need not satisfy the invariant. There is an analogy with numeric types. The intermediate value of an expression can fall outside the range of the type but will be within range when the final value is assigned to the object. For example, suppose type Integer is 16 bits (a small machine) but the registers perform arithmetic in 32 bits, then a statement such as

```
J := K * L / M;
```

could easily produce an intermediate result K * L outside the range of Integer but the final value could be in range.

In many cases it will not be necessary for the user to know that a type invariant applies to the type; it is after all merely a detail of the implementation. So perhaps the above should be rewritten as

```
package Places is
  type Disc_Pt is private;

  ...               -- various operations on disc points
private
  type Disc_Pt is
    record
      X, Y: Float range –1.0 .. +1.0;
    end record
    with Type_Invariant =>
              Disc_Pt.X**2 + Disc_Pt.Y**2 <= 1.0;

end Places;
```

In this case we do not need to declare a function Check_In at all. Note the use of the type name Disc_Pt in the invariant expression. This is another example of the use of a type name to denote a current instance (this is familiar from way back in Ada 83 with task type names).

We now turn to consider the places where a type invariant on a private type T is checked. These are basically when it can be changed from the point of view of the outside user. They are

- after default initialization of an object of type T,
- after a conversion to type T,
- after assigning to a view conversion having a part of type T,
- after a call of T'Read or T'Input,
- after a call of a subprogram declared in the immediate scope of T and visible outside that returns a result with a part of type T or has an **out** or **in out** or access parameter with a part of type T.

Note that by saying a part of type T, the checks not only apply to subprograms with parameters and results of type T but they also apply to parameters and results whose components are of the type T or are view conversions involving the type T.

Beware, however, that the checks do not extend to deeply nested situations, such as components with components that are access values to objects that themselves involve type T or worse. Thus there are holes in the protection offered by type invariants. However, if the types are straightforward and the writer does not do foolish things like surreptitiously export access types referring to T then all will be well. It is another example of there being no defence against foolish programmers.

The checks on type invariants regarding parameters and results can be conveniently implemented in the body of the subprogram in much the same way as for postconditions. This saves duplicating the code of the tests at each point of call.

If a subprogram such as Flip which is visible outside is called from inside then the checks still apply. This is not strictly necessary of course, but fits the simple model of the checks being in the body and so simplifies the implementation.

If an untagged type is derived then any existing specific invariant is inherited for inherited operations. However, a further invariant can be given as well and both will apply to the inherited operations. This fits in with the model of view conversions used to describe how an inherited subprogram works on derivation. The parameters of the derived type are view converted to the parent type before the body is called and back again afterwards. As mentioned above, view conversions are one of the places where invariants are checked.

However, if we add new operations then the old invariant does not apply to them. In truth, the specific invariant is not really inherited at all; it just comes along for free with the inherited operations that are not overridden. So if we do add new operations then we need to state the total invariant required.

Note that this is not quite the same model as specific postconditions. We cannot add postconditions to an inherited operation but have to override it and then any specific postconditions on the parent are lost. In any event, in both cases, if we want to use inheritance then we should really use tagged types and class wide aspects.

So there is also an aspect Type_Invariant'Class for use with private tagged types. The distinction between Type_Invariant and Type_Invariant 'Class has similarities to that between Post and Post'Class.

The specific aspect Type_Invariant can be applied to any type but Type_Invariant'Class can only be applied to tagged types. A tagged type can have both an aspect Type_Invariant and Type_Invariant 'Class.

Type_Invariant cannot be applied to an abstract type.

Type_Invariant'Class is inherited by all derived types; it can also be applied to an abstract type.

Note the subtle difference between Type_Invariant and Type_Invariant'Class. Type_Invariant'Class is inherited for all operations of the type but as noted above Type_Invariant is only incidentally inherited by the operations that are inherited.

An interesting rule is that Type_Invariant'Class cannot be applied to a full type declaration which completes a private type such as Disc_Pt in the example above. This is because the writer of an extension will need to see the applicable invariants and this would not be possible if they were in the private part.

So if we have a type T with a class wide invariant thus

```
type T is tagged private
  with Type_Invariant'Class => F(T);
procedure Op1(X: in out T);
procedure Op2(X: in out T);
```

and then write

```
type NT is new T with private
  with Type_Invariant'Class => FN(NT);
  overriding
  procedure Op2(X: in out NT);
  not overriding
  procedure Op3(X: in out NT);
```

then both invariants F and FN will apply to NT.

Note that the procedure Op1 is inherited unchanged by NT, procedure Op2 is overridden for NT and procedure Op3 is added.

Now consider various calls. The calls of Op1 will involve view conversions as mentioned earlier and these will apply the checks for FN and the inherited body will apply the checks for F. The body of Op2 will directly include checks for F and FN as will the body of Op3. So the invariant F is properly inherited and all is well.

Remember that if the invariants were specific and not class wide then although Op1 will have checks for F and FN, Op2 and Op3 will only check FN.

In the case of the type Disc_Pt we might decide to derive a type which requires that all values are not only inside the unit circle but outside an inner circle – in other words in an annulus or ring. We use the class wide invariants so that the parent package is

```
package Places is
  type Disc_Pt is tagged private
    with Type_Invariant'Class => Check_In(Disc_Pt);

  function Check_In(D: Disc_Pt) return Boolean;
  ...              -- various operations on disc points
private
  type Disc_Pt is tagged
    record
      X, Y: Float range –1.0 .. +1.0;
    end record;

  function Check_In(D: Disc_Pt) return Boolean is
    (D.X**2 + D.Y**2 <= 1.0)
      with Inline;

end Places;
```

And then we might write

```
package Places.Inner is
  type Ring_Pt is new Disc_Pt with null record
    with Type_Invariant'Class => Check_Out(Ring_Pt);

  function Check_Out(R: Ring_Pt) return Boolean;

private

  function Check_Out(R: Ring_Pt) return Boolean is
    (R.X**2 + R.Y**2 >= 0.25)
      with Inline;

end Places.Inner;
```

And now the type Ring_Pt has both its own type invariant but also that inherited from Disc_Pt thereby ensuring that points are within the ring or annulus. It is unfortunate that we could not make the size of the inner circle a discriminant but a discriminant cannot be of a real type. Ah well, perhaps in Ada 2019??

Finally, it is worth emphasizing that it is good advice not to use inheritance with specific invariants but they are invaluable for checking internal and private properties of types.

5 Subtype predicates

The final major facility to be discussed here is subtype predicates. These are not really contractual in the sense that preconditions, postconditions and invariants are contractual but are more akin to constraints.

Subtype predicates are of two kinds, Static_Predicate and Dynamic_Predicate. They can be applied to subtype declarations and to type declarations using aspect specifications. For example, in the Introduction we met

```
subtype Even is Integer
  with Dynamic_Predicate => Even mod 2 = 0;

subtype Winter is Month
  with Static_Predicate => Winter in Dec | Jan | Feb;
```

The predicates take an expression of a Boolean type and again we note the use of the subtype name to denote the current instance. In the case of Dynamic_Predicate, the expression can be any Boolean expression.

However, in the case of Static_Predicate, the expression is restricted and can only be

- a static membership test where the choice is selected by the current instance,
- a static case expression selected by the current instance,

- a call of the predefined operations =, /=, <, <=, >, >= where one operand is the current instance,
- an ordinary static expression,

and, in addition, a call of a Boolean logical operator **and**, **or**, **xor**, **not** whose operands are such static predicate expressions, and, a static predicate expression in parentheses.

So we see that the predicate in the subtype Even cannot be a static predicate because the operator **mod** is not permitted with the current instance. But **mod** could be used in an inner static expression.

However, the predicate in the subtype Winter can be a static predicate because it takes the from of a membership test where the choice is selected by the current instance and whose individual items are static. Note that membership tests are considerably enhanced in Ada 2012; further details will be given in a later paper. Another useful example of this kind is

```
subtype Letter is Character
  with Static_Predicate => Letter in 'A' .. 'Z' | 'a' .. 'z';
```

Static case expressions are valuable because they provide the comfort of covering all values of the current instance. Suppose we have a type Animal

```
type Animal is (Bear, Cat, Dog, Horse, Wolf);
```

We could then declare a subtype of friendly animals

```
subtype Pet is Animal
  with Static_Predicate => Pet in Cat | Dog | Horse;
```

and perhaps

```
subtype Predator is Animal
  with Static_Predicate => not (Predator in Pet);
```

or equivalently

```
subtype Predator is Animal
  with Static_Predicate => Predator not in Pet;
```

Now suppose we add Rabbit to the type Animal. Assuming that we consider that rabbits are pets and not food, we should change Pet to correspond but we might forget with awkward results. Maybe we have a procedure Hunt which aims to eliminate predators

```
procedure Hunt(P: in out Predator);
```

and we will find that our poor rabbit is hunted rather than petted!

What we should have done is use a case expression controlled by the current instance thus

```
subtype Pet is Animal
  with Static_Predicate =>
    (case Pet is
          when Cat | Dog | Horse => True,
          when Bear | Wolf => False);
```

and now if we add Rabbit to Animal and forget to update Pet to correspond then the program will fail to compile.

Note that a similar form of if expression where the current instance has to be of a Boolean type would not be useful and so is excluded.

Subtype predicates, like pre- and postconditions and type invariants are similarly monitored by the pragma Assertion_Policy. If a predicate fails (that is, has value False) then Assertion_Error is raised.

Subtype predicates are checked in much the same sort of places as type invariants. Thus

- on a subtype conversion,
- on parameter passing (which covers expressions in general),
- on default initialization of an object.

Note an important difference from type invariants. If a type invariant is violated then the damage has been done. But subtype predicates are checked before any damage is done. This difference essentially arises because type invariants apply to private types and can become temporarily false inside the defining package as we saw with the procedure Flip applying to the type Disc_Pt.

If an object is declared without initialization and no default applies then any subtype predicate might be false in the same way that a subtype constraint might be violated.

Beware that subtype predicates like type invariants are not foolproof. Thus in the case of a record type they apply to the record as a whole but they are not checked if an individual component is modified.

Subtype predicates can be given for all types in principle. Thus we might have

```
type Date is
  record
    D: Integer range 1 .. 31;
    M: Month;
    Y: Integer;
  end record;
```

and then

```
subtype Winter_Date is Date
  with Dynamic_Predicate => Winter_Date.M in Winter;
```

Note how this uses the subtype Winter which was itself defined by a subtype predicate. However, Winter_Date has to have a Dynamic_Predicate because the selector is not simply the current instance but a component of it.

We can now declare and manipulate a Winter_Date

```
WD: Winter_Date := (25, Dec, 2011);
...
Do_Date(WD);
```

and the subtype predicate will be checked on the call of Do_Date. However, beware that if we write

```
WD.Month := Jun;            -- dodgy
```

then the subtype predicate is not checked because we are modifying an individual component and not the record as a whole.

Subtype predicates can be given with type declarations as well as with subtype declarations. Consider for example declaring a type whose only allowed values are the possible scores for an individual throw when playing darts. These are 1 to 20 and doubles and trebles plus 50 and 25 for an inner and outer bull's eye. We could write these all out explicitly

```
type Score is new Integer
  with Static_Predicate =>
     Score in 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12
            | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21
            | 22 | 24 | 25 | 26 | 27 | 28 | 30 | 32 | 33
            | 34 | 36 | 38 | 39 | 40 | 42 | 45 | 48 | 50
            | 51 | 54 | 57 | 60;
```

But that is rather boring and obscures the nature of the predicate. We can split it down by first defining individual subtypes for doubles and trebles as follows

```
subtype Single is Integer range 1 .. 20;
subtype Double is Integer
  with Static_Predicate =>
     Double in 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20;

subtype Treble is Integer
  with Static_Predicate =>
    Treble in 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30;
subtype Score is Integer
  with Static_Predicate =>
    Score in Single or Score in Double or
     Score in Treble or Score in 25 | 50;
```

Note that it would be neater to write

```
subtype Score is Integer
  with Static_Predicate =>
    Score in Single | Double | Treble | 25 | 50;
```

Observe that it does not matter that the individual predicates overlap. That is a score such as 12 is a Single, a Double and a Treble.

If we do not mind the predicates being dynamic then we can write

```
subtype Double is Integer
  with Dynamic_Predicate =>
    Double mod 2 = 0 and Double / 2 in Single;
```

and so on. Or we could even use a quantified expression

```
subtype Double is Integer
  with Dynamic_Predicate =>
    (for some K in Single => Double = 2*K);
```

or go all the way in one lump

```
type Dyn_Score is new Integer
  with Dynamic_Predicate =>
    (for some K in 1 .. 20 =>
          Score = K or Score = 2*K or Score = 3*K)
    or Score in 25 | 50;
```

There are some restrictions on the use of subtypes with predicates.

If a subtype has a static or dynamic predicate then it cannot be used as an array index subtype. This is to avoid arrays with holes. So we cannot write

```
type Winter_Hours is array (Winter) of Hours;    -- illegal
type Hits is array (Score range <>) of Integer;   -- illegal
```

Similarly, we cannot use a subtype with a predicate to declare the range of an array object or to select a slice. So if we have

```
type Month_Days is array (Month range <>) of Integer;
The_Days: Month_Days := (31, 28, 31, 30, ... );
```

then we cannot write

```
Winter_Days: Month_Days(Winter);    -- illegal array
The_Days(Winter) := (Jan | Dec => 31, Feb => 29);
                                    -- really nasty illegal slice
```

However, a subtype with a static predicate can be used in a for loop thus

```
for W in Winter loop ...
```

and in a named aggregate such as

```
(Winter => 10.0, others => 14.0);    -- OK
```

but a subtype with a dynamic predicate cannot be used in these ways. Actually the named aggregate restriction is slightly more complicated. If the original subtype is not static such as

```
subtype To_N is Integer range 1 .. N;
```

then if To_N has a static predicate it still cannot be used in a named aggregate.

These rules can also be illustrated by considering the dartboard. We might like to accumulate a count of the number of times each particular score has been achieved. So we might like to declare

```
type Hit_Count is array (Score) of Integer;   -- illegal
```

but sadly this would result in an array with holes and so is forbidden. However, we could declare an array from 1 to 60 and then initialize it with 0 for those components used for hits and –1 for the unused components thus

```
type Hit_Count is array (1 .. 60) of Integer :=
                    (Score => 0, others => –1);
```

and we can use Score to indicate the used components.

The Hit_Count array can then be updated by the value of each hit as expected

```
A_Hit: Score := ... ;      -- next dart
Hit_Count(A_Hit) := Hit_Count(A_Hit) + 1;
```

If we attempt to assign a value of type Integer which is not in the subtype Score to A_Hit then Assertion_Error is raised.

After the game, we can now loop through the subtype Score and print out the number of times each hit has been achieved and perhaps accumulate the total at the same time thus

```
for K in Score loop
   New_Line;  Put(Hit);  Put(Hit_Count(K));
   Total := Total + K * Hit_Count(K);
end loop;
```

The reason for the distinction between static and dynamic predicates is that the static form can be implemented as small sets with static operations on the small sets. Hence the loop

```
for K in Score loop ...
```

can be implemented simply as a sequence of 43 iterations. However, a loop such as

```
for X in Even loop ...
```

which might look innocuous requires iterating over the whole set of integers. Thus we insist on having to write

```
for X in Integer loop
   if X in Even then ...
```

which makes the situation quite clear.

Another restriction on the use of subtypes with predicates is that the attributes First, Last and Range cannot be applied. But Pred and Succ are permitted because they apply to the underlying type. As a consequence, if a generic body uses First, Last or Range on a formal type and the actual type has a subtype predicate then Program_Error is raised.

Subtype predicates can be applied to abstract types but not to incomplete types.

Subtype predicates are inherited as expected on derivation. Thus if we have

```
type T is ...
   with Static_Predicate => SP(T);
```

and then

```
type NT is new T
   with Dynamic_Predicate => DP(NT);
```

the result is that both predicates apply to NT rather as if we had written the predicate as SP(NT) **and** DP(NT). So if several apply they are anded together. If any one is dynamic then restrictions on the use of subtypes with a dynamic predicate apply.

There is no need for special predicates for class wide types in the way that we have both Type_Invariant and Type_Invariant'Class. So in the general case where a tagged type is derived from a parent and several progenitors

```
type T is new P and G1 and G2 with ...
```

where P is the parent and G1 and G2 are progenitors, the subtype predicate applicable to T is simply those for P, G1 and G2 all anded together.

6 Default initial values

It is often important that we can rely upon an object having a value within its subtype even before it is assigned to and this especially applies in the face of type invariants and subtype predicates. Consider a type Location whose type invariant In_Place requires the point to be within some place.

```
package Places is
   type Location is private
      with Type_Invariant => In_Place(Location);
   function In_Place(L: Location) return Boolean;
   procedure Do_It(X: in out Location; ... );

private
   type Location is
      record
         X, Y: Float range –1.0 .. +1.0;
      end record;

   ...
end Places;
```

If we just declare an object of type Location thus

```
Somewhere: Location;
```

then there is no guarantee that Somewhere is anywhere in particular. If the type invariant In_Place applies and a subprogram with an **in out** parameter such as Do_It is called

```
Do_It(Somewhere);
```

then it might be that some paths through Do_It do not assign a new value to X. Nevertheless, on return from Do_It, the type invariant In_Place will be checked on the parameter. If Somewhere by chance had an accidental initial value outside the space implied by In_Place then the call will fail. Now it might be that other parameters of the procedure indicate to the caller that Somewhere has not been updated in this case but unfortunately this information is unlikely to be available to the invariant.

One solution to this is to ensure that objects always have an initial value satisfying the requisite constraints, predicates or invariants. One might do this by assigning a safe initial value thus

```
Somewhere: Location := (0.0, 0.0);  -- illegal
```

but this is illegal because the type is private. We could of course export from the package Places a safe initial value so that we could write

```
Somewhere: Location := Places.Haven;
```

But this is often frowned upon because giving an explicit initial value can hide flow errors. It is thus best to ensure that the object automatically has a safe default value by writing perhaps

```
   type Location is
      record
         X, Y: Float range –1.0 .. +1.0 := 0.0;
      end record;
```

It is curious that Ada allows default initial values for components of records and provides them automatically for access types (**null**) but not for scalar types or for array types. This is remedied in Ada 2012 by the introduction of aspects Default_Value and Default_Component_Value for scalar types and arrays of scalar types respectively. The format is as expected

```
type My_Float is digits 20
  with Default_Value => 0.5;
```

```
type OK is new Boolean
  with Default_Value => True;
```

The usual rule regarding the omission of => True does not apply in the case of Default_Value for Boolean types for obvious reasons.

If possible, a special value indicating the status of the default should be supplied. This particularly applies to enumeration types. For example

```
type Switch is (On, Off, Unknown)
  with Default_Value => Unknown;
```

In the case of an array type this can be constrained or unconstrained and the default value will apply to all components.

```
type Vector is array (Integer range <>) of Integer
  with Default_Component_Value => 0;
```

Default initial values cannot be given to the predefined types but they can be given to types derived from them such as the Boolean type OK above.

In the case of a private type, any default has to be given on the full type declaration.

It is important to note that default initial values can only be given for types and not for subtypes. If a default initial value lies outside the range of a subtype then declaring an object of a subtype without its own specific initial value will raise Constraint_Error. So writing

```
subtype Known_Switch is Switch range On .. Off;
A_Switch: Known_Switch;
```

raises Constraint_Error because the default initial value Unknown is outside the range of the subtype Known_Switch.

If a record type is declared and some components are given initial values but others are not then explicitly given initial values take precedence over default values given by these aspects. Thus if we have

```
  type Location is
    record
      X: My_Float range –1.0 .. +1.0 := 0.0;
      Y: My_Float range  –1.0 .. +1.0;
    end record;
```

then the component X has default value 0.0 but component Y has default value 0.5, (since My_Float declared above has default value 0.5).

A final important point is that default initial values supplied by these aspects have to be static unlike default initial values for record components.

7 Storage occupancy checks

Finally, two new attributes are introduced to aid in the writing of preconditions. Sometimes it is necessary to check that two objects do not occupy the same storage in whole or in part. This can be done with two functional attributes X'Has_Same_Storage and X'Overlaps_Storage which apply to an object X of any type.

Their specifications are

```
function X'Has_Same_Storage(Arg: any_type)
                                        return Boolean;
```

```
function X'Overlaps_Storage(Arg: any_type)
                                        return Boolean;
```

As an example we might have a procedure Exchange and wish to ensure that the parameters do not overlap in any way. We can write

```
procedure Exchange(X, Y: in out T)
  with Pre => not X'Overlaps_Storage(Y);
```

Attributes are used rather than predefined functions since this enables the semantics to be written in a manner that permits X and Y to be of any type and moreover does not imply that X or Y are read.

The object X and the parameter Y could be components such as A(5) or indeed A(J) or even a slice A(1 .. N). Thus the actual addresses to be checked may not be statically determined but have to be determined at the point of call.

AI-191 shows the following curious example

```
procedure Count(A: in out Arrtype; B: in Arrtype)
  with Pre => not A'Overlaps_Storage(B)
is
  -- intended to count in A the number of value
  -- occurrences in B as part of a distribution sort
begin
  for I in B'Range loop
    A(B(I)) := A(B(I)) + 1;
  end loop;
end Count;
```

The author seems to have assumed that the array A has appropriate components and that they are initialized to zero. This also illustrates the use of an aspect specification in a subprogram body.

At the machine level Overlaps_Storage means that at least one bit is in common and Has_Same_Storage means that all bits are in common. Hence X'Has_Same_Storage(Y) implies X'Overlaps_Storage(Y).

In some applications involving the possibility of aliasing (messing with tree structures comes to mind) we do really want to check that two entities are not in the same place rather than just overlapping in which case it is more logical to use Has_Same_Storage.

References

[1] ISO/IEC JTC1/SC22/WG9 N498 (2009) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of Amendment 2 to ISO/IEC 8652*.

[2] *Rationale for the Design of the Ada Programming Language*, Ichbiah, Barnes, Firth and Woodger (1986), Honeywell and Alsys.

[3] J. G. P. Barnes (2006) *Programming in Ada 2005*, Addison-Wesley.

Implementation and Usage of the new Ada 2012 Execution Time Control Features

Kristoffer Nyborg Gregertsen, Amund Skavhaug

Department of Engineering Cybernetics, NTNU, Trondheim, Norway; email: {gregerts,amund}@itk.ntnu.no

Abstract

This paper describes an implementation of Ada 2012 execution time control supporting the new separate execution time clocks for interrupts that has a design with several benefits. The real-time and execution time features use the same clock and alarm abstraction reducing the amount of code needed for the implementation. The design also allows a single hardware timer to support these features, freeing other timer hardware for application use. Clock measurement is tick-less, removing the periodic clock overflow interrupts. While the implementation is for a GNAT bare-board run-time environment, the presented design principles should be applicable for other systems. Performance tests are done to find the additional overhead to context switches and interrupt handling caused by execution time control. In addition to execution time measurement for interrupts we also provide an interrupt timer, and extend the object-oriented real-time framework to facilitate execution-time control for interrupts. An example application using this feature is given.

Keywords: Ada 2012, execution time control, interrupt clocks, real-time, embedded, GNAT.

1 Introduction

Scheduling analysis of real-time systems rely on the worst-case execution time (WCET) of tasks being known. However, finding the WCET of an algorithm may be hard, for some cases it is not even possible to predict if an algorithm will ever halt [1]. Furthermore, pipelines, caches and other performance enhancing techniques used on contemporary computer architectures makes the WCET even harder to find [2]. This makes WCET analysis a costly and time consuming process. Also, the WCET will often be considerably longer than the average execution time as it includes the very unlikely event of many or all of the performance enhancing techniques failing. Therefore pessimistic scheduling is needed in order to provide an offline guarantee that all hard deadlines will be met, which again leads to poor processor utilization if there are not enough tasks with soft, or no, deadlines to use the remaining processor resources.

Execution time control is a simple, yet powerful tool that allows the total time a task has been executed on a processor to be measured, and a handler to be called when this execution time reaches a specified timeout value. Combined with a scheduling policy taking advantage of this feature, it allows online control of task execution time instead of relying solely on offline guarantees [3]. Execution time control also allows execution time servers such as the deferrable and sporadic server for soft sporadic tasks [4]. Furthermore, it facilitates tasks executing algorithms were there is an increasing reward with increased service (IRIS) [5]. In this case the algorithm is stopped when it has converged or its execution time budget is exhausted. If no acceptable result was computed in time a simpler algorithm may be executed.

Execution time control was standardized together with other new real-time features in Ada 2005 [6]. The standard did not state which execution time budget, if any, that is to be charged the execution time of interrupt handlers. All implementations known to the authors up to this point charged the running task this execution time [7, 8, 9, 10]. This causes inaccuracy to execution time measurement and was pointed out as an issue when the new Ada 2005 real-time features were evaluated [11]. The authors at NTNU have ported GNATforLEON [12], a bare-board run-time environment supporting the Ravenscar restricted tasking model, to the Atmel AVR32 UC3 microcontrollers series [13] and developed it further [14]. When Ada 2005 execution time control was implemented for this run-time environment, special execution time clocks for interrupts handling were added, one for each interrupt priority [15, 16]. This improved accuracy of execution time measurement for tasks and also allowed execution time control for interrupts. These features were presented by the authors at IRTAW 14 and suggested added to Ada 2012 [17]. At the same workshop the developers of MaRTE suggested measuring the execution time of all interrupt handling combined [18]. The workshop decided to suggest execution time measurement both for separate interrupt IDs and all interrupts combined to be added to Ada 2012 [19, 20]. These features are now included in the working draft for the Ada 2012 standard [21].

In this paper there is first a brief presentation of the Ada 2012 execution time control. Then follows an abstraction for clocks and alarms supporting both the real-time clock and timing events, and execution time clocks and timers for tasks and interrupts. It is shown how this design is implemented on the AVR32 UC3 microcontroller series, and performance test results are presented. After this, it is described how execution time control for interrupts is integrated into the object-oriented real-time framework, and an example application is given. Finally there is a discussion on the design and implementation, the implementation cost compared to the benefits of execution time control, the portability of the design, and the real-time framework extensions.

Listing 1: Interrupt execution time clocks

```
package Ada.Execution_Time is
   ...

   Interrupt_Clocks_Supported : constant Boolean
      := implementation−defined;

   Separate_Interrupt_Clocks_Supported : constant Boolean
      := implementation−defined;

   function Clock_For_Interrupts return CPU_Time;

   ...
end Ada.Execution_Time;

package Ada.Execution_Time.Interrupts is

    function Clock
      ( Interrupt  :  Ada.Interrupts. Interrupt_Id )
         return CPU_Time;

    function Supported
      ( Interrupt  :  Ada.Interrupts. Interrupt_Id )
         return Boolean;

end Ada.Execution_Time.Interrupts;
```

2 Ada 2012 real-time features

2.1 Execution time measurement and timers

The package Ada.Execution_Time defines the type CPU_Time representing elapsed execution time measurement and the function Clock to get the execution time of a task [21]. The execution time of a task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services on its behalf [21]. For Ada 2005 it was implementation defined which task, if any, that was charged the execution time used by interrupt handlers and run-time services on behalf of the system. Ada 2012 has the ability to account for either the total or separate execution time of interrupts handlers. Listing 1 shows the additions to the specification of Ada.Execution_Time and its new child package Interrupts to support this feature.

The constant Interrupt_Clocks_Supported indicates if the system supports measuring the total execution time of interrupt handlers by the use of the function Clock_For_Interrupts . The function will raise Program_Error when called if not supported. The constant Separate_Interrupt_Clocks_Supported indicates if the system supports measuring the execution time of interrupt handlers separately by the child package Interrupts . In this child package the function Clock returns the execution time for the handler of the given interrupt or raises Program_Error if separate execution time for interrupts is not supported. If Supported returns false for the given interrupt Clock is to return a CPU_Time equal to Time_Of (0).

2.1.1 Timers

The child package Ada.Execution_Time.Timers defines the tagged type Timer which is used for detecting execution time overruns for a single task. The type Timer_Handler identifies a protected procedure to be executed when the timer *expires*. Handlers are set to expire at a given execution time or after a given time interval using two overloading Set_Handler procedures, and may be cancelled using the procedure Cancel_Handler. The function Time_Remaining returns the time remaining until the timer expires. Implementations are allowed to limit the number of timers possible for a single task and raise Timer_Resource_Error if this limit is exceeded. In this work there is a limit of one timer for each task as this limitation is recommended for use with the Ravenscar profile [9]. The Ravenscar profile does however not allow timers, so by including these strict compliance with the profile is lost.

2.2 The real-time clock and timing events

The package Ada.Real_Time defines the types Time and Time_Span used for the monotonic real-time clock, and the function Clock to retrieve the value of this clock. The real time clock corresponds to the passing of physical time, either with the time of system initialization as epoch or another reference time frame.

2.2.1 Timing events

The child package Ada.Real_Time.Timing_Events defines the tagged type Timing_Event that allows protected procedures to be called at a specified time without the need for a task or delay statement. The type Timing_Event_Handler identifies a protected procedure to be executed when the timing event *occurs*. With the exception of the function Time_Of_Event returning the absolute time of the event instead of the time remaining, timing events are used in the same way as timers. Implementations are required to document the upper bound on the overhead of the handler being called. The Ravenscar profile only allows timing events declared at library level.

3 Implementation

3.1 Design

The functionality of the real-time clock (RTC) and execution time clocks (ETCs) are quite similar: both clocks support high accuracy measurement of the monotonic passing of time since an epoch, and both support calling a protected handler when a given timeout time is reached. The main difference is that the RTC is always active, while an ETC is active only when its corresponding task or interrupt is executed. The similarities allow a design where one implementation of clocks and alarms in the internal package System.BB.Time provides support for both execution time control and the real-time features. In addition alarms are used internally for real-time task delay.

The package System.BB.Time defines the type Time to represent the passing of time since the epoch as a 64-bit modular integer, and the type Time_Span as a 64-bit integer with range from -2^{63} to $2^{63} - 1$ to represent time differences. The package defines the limited private types Clock_Descriptor and Alarm_Descriptor to represents clocks and alarms respectively, and Clock_Id and Alarm_Id as access types for these. The private definitions of clocks and alarms are shown in Listing 2.

The package also defines public routines for clock and alarm operations used by the Ada 2012 execution time control and

Listing 2: Definition of clocks and alarms

```
type Clock_Descriptor is
   record
      Base_Time : Time;
      -- Base time of clock

      First_Alarm : Alarm_Id;
      -- Points to the first alarm of this clock

      Capacity : Natural;
      -- Remaining alarm capacity, no more alarms if zero

   end record;

type Alarm_Descriptor is
   record
      Timeout : Time;
      -- Timeout of alarm when set

      Clock : Clock_Id;
      -- Clock of this alarm

      Handler : Alarm_Handler;
      -- Handler to be called when the alarm expires

      Data : System.Address;
      -- Argument to be given when calling handler

      Next : Alarm_Id;
      -- Next alarm in queue when set, null otherwise

   end record;
```

real-time packages. These are also used by the internal package System.BB.Threads for thread wake-up. In addition there are procedures for changing the active execution time clock used by System.BB.Interrupts, System.BB.Protection and the context switch routine. The routines are described in more detail in the following.

3.2 Hardware timer

The 32-bit COUNT / COMPARE system registers of the Atmel AVR32 architecture are used as hardware timer in this work. The COUNT register is reset to zero at system start-up and is incremented by one every CPU clock cycle. The COMPARE interrupt is triggered when COUNT equals COMPARE, cleared when COMPARE is written, and disabled when COMPARE is zero, which is also the reset value of the register. For newer UC3 revisions the COUNT register is reset on COMPARE match, which is not desirable for our use. It is however possible to disable this behavior in the CPU configuration register.

Three hardware timer operations are provided in the package System.BB.CPU_Primitives and implemented using in-line assembler code. The function Get_Count returns a snap-shot value of COUNT. The procedure Adjust_Compare sets COMPARE according to the argument C while preventing that the interrupt is lost:

$$\text{COMPARE} \leftarrow max(C, \text{COUNT} + \epsilon)$$

Here ϵ is a small number of clock cycles, so that an interrupt will be pending immediately after leaving the procedure if C was less than COUNT. The procedure Reset_Count sets COUNT to zero and returns the previous COUNT value c_p in one atomic operation:

$$c_p \leftarrow \text{COUNT}_- \quad (1)$$

$$\text{COUNT}_+ \leftarrow 0 \quad (2)$$

This is done by two instructions, the first reading c_p from COUNT, the second writing the value 2 to COUNT as this is the number of clock cycles the two instructions take. The operation is done atomically as interrupts are disabled when executing kernel calls. No clock cycles are lost when resetting the COUNT register: the sum of c_p and COUNT equals the value COUNT would have had without reset. The COMPARE register is not altered by the reset procedure, and has to be updated with a call to Adjust_Compare if needed.

3.3 Clocks

The type Clock_Descriptor seen in Listing 2 represents clocks and has three data members: (1) The Base_Time that holds the part of the clocks elapsed time not present in the hardware timer. It is initialized to zero. (2) The First_Alarm pointing to the first set alarm of the clock. It is initialized to a sentinel alarm and is never **null** after this. (3) The Capacity gives the remaining number of alarms allowed for this clock. For the real-time clock it is initialized to Natural'Last which in practice means no limit on the number of alarms. For task clocks Capacity is initialized to one as is recommended for the Ravenscar profile [9]. We also allow one alarm for interrupt clocks for interrupts not of the highest interrupt priority.

The package body has Clock_Descriptors for the RTC, interrupt clocks, and the internal idle clock used when the system is executing the idle-loop. In order save memory there is a pool of interrupt clocks and a look-up table with Interrupt_ID as index, instead of having a Clock_Descriptor for every interrupt. This Ravenscar run-time environment is designed not to use dynamic memory in the kernel [12]. The pool size is set to allow at most ten interrupts, but this can be easily be changed in the package System.BB.Parameters. The Clock_Descriptor of threads is stored in the type Thread_Descriptor of the package System.BB.Threads.

3.3.1 Clock management

After initialization of the package there are precisely two active clocks: the RTC that is always active and the ETC that points either to the clock of the running thread, to the clock for the interrupt being handled or to the idle clock. The ETC is changed as a result of a context switch, interrupt handling, or system idling.

The low-level interrupt handler of the run-time environment calls Enter_Interrupt with the Interrupt_ID prior to calling the interrupt handler. This procedure pushes the current ETC on a stack and activates the interrupt clock found in the look-up table as the new ETC. After the interrupt handler is called a call to Leave_Interrupt pops and reactivated the old ETC. The interrupt handler may also be interrupted by a higher priority

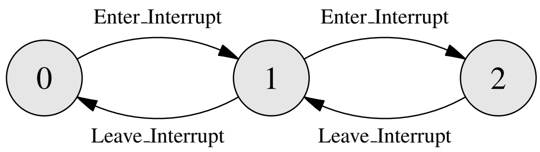


Figure 1: Stack states with two interrupt levels.

interrupt as seen in Figure 1. The stack size is limited by the systems number of interrupt levels.

There is no idle thread in the run-time environment. Instead the thread τ_a that finds the ready queue empty when leaving the kernel enters an idle-loop waiting for any thread to be made runnable by an interrupt. Prior to entering the idle loop a call to Enter_Idle activates the idle clock as the ETC. If τ_a is made runnable it calls Leave_Idle to reactivate its clock. Also a context switch may take place and change clock to the new running thread τ_b as seen in Figure 2. When τ_a resumes execution the idle clock will be activated by the context switch again instead of the task clock. In order to do this the Thread_Descriptor has a field Active_Clock that points either to the tasks own clock, or the idle clock if the tasks is executing the idle loop. Only one thread at a time will enter the idle loop.

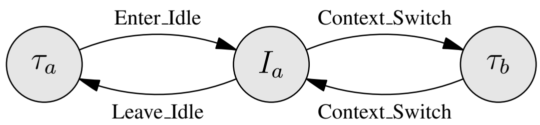


Figure 2: System idling with two tasks.

The states in Figure 2 are sub-states of state 0 in Figure 1, any of the states can be interrupted and will be restored when the interrupt handler is left. Since no task can have a base priority in the interrupt priority range in the Ravenscar profile context switches can only occur in state 0, after the task priority has be lowered back to the tasks base priority.

3.3.2 Measuring time

The use of the hardware timer is tick-less and therefore does not require a periodic clock overflow interrupt. Instead COUNT is reset using Reset_Count when the ETC is changed, and the base time of the RTC and the old ETC is incremented with the previous COUNT value c_p. By doing this the same hardware timer may be used for both the RTC and the ETC as seen in Figure 3.

The elapsed time of a clock t since the epoch is retrieved by the function Elapsed_Time, and is computed from the base time b and the COUNT register value:

$$t = \begin{cases} b + \text{COUNT} & \text{if clock is active} \\ b & \text{else} \end{cases}$$

An interrupt may occur after reading the base time but before reading COUNT in Elapsed_Time. This will update the base time and reset COUNT, making the sum of the earlier read base time and COUNT invalid. To avoid this there is a check

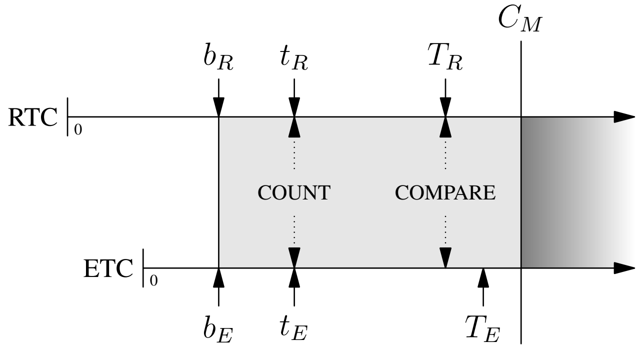


Figure 3: Relation between the RTC and ETC, and the hardware timer registers. The base time of the clocks are aligned.

after reading COUNT to see if the base time has been updated, in which case the updated base time will be returned as the elapsed time.

3.3.3 Setting the hardware timer

The COMPARE register is adjusted after updating ETC or after changing the first alarm of an active clock. If within the light gray region in Figure 3 the value C given to Adjust_Compare is the smallest difference d for the RTC and ETC between the first timeout T of the clock and its base time b:

$$C = min(min(d_R, d_E), C_M)$$

In rare cases b may be slightly larger than T. To handle this so that the COMPARE interrupt will be pending immediately after calling Adjust_Compare and prevent overflow d is computed as:

$$d = T - min(T, b)$$

Correct time measurement depends on COUNT never overflowing and C_M is a safety mechanism to prevent this critical error. By having $C_M = (2^{32} - 1) - C_S$ there will always be a pending COMPARE interrupt the last C_S clock cycles before overflow. This region is marked darker gray in Figure 3. If interrupts are not blocked by the system for longer than C_s the interrupt will be handled and COUNT reset when Enter_Interrupt is called, preventing overflow. The COMPARE interrupt handler will simply ignore this “false” interrupt. We use a large safety region $C_s = 2^{31}$ to provide ample time for the interrupt to be handled.

3.4 Alarms

The type Alarm_Descriptor seen in Listing 2 is used for representing internal alarms and has five data members: (1) Timeout that gives the time of event when set. (2) The Clock of the alarm given as argument to the alarm initialization procedure. If the Capacity for the clock is zero the initialization will not succeed and the alarm cannot be used. (3) Handler which is an access to the procedure that is called when the alarm expires and (4) the argument Data of type System.Address given when calling this handler. The handler and data are set during initialization of the alarm and remain constant after this. (5) The access Next pointing to the next alarm in the queue when the alarm is set, **null** otherwise.

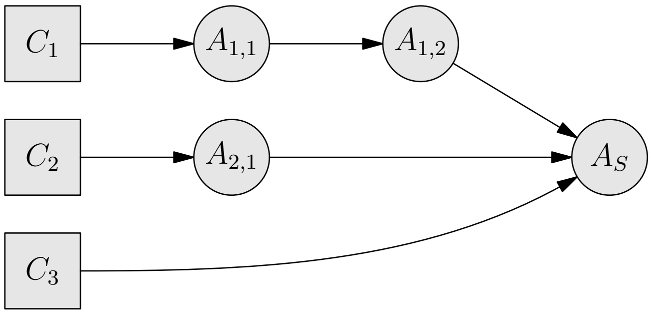


Figure 4: Three clocks set with two, one and zero alarms in addition to the sentinel at the end of the queue.

3.4.1 *The alarm queue*

The queue of pending alarms for clocks is managed as a linked list sorted in ascending order after the Timeout value of the alarms. Alarms with equal Timeout value are queued in FIFO order. To avoid the special condition of an empty queue there is a sentinel alarm with timeout at Time'Last that is always present at the end of the queue. The constant Time_Last seen by the user is set to Time'Last − 1 so that the sentinel is always last. This avoids an additional check when searching the queue. One sentinel alarm without handler is shared between all clocks as shown in Figure 4 to save memory.

The procedure Set takes the alarm and timeout as argument, sets the timeout field of the alarm, and searches the queue of the clock associated with the alarm for another alarm with timeout greater than the Timeout, the alarm is then inserted before this one and always before the sentinel. The procedure Cancel first checks that the alarm is set, and if so searches for the alarm in the queue and removes it. It is necessary to search the queue since to find the alarm before the one being removed as it is implemented as a single linked list. Both procedures reprogram the hardware timer if the alarm inserted or removed is first in the queue of an active clock.

3.4.2 *Calling alarm handlers*

The COMPARE interrupt handler has the highest interrupt priority. When this handler is called the procedure Alarm_Wrapper is called first for the RTC and then for the interrupted ETC on top of the stack. At this point the active ETC is that of the COMPARE interrupt itself, for which no alarms are allowed, so only the interrupt ETC on top of the stack or the RTC may be the cause of the interrupt. As the wrapper is called for both clocks there is no need to check which caused the interrupt. The alarm wrapper removes all alarms with timeout less or equal to the base time of the clock from the head of alarm queue one at the time, clears the alarm and calls the handler with the data as argument. The alarm handler can, and very often will, alter the alarm queue, so it is important to have the queue in a consistent state before calling the handler and reread the first alarm of the clock after calling the handler.

3.5 Ada 2012 interface

The implementation of the application programming interface as described by the Ada reference manual [6] is quite similar for the real-time and execution time control features as they use the same internal time, clock and alarm types.

Listing 3: Interrupt timer specification

```
package Ada.Execution_Time.Interrupts.Timers is

   type Interrupt_Timer (I : Ada.Interrupts.Interrupt_ID)
      is new Ada.Execution_Time.Timers.Timer
     (Ada.Task_Identification.Null_Task_Id'Access)
     with private;

private

   type Interrupt_Timer (I : Ada.Interrupts.Interrupt_ID)
      is new Ada.Execution_Time.Timers.Timer
     (Ada.Task_Identification.Null_Task_Id'Access)
     with null record;

end Ada.Execution_Time.Interrupts.Timers;
```

3.5.1 *Clocks*

The functions named Clock in the packages Ada.Real_Time, Ada.Execution_Time and Ada.Execution_Time.Interrupts all call Elapsed_Time with the Clock_Id of the RTC, a task clock or an interrupt clock as argument respectively. If there is no internal clock for a given interrupt CPU_Time_First is returned. To get the total execution time spent on interrupt handlers interrupts Clock_For_Interrupts iterates through all Interrupt_IDs and finds sum of calling Clock for each.

3.5.2 *Timing events and timers*

The tagged types Timing_Events and Timer both have an Alarm_Descriptor, an Alarm_Id that points to this after initialization and a user handler of type Event_Handler and Timer_Handler respectively. Both types use their alarm to call a wrapper with the object as argument, that again calls the user handler. The difference is in the initialization of the alarm where Timing_Events use the RTC, while Timer uses the execution time clock of the task. The alarm initialization may fail for Timer in which case the exception Timer_Resource_Error will be raised. For Timing_Event the initialization is asserted to succeed.

For both types the procedure Set_Handler first calls Cancel of System.BB.Time to remove the alarm from the queue if necessary before it sets the user handler and calls Set if this handler is not **null**. This has to be done as Set expects the alarm to be cleared. The procedure Cancel_Handler checks if the user handler is set in which case Cancel is called and Cancelled is set to true. Operations are done atomically by using the package System.BB.Protection for blocking interrupts.

3.5.3 *Interrupt timers*

To allow execution time control for interrupts the non-standard package Ada.Execution_Time.Interrupts.Timers shown in Listing 3 defines the tagged type Interrupt_Timer that inherits Timer and its operations. Note that the constant Null_Task_Id from Ada.Task_Identification has to be marked **aliased** to be used as discriminant when inheriting Timer. No body is needed for this package. The initialization procedure for timers checks if the object is of type Interrupt_Timer in which case it uses the interrupt clock instead of task clock. Interrupt timers are used in the exact same way as task timers.

Table 1: Performance test results in CPU cycles

Test	Implementation		
	TI-ETC	T-ETC	N-ETC
Context switch	602	602	471
Timing event	381	272	270
Interruption cost	296	503	–

4 Performance

Performance testing of the implementation is done with the Atmel AVR32 UC3A0512 microcontroller on the EVK1100 evaluation board. For the tests the microcontroller is run at 60 MHz, and is programmed and debugged using the Atmel JTAG ICE Mk II. Test data is sent over the serial line to the PC where it is retrieved and analyzed using GNU Octave.

The implementation with support for task and interrupt execution time control (TI-ETC) is tested against two other versions of the run-time environment: one where support for execution time control is completely removed (N-ETC), and one that supports execution time control for tasks only (T-ETC). Here N-ETC use the COUNT / COMPARE registers for the RTC in the same way as TI-ETC, with the exception of COUNT being reset in the COMPARE handler. This means that it has zero additional overhead to context switches and interrupt handling. For T-ETC the difference from TI-ETC is that the interrupt clocks and corresponding packages are removed, together with the calls to Enter_Interrupt and Leave_Interrupt in the low-level interrupt handler. This implementations should have zero additional overhead to interrupt handling compared to N-ETC, but the same additional overhead as TI-ETC for context switches.

4.1 Context switch overhead

The purpose of this test is to find the overhead to context switches by changing the execution time clock. We test without an alarm being set for the clock as the overhead is found to be the same regardless of alarm status. The test is done by having a task τ_a release a higher priority task τ_b that is blocked on an entry of a protected object. The release time is read by the protected procedure opening the entry, and is returned to τ_b by the entry. After being released τ_b reads the clock and the two time values are transferred over the USART line before the task blocks again and the test is repeated.

The first row in Table 1 shows the results for the implementations. The exact same number of clock cycles was measured in all samples for this test. This is due to simplicity of the executed test program and the deterministic nature of the UC3 microcontroller. The additional overhead caused by execution time control is inferred to be 131 clock cycles or 2.2 μs at the clock frequency used in the test.

4.2 Timing event overhead

The system is required to document the overhead of handling timing event occurrences. This is also a good measure of interrupt handling overhead in general caused by execution time control. The program has a single timing event that is programmed to occur with random intervals between 1 and 3 milliseconds. When the handler is called the difference between the timeout and the clock is recorded. After 100 samples the data is transferred over the USART line and the test is repeated.

The second row in Table 1 shows the results for the implementations in clock cycles. As before there was only one measured overhead value for each implementation due to the simplicity of the test program and the determinism of the UC3. It is inferred from the results that execution time control gives an additional overhead of 111 clock cycles to interrupt handling, or 1.85 μs at the clock frequency used for the test. The difference of two clock cycles between T-ETC and N-ETC is inferred to be caused by small differences in the function Elapsed_Time reading the real-time clock.

4.3 Cost to interrupted task

The execution time cost to the task being interrupted is greater than zero, as the interrupt clock is activated by the low-level interrupt handler, and not by hardware. The purpose of this test is to find this cost. The test is done by having a single task τ first setting a timer for its own execution time clock to expire in 20 ms if this timer is not already set, then reading its execution time clock, busy waiting 10 millisecond, and then reading this clock again. The clock values are transferred over the USART line and the test is repeated. Only the interrupt caused by the timer can occur between the two clock readings, and it can occur only once. A protected procedure with **null** as the only statement is used as handler. To find the cost we compare the difference in execution time when interrupted to when the task is not interrupted. This test is only relevant for TI-ETC and T-ETC.

The last row in Table 1 shows the cost to the interrupted task in clock cycles for the implementations with and without separate execution time clocks for interrupts. The execution time when not interrupted was always the same number of clock cycles for both implementations due to the deterministic nature of the UC3 microcontroller. When interrupted the execution time varied with one clock cycle. The worst-case cost of interruption is shown.

5 Use of interrupt timers

To ease development of real-time applications an object-oriented framework has been developed by several contributors in the Ada community [22]. The framework provides common real-time patterns such as periodic and sporadic tasks, detection of deadline miss and overrun detection, execution-time servers and more. By integrating the non-standard Interrupt_Timer into this framework it is also possible to control the execution-time spent on interrupt handling and thereby prevent deadlines being lost due to bursts of interrupts. The framework components related to interrupt handling can be separated into three parts: (1) the interface Interrupt_Controller used to control hardware interrupt generation; (2) the protected interface Interrupt_Server used to control the execution time spent handling a given Interrupt_ID

Listing 4: Definition of interrupt controller

```
package Interrupt_Controllers is

   type Interrupt_Controller is limited interface;

   procedure Enable
     (C : in out Interrupt_Controller ;
      I : Interrupt_ID) is abstract;

   procedure Disable
     (C : in out Interrupt_Controller ;
      I : Interrupt_ID) is abstract;

   function Supported
     (C : Interrupt_Controller ;
      I : Interrupt_ID) return Boolean is abstract;

   type Any_Interrupt_Controller is
     access all Interrupt_Controller 'Class;

   Unsupported_Interrupt : exception;

end Interrupt_Controllers;
```

in accordance with some policy; (3) the protected interrupt handlers, the framework provides the release mechanism Sporadic_Interrupt to release tasks as a result of an interrupt.

5.1 Interrupt controller

The interface Interrupt_Controller is defined as shown in Listing 4. The interface will typically be implemented by a peripheral driver. Depending on the peripheral it may control one or more interrupts. Use of the interface is very straight-forward: Enable enables the generation of given Interrupt_ID and Disable disables it. The function Supported indicates if the controller supports the interrupt, if other operations of a controller is called with an unsupported interrupt the Unsupported_Interrupt exception will be raised.

5.2 Interrupt servers

The interface Interrupt_Server shown in Listing 5 uses Interrupt_Controller to control the execution time spent handling a given interrupt according to a policy by enabling and disabling its generation. The tagged type Interrupt_Server_Parameters is used to pass the controller and the execution time budget to implementations of the interface.

The protected object Deferrable_Interrupt_Server shown in Listing 6 and 7 implements this interface following the deferrable server policy. This allows us to model the execution time spent handling the given interrupt as a periodic task with a given period and budget. The type Deferrable_Server_Parameters defines the additional parameters needed by the server, in this case the replenishing period of the execution time budget. Notice that the Interrupt_ID is given as a separate discriminant, this is needed to declare the timer statically in the protected object. Internally the deferrable server has a timing event used to call the procedure Replenish periodically with the period given as parameter. The procedure sets the execution time budget for the interrupt using the interrupt timer, and enables the interrupt if necessary. The first call to Replenish is at the system epoch, and will enable the generation of the

Listing 5: Interrupt server interface

```
package Interrupt_Servers is

   type Interrupt_Server_Parameters is tagged
      record
         Controller : Any_Interrupt_Controller;
         Budget : Time_Span;
      end record;

   type Interrupt_Server is protected interface;

   procedure Initialize
     (S : in out Interrupt_Server) is abstract;

   type Any_Interrupt_Server is access all Interrupt_Server;

end Interrupt_Servers;
```

Listing 6: Deferrable interrupt server specification

```
package Interrupt_Servers.Deferrable is

   type Deferrable_Server_Parameters
     is new Interrupt_Server_Parameters with
      record
         Period : Time_Span;
      end record;

   protected type Deferrable_Interrupt_Server
     (I       : Interrupt_ID;
      Param : access Deferrable_Server_Parameters) is
      new Interrupt_Server with

      procedure Initialize ;

      pragma Priority (Any_Priority'Last);

   private

      procedure Replenish (Event : in out Timing_Event);
      procedure Overrun (TM : in out Timer);

      Replenish_Event : Timing_Event;
      Execution_Timer : Interrupt_Timer (I);
      Next : Time;
      Disabled : Boolean := True;

   end Deferrable_Interrupt_Server;

end Interrupt_Servers.Deferrable;
```

interrupt. The procedure Overrun is called when the execution time budget is exceeded and disables the generation of the interrupt.

5.3 Example application

Our example application has a real-time task implemented by a tagged type inheriting Periodic_Task of the real-time framework. The task has period 10 ms and a 5 ms budget, and we use the periodic release mechanism with overrun and deadline miss detection. For each release the task simply busy waits 75% of its budget.

In addition the application receives data from the PC through the USART line. We use the same hardware setup as for the performance tests. The tagged type USART_Controller implements Interrupt_Controller and is used to setup, enable and

Listing 7: Deferrable interrupt server body

```
package body Interrupt_Servers.Deferrable is

   protected body Deferrable_Interrupt_Server is

      procedure Initialize is
      begin
         pragma Assert (Param.Controller.Supported (I));
         Next := Epoch;
         Replenish_Event.Set_Handler
           (Next, Replenish'Access);
      end Initialize ;

      procedure Replenish (Event : in out Timing_Event) is
      begin
         Execution_Timer.Set_Handler
           (Param.Budget, Overrun'Access);
         if Disabled then
            Disabled := False;
            Param.Controller.Enable (I);
         end if;
         Next := Next + Param.Period;
         Event.Set_Handler (Next, Replenish'Access);
      end Replenish;

      procedure Overrun (TM : in out Timer) is
      begin
         pragma Assert (not Disabled);
         Disabled := True;
         Param.Controller.Disable (I );
      end Overrun;

   end Deferrable_Interrupt_Server;

end Interrupt_Servers.Deferrable;
```

disable the RX interrupt of the USART. A protected object with the USART interrupt handler counts the number of characters received. The environment task outputs this count every second. This task has lower priority than the real-time task and no deadline.

The baud rate of the USART line is a far higher rate than the system is able to receive using interrupts. However, the *intended* usage is that characters are typed one-by-one to the serial line by the user, and therefore will be limited to a few characters per second. Since we do not fully trust this limitation to be respected, a deferrable interrupt server is included to control the execution time spent handling receive USART interrupt. We let the server have a replenishing period of 10 ms, the same period as the real-time task, and a budget of 1 ms. Hence, the total utilization not considering the background task, is 60% which is known to be safe using RMA. The parts of the application related to interrupt handling are shown in Listing 8.

Running on the UC3A0512 of the EVK1100 evaluation board, the application correctly counts each character sent by typing in the serial communication program “minicom”. In order to test the interrupt execution time control, we use the “cat” command to write the entire source code of the application to the serial device file, and observe that the USART interrupt is disabled when the budget is exceeded and re-enabled when it is replenished. During the test the real-time task did not miss any deadline. However, only 40% of the characters

Listing 8: Usage of interrupt server

```
package body Test is

   USART : aliased USART_Controller (USART_1_Address);

   Param : aliased constant Deferrable_Server_Parameters
     := (Controller => USART'Access,
         Budget     => Milliseconds (1),
         Period     => Milliseconds (10));

   USART_Server : Deferrable_Interrupt_Server
     (USART_1, Param'Access);

   protected RX_Counter is
      pragma Interrupt_Priority (USART_1_Priority);
      function Get_Count return Natural;
   private
      procedure Increment;
      pragma Attach_Handler (Increment, USART_1);
      Count : Natural := 0;
   end RX_Counter;

   protected body RX_Counter is
      function Get_Count return Natural is
      begin
         return Count;
      end Get_Count;
      procedure Increment is
      begin
         USART.Clear (USART_1);
         Count := Count + 1;
      end Increment;
   end RX_Counter;

   procedure Run is
      Next : Time := Epoch;
   begin
      loop
         delay until Next;
         Put (RX_Counter.Get_Count);
         New_Line;
         Next := Next + Seconds (1);
      end loop;
   end Run;

begin
   USART.Initialize;
   USART_Server.Initialize;
end Test;
```

sent were successfully received by the system. This loss could be prevented by using USART hardware flow control or buffering, but we want to keep the example application simple. As expected the real-time task misses all its deadlines during the burst when the interrupt server is removed from the system.

6 Discussion

6.1 Design and implementation

Our design supports both the real-time clock and timing events, and execution time clocks and timers using one internal clock and alarm implementation. This removes most of the near duplicate code compared to separate implementations. Table 2 shows code metrics for the implementations with full, task only and no execution time control as reported by the “gnatmetric” tool. Only packages that are different for

Table 2: Code metrics for implementations

Implementation	Decl.	Stat.	SLOC
TI-ETC	243	516	759
T-ETC	221	473	694
N-ETC	158	264	422

the implementations are included. As seen the difference between full and task only execution time control is small, only 65 logical code lines which includes two additional packages for interrupt clocks and timers. For System.BB.Time the difference is only 11 logical code lines. The difference between full and no execution time control is greater, 337 logical code lines, but this includes seven additional packages for execution time control. For System.BB.Time the difference is only 27 logical code lines. Overall the number of code lines added by execution time control seems small and acceptable compared to the features provided.

Another benefit of our design is that one hardware timer is sufficient to support both the RTC and the ETC. By using only one hardware timer and one clock interrupt, our system is easier to understand and debug as there are no race conditions between interrupts of different hardware timers that need to be handled. The reduced hardware requirements for the run-time environment also frees timers for the application. Compared to the earlier implementation of execution time control [16] that used one of the two Timer / Counter hardware timer units of the UC3A microcontroller, both these are available for the application with the new design and can be used for pulse-wave modulation (PWM), external signal generation and more.

The tick-less design means that there are no periodic clock interrupts to increment the most significant part (MSP) part of the time value. If context switches and interrupts occur more often than C_M which is 35.8 seconds on our system running at 60 MHz, there will be no interrupts caused by clock measurement. For typical real-time systems there will be more frequent context switches and interrupts than this. The execution time of the clock overflow handlers may not be negligible, meaning that it could affect scheduling analysis. While the tick-less design comes at the cost of additional overhead to context switches and interrupt handling, the benefits of removing the periodic clock tick is greater.

6.2 Portability

While our design is implemented on the AVR32 UC3 microcontroller series, it should be portable to any architecture where it is possible to implement the routines Get_Count, Adjust_Compare and Reset_Count according to their specification. With minor modifications it should also be possible to use 16-bit hardware timers instead of the 32-bit timer used in this paper. In this case it would be necessary to reduce the clock resolution as overflow interrupts would occur every 546 μs at the resolution of 60 MHz used in this paper.

Our implementation uses a hardware timer within the processor core, giving the benefit of a deterministic, constant access time. It is possible to use a peripheral hardware timer, although it may be harder to implement Reset_Count without clock cycle leakage as the access time for reading and writing timer registers over the peripheral bus would not be constant for most systems.

6.3 Overhead caused by switching clocks

The two overhead tests measure the time it takes either between two clock readings, or the time between an event taking place at a known time and reading the clock. It is known whether this time includes changing execution time clocks or not for the implementation being tested. When comparing results it is important to remember that there are minor changes in the compiler output that affect the result, and that the function reading the clock also has minor changes between the implementations with and without execution time control. However, the main difference in overhead is caused by changing clocks and the results are considered valid. The context switch and interrupt handling overhead was found to be 131 and 111 clock cycles respectively. The small difference of 20 clock cycles between the two results is due to differences in clock management.

The additional overhead to context switches and interrupt handling caused by the full implementation is significant. At the clock frequency of 60 MHz used in the tests this additional overhead is 2.2 μs and 1.85 μs respectively. This adds to the latency for interrupt handlers and task release, and reduces the overall system performance. Still, the overhead is not prohibitively high taking into account the benefits provided by execution time control. Also, this overhead includes the cost of the tick-less timer that removes the overhead to tasks and interrupts caused by the periodic clock interrupt.

6.4 Cost of interruption

The test measuring the execution time cost to a task being interrupted is more accurate than the overhead tests as we compare the difference when the interrupt did and did not happen for the same implementation. By design we know that at most one interrupt may occur between reading the clocks. The cost of interruption to the task when using interrupt clocks was 297 or 298 clock cycles. When using the clock of the interrupted task the cost was 502 or 503 clock cycles. The difference between the two implementations is thus 205 clock cycles, but without interrupt clocks the cost includes the whole execution time overhead of calling the timer handler including the Alarm_Wrapper and Execute_Handler procedures. The cost would be less if an ordinary interrupt handler was used.

The small but noticeable cost to the interrupted task when using interrupt clocks means that if a task is interrupted many times its budget may have to be extended to allow for this. Without interrupt clocks the cost of interruption is varying, depending on what is done in the interrupt handler. In the case of very simple handlers this cost may even be lower than when using interrupt clock due to the overhead of changing clocks. Still, having a constant cost regardless of what is done in the handler is better for analysis. It is also possible to transfer execution time from the task clock to the interrupt clock before and after calling the interrupt handler to refund the tasks cost without its clock going backwards observably. While this scheme would reduce the cost to interrupted tasks, it would increase the complexity and also would need to be tuned depending on compiler output, and was therefore discarded.

6.5 Hardware support

Ideally we would like to have near zero overhead to context switches and interrupt handling caused by execution time control, and near zero cost of interruption for tasks. This is not feasible without a specialized hardware timer that allows execution time clocks to be changed more efficiently. Therefore the authors have designed a Time Management Unit (TMU) supporting 64-bit timer values and atomic clock changes [23]. It is designed to have a simple memory mapped interface accessible though the peripheral bus, making it portable to different architectures. The TMU has been implemented with the AVR32 UC3 core as a part of a masters thesis at NTNU in cooperation with Atmel Norway [24]. Simulation results indicates that the overhead of switching clocks can be reduced to less than 50 clock cycles by using this hardware timer.

6.6 Interrupt timer

The interrupt timer is not a part of the Ada 2012 standard but should in the authors opinion be added to the next revision for the following reasons. First it provides execution time control for interrupts similar to that for tasks. If we measure the execution time for interrupts it should also be controllable by means such as the framework extensions described in this paper. This is important as the execution time spent handling interrupts may be very hard to predict as the interrupts may be generated by external hardware that are not controlled by the application. Alternatives to interrupt timers are to count the number of interrupts and disable the interrupt if the count gets to high, or to poll the execution time of the interrupt after the handler is called and disable the interrupt if the budget is exceeded. These solutions are less precise and also less efficient than using interrupt timers.

Also, the cost of including interrupt timers is small for our implementation as the same clock abstraction and hardware timer is used for task and interrupts. Since the interrupt timer inherits the operations from the task timer, no additional code is needed other than the definition of the tagged type and the code to initialize interrupt timers.

6.7 Framework extensions

The interrupt timer allows us to extend the object-oriented real-time framework to also provide execution time servers for interrupts following the same pattern as used for task execution time servers. While the task server controls the execution time for a group of tasks released sporadically, the interrupt server controls the execution time spent invoking one interrupt handler many times. The object-oriented nature of the framework allows us to create servers suitable for different needs. We have implemented the deferrable server under the assumption that it is acceptable to ignore interrupts for a while, but other schemes may for instance be to reconfigure the system into fail-safe mode in the case of interrupt overruns.

The deferrable interrupt server has a budget that is replenished periodically, and disables interrupt generation if this budget is exceeded. Since there is no way to cancel the interrupt being handled in Ada, the budget has to allow for an overrun of one additional handler invocation for the cases where the budget is exceeded right after entering the low-level handler. It should be considered adding a user handler that is called to notify the application when an interrupt is disabled, to allow for instance hardware diagnostics. This could of course also be done in the Disable procedure of the peripheral driver. In this case it could be useful to add a cause argument to this procedure.

6.8 Example application

The example application is typical in that we must assume one rate of interrupts, but cannot guarantee it as the generation the interrupt is not controlled by the application. Burst of interrupts may also be caused by permanent or transient hardware faults. The result is that the system has to handle more interrupts than budgeted for in the real-time analysis, if the effects of interrupt handling was analyzed at all. This could cause deadlines to be missed and thereby system failure. The presented extensions to the real-time framework provides an easy way to protect our real-time application against these situations.

In the example application we use the USART RX interrupt to receive data sent on the serial line. This is reasonable and efficient given that we know that the characters are sent by the user typing in a serial communication program. However the high baud rate means that the system could be overloaded with interrupts if this limitation is not respected. By using the deferrable interrupt server of the real-time framework we can easily set a budget for the interrupt so that our real-time task is guaranteed sufficient execution time to meet its deadline. No deadlines were lost due to burst of interrupts when the application was tested with the deferrable server, while several deadlines were lost during the burst when the server was not used. This gives a good indication that the deferrable interrupt server works as intended.

7 Conclusion

Our implementation of Ada 2012 execution time control has a design with several benefits. By using a single clock and alarm abstraction to support both the real-time and execution time clocks, we have reduced the amount of code needed for the implementation. This also allows just one hardware timer to support both these clocks, reducing the complexity of the system and the hardware requirements of the run-time environment. This frees valuable hardware timers for the application. We use the hardware timer in a tick-less manner, meaning that there are no periodical clock interrupts. By requiring only one hardware timer the design should also be easy to port to other architectures with similar timers.

Performance testing shows a noticeable overhead to context switch and interrupt handling caused by our implementation of execution time control. However, this is in our opinion justified by the value of the provided features, and the tick-less clock measurement. We also found that there is a low constant execution time cost to tasks being interrupted. While zero cost is the ideal, this constant cost is an improvement in analyzability compared to the varying, and in most cases higher, cost without separate execution time measurement for interrupts.

We have presented an interrupt timer providing execution time control for interrupts similar to that for tasks. This feature is not a part of the Ada 2012 standard where the execution time for interrupts can only be measured, and not controlled. By extending the object-oriented real-time framework using the interrupt timer we provide a deferrable execution time server for interrupts so that the time spent on interrupt handling may be analyzed as a periodic task. The example application shows that our framework extensions provide an easy and elegant solution to prevent deadlines being missed due to bursts of interrupts. In the authors opinion interrupt timers should be added to the next revision of the Ada programming language.

8 Further work

Work is in progress with an implementation using a specialized Time Management Unit (TMU) for execution time control instead of the COUNT / COMPARE timer, and test this implementation with the AVR32 UC3 core in cooperation with Atmel Norway.

References

[1] A. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, vol. 42, no. 2, 1937.

[2] R. Wilhelm *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *Trans. on Embedded Computing Sys.*, vol. 7, no. 3, pp. 1–53, 2008.

[3] A. Wellings and A. Burns, *Ada-Europe 2007*, ch. Real-Time Utilities for Ada 2005, pp. 1–14. Springer Berlin / Heidelberg, 2007.

[4] A. Burns and A. Wellings, "Programming execution-time servers in Ada 2005," in *Proc. 27th IEEE International Real-Time Systems Symposium RTSS '06*, pp. 47–56, Dec. 2006.

[5] C. M. Krishna and K. G. Shin, *Real-Time Systems*. McGraw-Hill International Edition, 1997.

[6] ISO/IEC, *Ada Reference Manual - ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1.*

[7] M. G. Harbour *et al.*, "Implementing and using execution time clocks in Ada hard real-time applications," in *Lecture Notes in Computer Science*, vol. Volume 1411/1998, pp. 90–101, Springer Berlin / Heidelberg, 1998.

[8] M. G. Harbour and M. A. Rivas, "Managing multiple execution-time timers from a single task," *Ada Lett.*, vol. XXIII, no. 4, pp. 28–31, 2003.

[9] J. A. de la Puente and J. Zamorano, "Execution-time clocks and Ravenscar kernels," *Ada Lett.*, vol. XXIII, no. 4, pp. 82–86, 2003.

[10] S. Urueña, J. Pulido, J. Redondo, and J. Zamorano, "Implementing the new Ada 2005 real-time features on a bare board kernel," *Ada Lett.*, vol. XXVII, no. 2, pp. 61–66, 2007.

[11] A. Wellings, "Implementation experience with Ada 2005," *Ada Lett.*, vol. XXVII, no 2, pp. 59–60, 2007. session report.

[12] J. F. Ruiz, "GNAT pro for on-board misson-critical space applications," *Ada-Europe*, 2005.

[13] Atmel Corporation, *AVR32UC3 - Technical Reference Manual*, March 2010.

[14] K. N. Gregertsen and A. Skavhaug, "An efficient and deterministic multi-tasking run-time environment for Ada and the Ravenscar profile on the Atmel AVR32 UC3 microcontroller," in *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, pp. 1572–1575, April 2009.

[15] K. N. Gregertsen, "Execution time management for AVR32 Ravenscar," Master's thesis, Norwegian University of Science and Technology (NTNU), 2008.

[16] K. N. Gregertsen and A. Skavhaug, "Implementing the new ada 2005 timing event and execution time control features on the avr32 architecture," *Journal of Systems Architecture*, vol. 56, pp. 509–522, 2010.

[17] K. N. Gregertsen and A. Skavhaug, "Execution-time control for interrupt handling," *Ada Lett.*, vol. 30, 2010.

[18] M. A. Rivas and M. G. Harbour, "Execution time monitoring and interrupt handlers: position statement," *Ada Lett.*, vol. 30, 2010.

[19] T. Vardanega, M. G. Harbour, and L. M. Pinho, "Session summary: language and distribution issues," *Ada Lett.*, vol. 30, 2010.

[20] S. Michell and J. Real, "Conclusions of the 14th international real-time ada workshop," *Ada Lett.*, vol. 30, 2010.

[21] ISO/IEC, *Ada Reference Manual - ISO/IEC 8652:201x(E) (Draft 13).*

[22] A. Burns and A. Wellings, *Concurrent and Real-Time Programming in Ada*. Cambridge, 2007.

[23] K. N. Gregertsen and A. Skavhaug, "Functional specification for a Time Management Unit." Presented at SAFECOMP 2010.

[24] S. J. Søvik, "Hardware implementation of a Time Management Unit," Master's thesis, NTNU, 2010.

15th International Real-Time Ada Workshop (IRTAW-15)

Mario Aldea Rivas
Universidad de Cantabria, 39005-Santander, SPAIN; email: aldeam@unican.es

Abstract

The 15th International Real-Time Ada Workshop was held in Fuente Dé, Spain from September 14th to the 16th, 2011. The main focus was on reviewing and evaluating the Ada 2012 support for real-time systems and on developing proposals for future language revisions. The workshop was very successful in achieving its goals and a number of issues were identified for future language revisions.

1 Introduction

The 15th International Real-Time Ada Workshop (IRTAW-15) was held in the impressive location of Fuente Dé (Cantabria, Spain), a nice mountain area by the "Picos de Europa" National Park.

The hotel was located close to the walls of the glacial cirque of Fuente Dé and just by the base of the cable car going up to the top of "Picos de Europa".

The local organization by Michael González was excellent, and there was plenty of time for discussions, informal conversation and also for enjoying the beautiful surroundings of Fuente Dé.

The Program Committee accepted twelve papers as a basis for discussion, which are being published as part of the official Proceedings of the Workshop [1]. There were nineteen participants, coming from Europe (Spain, UK, Italy, Portugal and France) and North America (USA and Canada).

As in previous IRTAW meetings, all the attendees took active part in the technical discussions which were at the core of the workshop. The main points of the discussions and the overall conclusions are summarized in the rest of this report.

2 Technical program

The technical program was organized into four technical sessions (Table 1). Each session had a chair person and a rapporteur, who was in charge of writing a report of the session including the agreements reached. The sessions were organized into slots of four hours including a half an hour coffee break.

The topic of multiprocessors was addressed by quite a number of position papers this year, so the whole first day of the IRTAW 15 workshop was allocated to discussing multiprocessor issues. The topic was divided into two sessions, the first one about general multiprocessor topics and the second one centered on resource control.

Table 1. Workshop Programme

Day	Session
Wednesday Morning	A.1: Multiprocessor issues (part 1)
Wednesday Afternoon	A.2: Multiprocessor issues (part 2, resource control protocols)
Thursday Full day	B: Language profiles and application frameworks
Friday Morning	C: Ada Concurrency

The other two sessions were centered on language profiles and application frameworks, and on concurrency issues.

Figure 1 Technical Session

2.1 Session A.1: Multiprocessor Issues, Part 1

The goals of this session [14] were to review and evaluate the Ada 2012 support for multiprocessors, and think about possible additions to future (post Ada 2012) language revisions.

Specific issues discussed in this session were:

- The current definition of dispatching domains
- Per dispatching domain scheduling policies
- Dynamic dispatching domains
- Support for very large number of cores
- Non-SMP architectures

It was a very interesting discussion backed by the experience acquired by a number of participants on implementing the Ada 2012 support for multiprocessors in their systems.

Current definition of dispatching domains

Two minor problems in the definition of the System Dispatching Domain were pointed out:

- It is defined as a constant although it is in fact modified by the creation of other dispatching domains.
- Depending on the processor range chosen for the other dispatching domains, the System Dispatching Domain could represent a discontinuous range of processors.

The workshop did not consider these two problems so serious as to require a change in the Reference Manual but recommended explaining then in the Ada 2012 Rationale.

Per dispatching domain scheduling policies

In the last IRTAW a proposal was made to allow assigning specific scheduling policies to each dispatching domain.

During the discussion it was pointed out that this behavior could be achieved by combining the dispatching domains with the priority specific dispatching. The "workaround" would consist in allocating to each dispatching domain tasks in a particular priority band with the desired dispatching policy.

Therefore, the conclusion was there is not a strong motivation for trying to push forward this feature.

Dynamic dispatching domains

In Ada 2012 dispatching domains are static, not allowing migration of CPUs from one dispatching domain to another.

There was an agrement on considering CPU migration as an important feature for "mode changes". Therefore, it was decided to encourage the submission of concrete proposals on this topic for the next workshop.

Support for very large number of cores

The workshop agreed it would be desirable to have some kind of "fine-grained" parallelism primitives to parallelize blocks, loops, etc.

The submission of proposals about this subject was strongly encouraged for the next workshop.

Non-SMP architectures

In non-SMP architectures, some banks of memory are "closer" than others to each particular CPU. An Ada application that wants to execute efficiently in this kind of architectures should:

- have information about the memory map.
- be able to specify the location of the storage pools in order to allocate objects where they can be accessed more efficiently.

Proposals on these topics were encouraged for the next workshop.

2.2 Session A.2: Multiprocessor issues (part 2, resource control protocols)

The main goals of this session [15] were:

- To review and evaluate the efficacy of the Ada 2012 support in the area of multiprocessor resource control.
- To look beyond Protected Objects and Rendezvous to other paradigms amenable to be used in multiprocessor platforms.
- To review previous workshop proposals of new synchronization primitives to improve parallel execution of Ada programs.

Ada 2012 support in multiprocessor resource control

The session started with a discussion about the meaning of priority inheritance in partitioned systems. The conclusion was that priority inheritance is still meaningful in multiprocessor systems (provided the assignment of priorities is globally coherent), because the scheduling policy for each priority band is shared by all the dispatching domains.

Afterwards, a review of the most common shared data protocols for multiprocessor systems was presented. The workshop considered important that users are given an interface to control and define different access protocols than simple spin-locks. This interface would allow Ada programmers to use the best protocol for each application.

Looking beyond Protected Objects: Software Transactional Memory

Transactional Memory (TM) was presented as an alternative to lock-based protocols that could scale better in architectures with a medium/large number of cores. An implementation of TM in Ada was presented.

The workshop concluded that work on TM (an in other paradigms for concurrency interaction with larger number of cores) is important. Further work on this topic is encouraged.

Mechanisms to improve parallelism

A proposal (first stated at IRTAW-13) to support a parallel broadcast of calls to an array of protected objects was revisited.

The complexity of such functionality was pointed out, since the parallel calls would require some execution context. Due to this complexity the workshop decided to dismiss this functionality.

Finally a discussion was carried out on the possibility of parallel releasing of tasks in functions within Protected Objects

The difficulty of how to pass the data to the different tasks was pointed out. The workshop concluded that this would be a good mechanism to have, but that a suitable approach needs further investigation.

2.3 Session B: Language profiles and application frameworks

The issues discussed in this session [16] were:

- Beyond Ravenscar: extensions and applicability.
- Real-time framework – dealing with multiprocessors and mode changes.

Language profiles beyond Ravenscar

The session started with the presentation of a proposal for a new profile. This profile would go beyond Ravenscar, including functionalities in order to gain the ability to tolerate timing faults.

A key functionality to detect budget time overruns are the Execution Time Timers, so this service should definitively be included in the proposed profile.

In order to perform error recovery actions we need to be able to suspend/resume individual tasks. There was a discussion between dynamic priorities and asynchronous task control as the alternatives to be included in the profile in order to achieve this goal.

The topic was closed with a general agrement on the utility and goals of the new profile and a clear intuition of the kind of services to be included in it. There was an invitation to the group to further investigate the topic, and then discuss the findings at IRTAW-16.

Ravenscar and distribution

There was a presentation of a Ravenscar-compliant Distributed Systems Annex implementation. The implementation is not SPARK-compliant due to the use of generics and abstract types.

The group sentiment in that respect was that "educated" generics and abstract types are useful abstractions for the project and they should be retained.

Code archetypes and programming frameworks

Two reports were presented: one about the development of Ravenscar code patterns for automated code generation, and the other about the extension to multiprocessor architectures of the real-time programming framework.

Some complementarity was identified between both approaches and the group encouraged both teams to investigate the possibility of integrating their results.

Ravenscar and EDF

A proposal was examined for an EDF version of the Ravenscar profile. In order to simplify the runtime support for this profile, it would not include the Baker's stack resource protocol, but instead it would use non-preemptive critical sections.

Some issues were raised on whether the EDF alone is sufficient for safely programming HRT systems or if, on the contrary, either fixed priority scheduling or budget control should also be included in the profile.

Further research on this topic was encouraged for the next workshop.

2.4 Session C: Ada Concurrency

The main issues discussed in this session [17] were:

- Concurrency and real time vulnerabilities
- Deferred attributes

General concurrency vulnerabilities

The ISO/IEC/JTC 1/SC 22/WG 23 Programming Language Vulnerabilities Working Group is starting to consider concurrency vulnerabilities.

The proposal for six concurrency vulnerabilities was presented to the participants in the workshop:

- Thread activation
- Thread termination – directed
- Thread termination – premature termination
- Shared data access
- Concurrent data corruption
- Concurrency protocol errors

The workshop did some minor comments on some of them and agreed all of them are programming language vulnerabilities that should be considered by WG 23.

Real-time vulnerabilities

Afterwards, there was an open discussion in order to identify concurrency real-time specific vulnerabilities. Two of them were identified to be added to the general concurrency vulnerabilities listed above.

The first vulnerability identified ("Real-Time Timing") is related to the drift between clocks in different processors or the drift between the different clocks used by an application.

The second vulnerability ("Real-Time Scheduling") deals with the issues such as priority inversion, missed interrupts or events and others, that can cause a task to miss its deadline or other undesirable scheduling effects.

Deferred attributes

The discussion about this topic was started in session 1.A and finished in this session.

A presentation was made on the existing limitations of the current model of setting attributes (priority, deadline and affinity) that can cause undesirable effects when trying to change several of them simultaneously for the same task.

There was some discussion about whether these changes could be performed atomically from inside a protected operation. The conclusion was that this is not a valid approach when changing other task's attributes.

The group sentiment was that a mechanism is required to allow deferred attribute setting for the next dispatching point of a task.

Two alternative implementations of the aforementioned mechanism were discussed: using an attributes object or using a set of procedures.

It was agreed that this issue needs further investigation, modelling and trial implementations.

3 Conclusions

The meeting was considered successful by the participants.

An intensive revision and evaluation of the Ada 2012 support for real-time systems was made, in particular in reference to multiprocessors issues.

The meeting has also identified an important number of issues that should be revisited in further workshops. No specific proposals for language changes have been raised since, at this moment, the Ada 2012 standard is almost closed and we are yet quite far away from the following language revision.

Social program

The lunch breaks gave to the participants the opportunity to enjoy the impressive surroundings of Fuente Dé. On Wednesday we took the cable car to the top of the mountains, 800 meters above the workshop location. As the cable car went above the clouds, we were witnesses of the astonishing landscape of “Picos de Europa”. After eating our packed lunch we had a relaxing walk before coming back to the technical work.

The reception and dinner was held in a restaurant in Potes (the main town in the area). Potes was celebrating its annual festival, so the group could enjoy the festive atmosphere in the town.

Figure 2 The group at the top station of the cable car

Next Workshop

The next meeting of the workshop is planned for the York area, UK in the spring of 2013.

References

[1] Proceedings of the 15th International Ada Real-Time Workshop (2011). To be published in Ada Letters.

[2] A. Burns and A.J. Wellings. Support for Multiprocessor Platforms. To be published in Ada Letters.

[3] A. Burns, A.J. Wellings and A.H. Malik. TTF-Ravenscar: A Profile to Support Reliable High-Integrity Multiprocessor Ada Application. To be published in Ada Letters.

[4] Alan Burns. An EDF Run-Time Profile based on Ravenscar. To be published in Ada Letters.

[5] S. Lin and A.J. Wellings and A. Burns. Ada 2012, Resource Sharing and Multiprocessors. To be published in Ada Letters.

[6] José F. Ruiz. Going real-time with Ada 2012 and GNAT. To be published in Ada Letters.

[7] Héctor Pérez Tijero, J. Javier Gutiérrez, and Michael González Harbour. Adapting the end-to-end flow model for distributed Ada to the Ravenscar profile. To be published in Ada Letters.

[8] Marco Panunzio and Tullio Vardanega. Charting the evolution of the Ada Ravenscar code archetypes. To be published in Ada Letters.

[9] António Barros and Luís Miguel Pinho. Revisiting Transactions in Ada. To be published in Ada Letters.

[10] Sergio Sáez, Alfons Crespo. Deferred Setting of Scheduling Attributes in Ada 2012. To be published in Ada Letters.

[11] Stephen Michell. Programming Language Vulnerabilities – Proposals to Include Concurrency Paradigms. To be published in Ada Letters.

[12] Sergio Sáez, Jorge Real, and Alfons Crespo. Adding Multiprocessor and Mode Change Support to the Ada Real-Time Framework. To be published in Ada Letters.

[13] Juan Zamorano, Angel Esquinas, Juan A. de la Puente. Ada Real-Time Services and Virtualization. To be published in Ada Letters.

[14] José F. Ruiz. Session Summary: Multiprocessor Issues, Part 1. To be published in Ada Letters.

[15] Luís Miguel Pinho. Session Summary: Multiprocessor Issues, part 2 (resource control protocols). To be published in Ada Letters.

[16] Tullio Vardanega. Session Summary: Language Profile and Application Frameworks. To be published in Ada Letters.

[17] Stephen Michell. Session Summary: Concurrency Issues. To be published in Ada Letters.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/category/developers-center/gems/.

Gem #101: SOAP/WSDL server part

Pascal Obry, EDF R&D

Date: 14 March 2011

Abstract: In this Gem we build a server providing Web services on the network.

Introdution

This is the first part of a two-part Gem on SOAP (Simple Object Access Protocol).

In this Gem we will be building a SOAP server and you'll see that with Ada it is quite simple!

Let's take a simple package spec such as the following:

```
package Temperatures is

   type Celsius is new Float;
   type Fahrenheit is new Float;

   function To_Fahrenheit (C : Celsius) return Fahrenheit;
   function To_Celsius    (F : Fahrenheit) return Celsius;

end Temperatures;
```

The body is not shown here but it's part of the source packages that can be downloaded [1].

The first step is to generate the WSDL (Web Service Description Language). A WSDL is an XML language for describing Web services. In the WSDL we find a description of the types and the specs of the routines. A WSDL is similar to an IDL but based on XML.

To generate the WSDL, AWS come with the ASIS-based **ada2wsdl** tool:

```
$ ada2wsdl temperatures.ads -a http://localhost:8888
   -o temperatures.wsdl
```

The options are:

-a http://...	Specifies the end-point for the Web services.
-o temperatures.wsdl	Outputs WSDL into temperatures.wsdl.

Out of this WSDL it's possible to generate stubs (for calling Web services) or skeletons (for implementing Web services). In this first part we're building a server, so we don't need the stubs. AWS comes with a second tool called **wsdl2aws** to generate all the necessary the code:

```
$ wsdl2aws -nostub -cb -spec temperatures
   -main soap_server temperatures.wsdl
```

The options are:

-spec temperatures	To use the routines as implemented in Temperatures unit.
-cb	Generates the SOAP callbacks using the routines found in the spec specified above.
-main soap_server	Generates a main named soap_server, this main program starts the SOAP server by referencing a SOAP dispatcher using the callback routines.

Using the three options above is very handy for building a server that provides Web services and nothing more. The last actions are just to compile the server and run it:

```
$ gnatmake -gnat05 -Psoap_server
$ ./server
```

At this point the services are available on the network and can be called by other programs, possibly built with other languages (Java and C# are the most common ones).

In the second part of this series we will see how to call those services from Ada using AWS.

[1] http://www.adacore.com/2011/03/14/gem-101-soapwsdl-server-part/

Gem #102: SOAP/WSDL client part

Pascal Obry, EDF R&D

Date: 28 March 2011

Abstract: In this Gem we will use web services as described in a WSDL document.

Let's get started…

This is the second part of a two-part Gem series on SOAP and WSDL.

In this Gem we will be using a Web Service as described in a WSDL document. These services could be implemented in Java, C#, or Ada, because the WSDL is universal in the Web Services world.

In the previous Gem we generated a WSDL from a simple Ada spec. Let's use it to generate the necessary code to use these Web services. We again use the **wsdl2aws** tool, but this time to generate only the stubs:

```
$ wsdl2aws -f -noskel temperatures.wsdl
```

A set of packages is generated. Two are of interest to us at the moment, namely:

- Package temperatures_service-types.ads, containing the types used by the Web services.
- Package temperatures_service-client.ads, containing the Web services client spec.

For each Web Service routine, two specs are generated:

```
function To_Fahrenheit
  (C        : Celsius_Type;
   Endpoint : String := Temperatures_Service.URL;
   Timeouts : AWS.Client.Timeouts_Values :=
                           Temperatures_Service.Timeouts)
   return To_Fahrenheit_Result;

function To_Fahrenheit
  (Connection : AWS.Client.HTTP_Connection;
   C          : Celsius_Type)
   return To_Fahrenheit_Result;

--  Raises SOAP.SOAP_Error if the operation fails
```

The first connects and closes the connection for each call, whereas the second uses a persistent connection. The usage is straightforward. Now, let's build a small program which converts Celsius to Fahrenheit:

```
with Ada.Text_IO;
with Temperatures_Service.Client;
with Temperatures_Service.Types;

procedure SOAP_Client is
   use Ada;
   use Temperatures_Service;
   C : constant Types.Celsius_Type := 20.0;
   F : constant Types.Fahrenheit_Type :=
                             Client.To_Fahrenheit (C);
   package C_IO is new Text_IO.Float_IO
                             (Types.Celsius_Type);
   package F_IO is new Text_IO.Float_IO
                             (Types.Fahrenheit_Type);

begin
   Text_IO.Put ("Celsius    ");
   C_IO.Put (C, Aft => 1, Exp => 0);
   Text_IO.New_Line;
   Text_IO.Put ("Fahrenheit ");
   F_IO.Put (F, Aft => 1, Exp => 0);
   Text_IO.New_Line;
end SOAP_Client;
```

We can use the following simple project file to build this program:

```
with "aws";
project SOAP_Client is
   for Source_Dirs use (".");
   for Main use ("soap_client.adb");
end SOAP_Client;
$ gnatmake -gnat05 -Psoap_client
```

Now let's test it, first by starting the server we have built last week:

```
$ ./soap_server
```

Then running soap_client:

```
$ ./soap_client
Celsius    20.0
Fahrenheit 68.0
```

That's all there is to it. As we've shown, it's easy to use a Web Service in Ada when the WSDL is provided. It's still possible to use a Web Service without a WSDL, but in that case it would be necessary to hand-code it.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o K.U. Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada in Sweden

Ada-Sweden
attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada Switzerland

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch