ADA USER JOURNAL

Volume 37
Number 1
March 2016

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. The language of the journal is English.

Although the title of the Journal refers to the Ada language, related topics, such as reliable software technologies, are welcome. More information on the scope of the Journal is available on its website at *www.ada-europe.org/auj*.

The Journal publishes the following types of material:

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

Further details on our approach to these are given below. More complete information is available in the website at *www.ada-europe.org/auj*.

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues

The *Ada User Journal* is open to consider the publication of proceedings of workshops or panels related to the Journal's aims and scope, as well as Special Issues on relevant topics.

Interested proponents are invited to contact the Editor-in-Chief.

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. Our readers need not surf the web or news groups to find out what is going on in the Ada world and in the neighbouring and/or competing communities. We will 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 electronically. Authors are invited to contact the Editor-in-Chief by electronic mail to determine the best format for submission. The language of the journal is English.

Our refereeing process aims to be rapid. Currently, accepted papers submitted electronically are typically published 3-6 months after submission. Items of topical interest will normally appear in the next edition. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

In this first issue of 2016, I would like to point out to the reader the recent update to the Ada 2012 language standard. The update (Technical Corrigendum 1 to ISO/IEC 8652:2012), has been approved and officially published by ISO, and includes a variety of clarifications and minor corrections coming from implementation and user experience. Therefore, the first contribution of this issue of the Ada User Journal is an article by John Barnes, explaining the rationale for the changes.

Afterwards, and approximately at the same time that the 18th International Real-Time Ada Workshop (IRTAW) meets in Benicàssim, Spain (11-14 April 2016), the journal publishes the overview of the previous workshop, which took place last year in Bennington, Vermont, USA (hosted by Robert Dewar in one of his recurrent contributions to the Ada community). IRTAW is a forum which explores approaches and solutions for Ada's support to concurrent and real-time systems, having contributed to Ada evolutions in areas such as tasking features, real-time and high-integrity annexes, and the Ravenscar Profile. We expect to be able soon to publish also the results of the 2016 workshop, which for sure will give indications on the future of the language in these areas.

In this issue of the Journal, the reader will also find the usual News, Calendar and Forthcoming Events sections. The latter in particular with the advance information of the 21st International Conference on Reliable Software Technologies – Ada-Europe 2016, next June in Pisa, Italy. The conference will provide a strong and diverse program, with three keynote talks, 12 refereed scientific papers and 8 industrial presentations, a rich set of tutorials, and the special Ada & Parallelism session. The program of the conference is complemented with presentations from projects and students from the ITS EASY Post Graduate School, co-located with the conference, and the 3rd workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering (De-CPS 2016). Reasons more than enough for Ada and reliable software practitioners to attend this year's Ada-Europe conference!

Finally, the reader will find a reprint of an article which was published in the last December issue of the Ada User Journal. Unfortunately, a printer problem caused some of the copies of that issue of the Journal to have a blank page instead of what should be page 260. We only detected this problem after the copies being mailed, for which we apologize. To mitigate the inconvenience, we are making the issue available already in the online archive, as well as reprinting the article in this issue. We are also taking the necessary steps with the printer to guarantee that the problem does not happen again.

Luís Miguel Pinho
Porto
March 2016
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation. Email: jacob@jacob-sparre.dk

Contents

Ada-related Organisations

Ada 2012 Language Standard Corrigendum Approved by ISO

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Tue, 23 Feb 2016 07:33:36 -0000
Subject: Ada 2012 Language Standard Corrigendum Approved by ISO
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FOR IMMEDIATE RELEASE

Ada 2012 Language Standard Corrigendum Approved by ISO

Milestone marks smooth continuation of Ada language standardization process

----

[See the full announcement in the Press Release section of this issue. —sparre]

Ada-related Events

[To give an idea about the many Ada-related events organised by local groups, some information is included here. If you are organising 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. —sparre]

Ada-Europe 2016 in Pisa

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Mon, 14 Dec 2015 01:27:25 -0000
Subject: 2nd CfP Ada-Europe 2016 Conference, Pisa, Italy
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

[The Call for Participation of Ada-Europe 2016 is included in the Forthcoming Events Section —sparre]

FOSDEM 2016

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Wed, 20 Jan 2016 22:09:06 -0000
Subject: FOSDEM 2016 - Ada Developer Room - Sat 30 Jan 2016 - Brussels
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

Ada-Belgium is pleased to announce the program for its

Ada Developer Room at FOSDEM 2016

on Saturday 30 January 2016

Université Libre de Bruxelles (ULB), Solbosch Campus, Room AW1.124

Avenue Franklin D. Roosevelt Laan 50, B-1050 Brussels, Belgium

Organised in cooperation with Ada-Europe

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/16/160130-fosdem.html

http://fosdem.org/2016/schedule/track/ada

FOSDEM, the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organised early each year in Brussels, Belgium. It is highly developer-oriented and brings together 5000+ participants from all over the world. The goal is to provide open source developers and communities a place to meet with other developers and projects, to be informed about the latest developments in the open source world, to attend interesting talks and presentations on various topics by open source project leaders and committers, and to promote the development and the benefits of open source solutions. The 2016 edition takes place on Saturday 30 and Sunday 31 January. It is free to attend and no registration is necessary.

In this edition, Ada-Belgium organises once more a series of presentations related to Ada and Free or Open Software in a s.c. Developer Room. The "Ada DevRoom" at FOSDEM 2016 is held on the first day of the event, Saturday 30 January 2016.

Ada is a general-purpose programming language originally designed for safety- and mission-critical software engineering. It is used extensively in air traffic control, rail transportation, aerospace, nuclear, financial services, medical devices, etc. It is also perfectly suited for open source development. The latest Ada standard was published by ISO in December 2012. As with the prior Ada 1995 and Ada 2005 standards, the first full implementation of the new Ada 2012 standard was made available in the GNU Compiler Collection (GNAT).

The Ada DevRoom aims to present the capabilities offered by the Ada language (such as object-oriented, multicore, or embedded programming) as well as some of the many exciting tools and projects using Ada.

Ada Developer Room Presentations (AW1.124, 59 seats)

The presentations in the Ada DevRoom start after the opening FOSDEM keynote. The program runs from 10:30 to 19:00, and consists of 7 hours with 11 talks/demos by 9 presenters from 5 different countries, plus 3 half-hour breaks with informal discussions.

10:30-11:00 - Arrival & Informal Discussions

Feel free to arrive early, to start the day with some informal discussions while the set-up of the DevRoom is finished.

11:00-11:05 - Welcome
by Dirk Craeynest - Ada-Belgium

Welcome to the Ada Developer Room at FOSDEM 2016, which is organised by Ada-Belgium in cooperation with Ada-Europe. Ada-Belgium and Ada-Europe are non-profit organisations set up to promote the use of the Ada programming language and related technology, and to disseminate knowledge and experience into academia, research and industry in Belgium and Europe, resp. Ada-Europe has member-organisations, such as Ada-Belgium, in various countries. More information on this DevRoom is available on the Ada-Belgium web-site (see URL above).

11:05-11:55 - An Intro to Ada for Beginning and Experienced Programmers
by Jean-Pierre Rosen - Adalog

An overview of the main features of the Ada language, with special emphasis on those features that make it especially attractive for free software development. Ada is a feature-rich language, but what really makes Ada stand-out is that the features are nicely integrated towards serving the goals of software engineering. If you prefer to spend your time on designing elegant solutions rather than on low-level debugging, if you think that software should not fail, if you like to build programs from readily available components that you can trust, you should really consider Ada!

12:00-12:50 - Make with Ada - Small Projects to Have Fun with Ada!
by Fabien Chouteau - AdaCore

In this talk I will present the first 4 projects of the "Make with Ada" blog post series: a solenoid engine, an Apollo lunar lander simulator, a software synthesizer framework, and a formally proven smartwatch app. I will also explain the motivation behind this series, where we want to go, and the feedback we've got from it.

13:00-13:25 - Adopting an Ada Program - the Experience of Whitaker's Words
by Martin Keegan - Unipart & Open Book Publishers

I present my experiences of adopting the maintenance of Whitaker's Words, a Latin dictionary written in Ada by Col William Whitaker, who was deeply involved in the creation of Ada itself. This will be the perspective of someone from outside the Ada community who found he really liked the language, and the challenges I faced learning Ada from online materials, converting the Words source code to more idiomatic (post Ada-83) forms, adopting the tooling, accessing community support, finding collaborators, making Ada play nicely with the Web, and so on. Whitaker's Words may be one of the most widely-used pieces of Ada software, and a quick Twitter search suggests it plays a key role in helping students cheat on their Latin translation homework. As a linguist and hacker, what really interests me is the use of Ada's type system to encode Latin's grammar.

13:30-13:55 - Creating a 3D Game Engine in Windows - Lessons Learned from Doom 3 BFG
by Justin Squirek

Ada Doom 3 is an open source project created as both an experiment and as a serious attempt at making a Windows game engine capable of fully rendering Doom 3 assets. Engineering a complete OS media layer and 3D engine that facilitates multiple platforms presents many unique challenges. These challenges and solutions will be discussed. I will also cover how Ada aided in the process of reverse engineering the half million line Doom 3 BFG (Id Tech 4 BFG) code base and how its typing system helped steer the development of Ada Doom 3 to its current state.

14:00-14:30 - Informal Discussions

A half-hour slot has been reserved for much needed interaction and informal discussion among Ada DevRoom participants and anyone potentially interested in Ada.

14:30-14:55 - Heterogeneous Parallel Computing with Ada Tasking
by Jan Verschelde - University of Illinois at Chicago

Consider the organisation of parallel heterogeneous computations. The sequential version runs in two stages: the first stage produces jobs that can be computed independently from each other in the second stage. The producer in the first stage is executed by one task, while the other tasks compute the jobs from the second stage, as the jobs are managed by a queue, implemented by a thread safe package. This design will be illustrated with an application that involves the refactoring of code in the free and open source package PHCpack, a package to solve polynomial systems by polynomial homotopy continuation.

15:00-15:50 - Micro- and Macro-Optimising a Distributed System
by Philippe Waroquiers - Eurocontrol

Or how to upload a 30000 flights simulation in 15 seconds. The Eurocontrol Flow Management System provides a simulation functionality to evaluate air traffic flow management measures (such as delay assignments or reroutings) before applying them operationally. This implies to upload a day worth of traffic in a simulation environment. This talk will describe various techniques and tools used to optimise the simulation startup time, and will discuss the gains reached via micro-optimisation (among others using Valgrind) or via macro-optimisation (such as using parallelism features of Ada).

16:00-16:25 - Controlling a Train Model w. GNAT GPL for Raspberry Pi 2
by Tristan Gingold - AdaCore

The GNAT GPL 2015 release by AdaCore includes a cross-compiler for a new platform: Raspberry Pi 2. We have used this platform to drive and control a real model train in Ada. SPARK was used to prove absence of collisions. I will present the hardware part as well as the software part, and show a video of the model train in action.

16:30-16:55 - CrazyFlie Drone Software in SPARK Ada
by Tristan Gingold - AdaCore

An AdaCore intern has rewritten the CrazyFlie drone software, originally in C, into SPARK. In addition to fixing some bugs, this allowed to prove absence of runtime errors. I will present the various techniques used to achieve that result, and plan to do a live demo of free fall detection.

17:00-17:50 - Memory Management with Ada 2012
by Jean-Pierre Rosen - Adalog

Dynamic memory management has always been a source of trouble, and garbage collection is just a way to overcome the lack of proper memory management in many languages. This presentation shows how Ada addresses this issue in several original ways: first by requiring much less dynamic memory than other languages, and then by providing powerful tools for controlling allocation and deallocation when it is necessary.

18:00-18:25 - A Command-Line Driver Generator
by Jacob Sparre Andersen - JSA Research & Innovation

A tool, which can take an Ada package specification, and generate a command-line driver for calling the procedures declared in the package. Which of the procedures is called is controlled by the names of the arguments passed to the driver program. The presentation will cover: how to use the tool; and some details of how the tool works - using the Ada Semantic Interface Standard (ASIS).

18:30-19:00 - Informal Discussions & Closing

Informal discussion on ideas and proposals for future events.

More information on Ada DevRoom

Speakers bios, pointers to relevant information, links to the FOSDEM site, etc., are available on the Ada-Belgium site at <http://www.cs.kuleuven.be/~dirk/ada-belgium/events/16/160130-fosdem.html>

We invite you to attend some or all of the presentations: they will be given in English. Everybody interested can attend FOSDEM 2016; no registration is necessary.

We hope to see many of you there!

Ada Semantic Interface Specification (ASIS)

Getting Started with ASIS

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Tue, 15 Dec 2015 08:42:07 +0100
Subject: Re: ASIS "housekeeping"
Newsgroups: comp.lang.ada

> I want to work with ASIS and can't figure how/where to start.

Ptree is a small utility that comes with AdaControl and helps you understand the ASIS view of an Ada program. Quite useful when you are starting with ASIS, and studying the source of Ptree is a good way to start, it is a quite simple use of ASIS.

If you want more sophisticated examples, you can study the code of AdaControl itself, but be aware that it is one of the most demanding applications for ASIS... Some rules are fairly simple, other need a deep understanding of the Ada language itself.

[...]

Ada-related Resources

Mascot Sightings

From: David Botton <david@botton.com>
Date: Tue, 9 Feb 2016 16:31:38 -0800
Subject: New Ada Mascot Sighting Added
Newsgroups: comp.lang.ada

I've been busy but the Ada Mascot keeps flying :) http://joinadanow.com/#mascot

Tons of people are showing off their mugs (no profits on products)

Latest sighting added from the Ada PDF Writer Demo. Contact me if your use of the Ada Mascot is not listed.

http://joinadanow.com/#sightings

[See also "Join Ada Now", AUJ 36-3, p. 121. —sparre]

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon Feb 29 2016
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn[1]: 2_374 members
- Reddit[2]: 818 readers
- Google+[3]: 619 members
- StackOverflow[4]: 327 followers
- Twitter[5]: 5 tweeters

[1] https://www.linkedin.com/groups?gid=114211

[2] http://www.reddit.com/r/ada/

[3] https://plus.google.com/communities/102688015980369378804

[4] http://stackoverflow.com/questions/tagged/ada

[5] https://twitter.com/search?f=realtime&q=%23AdaProgramming

[See also "Ada on Social Media", AUJ 36-4, p. 201. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon Feb 29 2016
Subject: Repositories of Open Source software

GitHub: 1_192 repositories [1]
301 developers [1]
791 issues [1]

Rosetta Code: 625 examples [2]
30 developers [3]
0 issues [4]

Sourceforge: 246 repositories [5]

BlackDuck OpenHUB: 214 projects [6]

Bitbucket: 74 repositories [7]

OpenDO Forge: 24 projects [8]
481 developers [8]

Codelabs: 14 repositories [9]

AdaForge: 8 repositories [10]

Assembla: 5 projects [11]

[1] https://github.com/search?q=language%3AAda&type=Repositories

[2] http://rosettacode.org/wiki/Category:Ada

[3] http://rosettacode.org/wiki/Category:Ada_User

[4] http://rosettacode.org/wiki/Category:Ada_examples_needing_attention

[5] http://sourceforge.net/directory/language%3Aada/

[6] https://www.openhub.net/tags?names=ada

[7] https://bitbucket.org/repo/all?name=ada&language=ada

[8] https://forge.open-do.org/

[9] http://git.codelabs.ch/

[10] http://forge.ada-ru.org/adaforge

[11] https://www.assembla.com/tag/ada

[See also "Repositories of Open Source Software", AUJ 36-4, p. 201. —sparre]

Ada-related Tools

GNAT Bareboard Drivers

From: Pat Rogers <rogers@adacore.com>
Date: Fri Oct 9 2015
URL: https://github.com/AdaCore/bareboard

1. Introduction

This repository contains Ada source code and complete sample GNAT projects for selected bare-board platforms supported by GNAT. Initially the repository contains software for ARM platforms from a specific vendor, but we intend this to be a location for both AdaCore and the community in general to contribute support for additional processors, platforms, and vendors.

2. License

All files are provided under a non-restrictive Berkeley Software Distribution (BSD) license. As such, and within the conditions required by the license, the files are available both for proprietary ("commercial") and non-proprietary use.

For details, see the "License" section in the release notes accompanying the HAL drivers provided by ST Microelectronics.

3. Requirements

The software is written in Ada 2012 and uses, for example, preconditions, postconditions, and the high-level iterator form of for-loops.

In addition, a GNAT implementation-defined pragma is used extensively. This pragma makes it possible to avoid explicit temporary copies when assigning components of types representing hardware registers requiring full word or full half-word accesses. The pragma is named Volatile_Full_Access. Those persons wishing to submit additions to the library should see the GNAT Reference Manual for details.

Therefore, building with the sources requires a compiler supporting both Ada 2012 and the GNAT-defined pragma Volatile_Full_Access. The "GNAT GPL 2015" compiler for ARM (ELF) is one such compiler. A recent GNAT Pro compiler for that target will also suffice.

4. Content

Initial provision is for the hardware in the STM32F4 family of 32-bit MCUs, as defined in the "RM0090 Reference Manual" (Doc ID 018909 Rev 6, Feb 2014) by STMicroelectronics and made available on the "STM32F4 Discovery" and the "STM32F429 Discovery" kit boards.

Specifically, there are low-level device drivers, higher-level component drivers, small demonstration programs for the drivers, and larger example applications. "Component" drivers are those that are implemented using the lower-level device drivers (e.g., SPI or GPIO), such as the gyroscope and accelerometer on the Discovery boards.

The small driver demonstration programs and the larger applications programs are provided as full projects, including GNAT project files, and are ready to build either within GPS or on the command-line.

Not all devices defined by the Reference Manual are supported, and not all those supported are complete. We encourage contributions of corrections, enhancements, and new drivers.

GnatDroid

From: John Marino <dragonlace.cla@marino.st>
Date: Mon, 30 Nov 2015 03:28:04 -0800
Subject: ANN: GnatDroid for Android-x86
Newsgroups: comp.lang.ada

The GNAT-to-Android/ARM cross-compiler known as GnatDroid-ARMv7 has been available on FreeBSD and

DragonFly since even before it was officially supported in GCC.

I've produced a second cross-compiler, GnatDroid-x86 which targets the 32-bit Android/x86 platform. The nice thing about Android/x86 is that it can be installed on a virtual machine like VirtualBox, so it can be easier to test programs without actual hardware that GnatDroid-ARMv7.

GnatDroid is only available on FreeBSD and DragonFly BSD, although somebody could replicate the effort for other platforms if so desired:

http://www.freshports.org/lang/gnatdroid-x86

NBAda

From: Olivier Henley <olivier.henley@gmail.com>
Date: Mon, 21 Dec 2015 11:13:34 -0800
Subject: NBAda (A library of lock-free data structures and algorithms for Ada) has never been mentioned here.
Newsgroups: comp.lang.ada

For info and reference:

http://www.gidenstam.org/Ada/Non-Blocking/

From: Anders Gidenstam <anders.gidenstam@gmail.com>
Date: Mon, 21 Dec 2015 23:54:30 -0800
Subject: Re: NBAda (A library of lock-free data structures and algorithms for Ada) has never been mentioned here.
Newsgroups: comp.lang.ada

> [...]

IIRC, it has been mentioned here before, but that was probably 10+ years ago. I used to be a regular reader (but maybe not a so frequent poster) of this news group back in the last years of the 1990s and the early 2000s.

I will consider adding the GM GPL exception, if there is interest for that. [...]

Also note that this is long stagnant code from the days of Ada 95, and it is also far down my TODO list for the rather small amount of spare time I have. Though, never say never..

From: Olivier Henley <olivier.henley@gmail.com>
Date: Mon, 21 Dec 2015 11:55:11 -0800
Subject: Re: NBAda (A library of lock-free data structures and algorithms for Ada) has never been mentioned here.
Newsgroups: comp.lang.ada

> [...] where are the sources?

http://www.gidenstam.org/Ada/Non-Blocking/src/NBAda-0.1.0-pre0.tar.gz

or

https://github.com/andgi/NBAda

> P.S. It is not GM GPL, only GPL, right?

COPYING file mentions GNU GENERAL PUBLIC LICENSE Version 2

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 21 Dec 2015 21:23:52 +0100
Subject: Re: NBAda (A library of lock-free data structures and algorithms for Ada) has never been mentioned here.
Newsgroups: comp.lang.ada

[...]

It looks that the implementation uses machine code insertions. Starting with gcc 4.7 (or something) that should not be necessary any more. Assuming the compiler is GNAT, of course.

[...]

PDF_Out

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 12 Jan 2016 09:59:30 -0800
Subject: Ann: Ada PDF Writer v.001
Newsgroups: comp.lang.ada

I am pleased to announce the first release of Ada PDF Writer. With the PDF_Out package you can write easily PDF files with text (there are Text_IO-like commands), vector graphics, images (JPEG). Headers and footers can be programmed. You can produce automatically reports, invoices, tickets, charts, maps etc. from your Ada program. You can also use PDF_Out as a simple device-independent medium for plotting a graph, without fighting with windows, buttons, GUI toolkits, etc.

The Ada PDF Writer library code is unconditionally portable (independent of compiler and of target machine's OS or CPU). No external toolkit or resource is needed.

Ada PDF Writer is free and open-source.

URL: http://apdf.sf.net/

Gnoga

From: Pascal Pignard <p.p11@orange.fr>
Date: Sun, 17 Jan 2016 11:10:45 +0100
Subject: Gnoga internationalization support with Zanyblue.
Newsgroups: comp.lang.ada

Gnoga now supports internationalization with the help of Zanyblue.

http://sourceforge.net/projects/zanyblue/

Also included JLokalize GUI tool managing language properties.

http://jlokalize.sourceforge.net

Both in deps directory.

Connect Four demo has been localized. Update Gnoga with GIT and do 'make connect_four'.

For now, in addition to original English, only French language is present. The language displayed is set based on browser localization, so French is displayed with a French running browser ;-)

Feel free to propose other localizations.

[See also "Gnoga", AUJ 36-2, p. 63. —sparre]

Traceback Wrapper

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 19 Jan 2016 08:52:02 -0800
Subject: Re: Re-write a file in Ada
Newsgroups: comp.lang.ada

> [...]

There is a way that doesn't need calling post-mortem addr2line: wrap your main procedure with the TB_Wrap generic procedure below, by instantiating the wrapper like this:

```
 with TB_Wrap, To_BMP;
 pragma Elaborate_All(TB_Wrap);
 procedure TB_To_BMP is
     new TB_Wrap(To_BMP);


 --------------------------------------------------------------
  --  File: TB_Wrap.ads
  --  Description: Trace-back wrapper for
  --  GNAT 3.13p+ (spec.)
--------------------------------------------------------------
generic
   with procedure My_Main_Procedure;
procedure TB_Wrap;

--------------------------------------------------------------
  --  File: TB_Wrap.adb
  --  Description: Trace-back wrapper for
  --  GNAT 3.13p+ (body)
--------------------------------------------------------------
with GNAT.Traceback.Symbolic;
with Ada.Exceptions, Ada.Text_IO;

procedure TB_Wrap is
    --  pragma Compiler_Options ("-g");
    --  pragma Binder_Options ("-E");
use Ada.Exceptions, Ada.Text_IO;
use GNAT.Traceback.Symbolic;
begin
    My_Main_Procedure;
exception
    when E : others =>
      New_Line (Standard_Error);
      Put_Line (Standard_Error,"-----------------
          [Unhandled exception ]---------------");
      Put_Line (Standard_Error, " >
           Name of exception . . . . .: " &
           Ada.Exceptions.Exception_Name
           (E) );
      Put_Line (Standard_Error, " >
           Message for exception . . .: " &
           Ada.Exceptions.
                  Exception_Message (E) );
      Put_Line (Standard_Error, " >
           Trace-back of call stack: " );
      Put_Line (Standard_Error,
           GNAT.Traceback.Symbolic.
                  Symbolic_Traceback (E) );
  end TB_Wrap;
```

Emacs Ada Mode

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 23 Jan 2016 18:11:11 -0600
Subject: 5.1.9 available in Gnu ELPA
Newsgroups: gmane.comp.lang.ada.emacs

ada-mode 5.1.9 is now available in Gnu ELPA

[See also “Emacs Ada Mode”, AUJ 36-2, p. 64. —sparre]

Generic Library for Algorithms and Containers

From: Emmanuel Briot <briot@adacore.com>
Date: Tue Jan 26 2016
URL: https://github.com/AdaCore/ada-traits-containers

Generic Ada Library for Algorithms and Containers

Goals

This library is another containers library for Ada. Although it provides containers that do not exist in the standard Ada runtime (graphs for instance), it is more interesting for the flexibility it proposes:

- Bounded/Unbounded containers and even more variants suitable for use with the SPARK language.
- Finite/Indefinite elements, and even more specialized variants optimized for specific types
- Pre and Post conditions, compatible with SPARK, so that some variants of the containers can be proven.
- Highly efficient; the user has full control over memory allocations, checks, locks, ...

All this flexibility is done via the intensive use of generic packages, themselves used to instantiate other generic packages.

Check the documentation for more details on the design of the API, and its current usage.

Compiling

The library itself is pure Ada code, and only requires a working Ada compiler to be available in your environment.

This library comes with a test suite which measures the performance of the various variants of the containers, and compares them with C++ equivalent (or near equivalents). This test suite generates a nice interactive HTML file.

Compiling and running the test suite requires that you also have a C++ compiler in your environment. In addition, you must install the Boost Graph Library (http://www.boost.org).

You must also download and install the GNAT Components Collection.

Once this is done, modify the shared.gpr file. Set the variable ```Boost_Include``` to point to the install prefix for Boost:

```
Boost_Include := "/usr/include";
```

Finally, compile and run the test with

```
make
```

and finally open the file index.html in a browser to view the performance comparison.

Package Registry Discussion

From: Olivier Henley <olivier.henley@gmail.com>
Date: Thu, 28 Jan 2016 17:13:19 -0800
Subject: Ada package registry?
Newsgroups: comp.lang.ada

Anyone thought about doing something similar to DUB, the D package registry (http://code.dlang.org/) ... for Ada?

(I know there are other projects for other languages but I also know for a fact that the D community did an exhaustive overview about these techs before committing to theirs)

In my opinion this is a silver bullet for the adoption/recruiting of new users to the language. It is also a silver bullet to speed up lib and application complexity/diversity. Just look at the amount/diversity of packages available at code.dlang.org. All this happened in about two years.

1. It centralises libs/applications existence knowledge. ('Ah there is a generic Markov chain library I was not aware of... why not undertake my next model-based multivariable control system using Markov chains in Ada. Cool.')
2. It centralises libs/applications efforts. (Huge impact when it comes to collaboration and reducing 'duplicate half-dead efforts' concerning specific tech: 'Ah there is a generic Markov chain library I was not aware of... why not help this person improve the lib as I am a Markov Chain specialist. Cool.')
3. You get improved code for free and automatically. Dependencies updates are retrieved on command.
4. It eases complex application build setup and maintenance.

For sure, at the moment, Ada initiatives are not gathered properly. For example, were you aware of the existence of a lean Ada library for generating UUIDs? No you did not and maybe this is why you did your own 'kind-ish' version at your job or on your free time to fulfil the requirements of another project.

An Ada package registry would have informed you of its existence. From there you could have decided to use this lib, improve this lib, propose a new one and/or decide to start that new application that was waiting for that specialised work of implementing UUIDs to be lifted for you. My two cents.

Comments? Thoughts?

How would you do it... or not/why not?

P.S. Before we get there... the fact that many Wikipedia articles are of poor quality does not affect its tremendous usefulness.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 29 Jan 2016 16:27:39 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> For sure, at the moment, Ada initiatives are not gathered properly.

Definitely false. I've been maintaining lists of sources of Ada packages and the Ada-wide search engine ever since we took over maintenance of AdaIC. For specific requirements, definitely try the Ada-wide search engine. (http://www.adaic.org/ada-resources/ada-on-the-web/). If there's an Ada-related site that's not included in that, it's either because the owner asked us not to include it, or we don't know about it at all. (In the latter case, send a message to news@adaic.com and it will get included in a future listing.)

(I don't recommend using the raw listing of libraries for searching for particular libraries, simply because that doesn't look inside of the various repositories [those are listed as single listing], the search engine does.)

> For example, were you aware of the existence of a lean Ada library for generating UUIDs? [...]

When I stuck "UUID" into the Ada-wide search engine, I got 6 (!) hits. Not sure that any of them are relevant, but if not that's because there is no such library that's ever been announced here or sent to AdaIC. In which case, it doesn't exist (practically). [The primary reason I read this newsgroup daily is to pick up announcements for AdaIC.]

P.S. I'm skeptical that any such repository would be kept very current. After the organizers initially populate it, I think it's not very likely that much updating would get done. After all, all library authors have to do to get on the AdaIC is send us a link -- I do all of the rest of the work. Yet hardly anyone does that. To have some sort of automated repository would require authors to do more: mirror their work somewhere they're not used to, or write a complex description for the repository, or more. (What are the odds that such a repository could figure out how to pull files from the version control on RRSoftware.Com and Ada-Auth.org, for instance? It could surely be done, but it would require some custom coding and I can't see how that would happen. In the absence of that, one is clearly going to have a subset of offerings...)

From: Olivier Henley
<olivier.henley@gmail.com>
Date: Sun, 31 Jan 2016 11:10:09 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> Definitely false. I've been maintaining lists of sources of Ada packages and the Ada-wide search engine ever since we took over maintenance of AdaIC.

First, thank you for the effort and reference. I was not aware of the relevance of AdaIC. But...

1. The fact that a new user is not "naturally" directed to AdaIC is a good sign the community is not properly organised... in comparison with other flourishing communities.

2. I do not mean to break the party but some links are dead or leads to dead ends e.g. code removed from page. Not to mention that some referenced website "are yelling" 1997 abandoned code/project. I know we should not and cannot judge code by its packaging but we also know how presentation is important. Unity3D may be crappy code inside, I dont know, but look how they present it to you : https://unity3d.com/. Millions of people are jumping on their bandwagon. So redirecting to too old and weird websites, sharing their code, is not a good idea when your goal is to promote the idea that Ada is "actually relevant".

> When I stuck "UUID" into the Ada-wide search engine, I got 6 (!) hits. Not sure that any of them are relevant ...

Well that is the main goal, to find relevant packages. I did the exercise too and AdaID did not come up so we are not in business ... yet. https://github.com/anthony-arnold/AdaID

> P.S. I'm skeptical that any such repository would be kept very current. [...]

Yes it would. I think you underestimate what is meant by package repository and/or package manager, at least like DUB:

1. There is a website, e.g code.dlang.org, wiki style with limited editing power; enough to add your package infos, e.g JSON file with all infos, authors, name, dependencies etc.

2. Every package code base has to reside on some "handled" version control system ecosystem like github, bitbucket etc.

3. On your machine you need an executable e.g dub.exe

Using this executable, dub.exe, you can fetch code dependencies, generate solutions and build libs and application with simple commands. Other commands let you control all kind of stuff like the particular code version of one of your lib dependencies.

a. Every time a lib/app owner commit new code to its repo (github, bitbucket etc) it is "automatically" maintained at code.dlang.org because this server only needs to know where to check for a particular package.

b. Dub.exe, on your machine, talks to this server and is capable of issuing git commands. Dub asks for some package, server gives infos and from there dub.exe drives git to fetch code, to a particular version, and then generate solutions and build if needs be.

c. Once in that loop, you can share your package that reference package that reference package that refer...

> To have some sort of automated repository would require authors to do more: mirror their work somewhere they're not used to,

To use github/bitbucket/etc is mandatory. I would not hire a guy that does not know or is not interested to learn git/mercurial and use github/bitbucket/etc. Not being aware of these as a programmer, in 2016, only demonstrate serious lack of curiosity and competencies deprecation.

> What are the odds that such a repository could figure out how to pull files from the version control on RRSoftware.Com and Ada-Auth.org, for instance?

It does not matter if not ALL sources are listed. It's a tool for the future, a community policy: new packages should be setup this way so we can build more effectively, together.

On Sunday, January 31, 2016 at 10:56:07 AM UTC-5, Mart van de Wege wrote:

> [...] In both cases a hard build dependency on a working gcc environment exists; it is the end-user's responsibility to make sure it does exist and that the proper header files for the dependencies are installed.

Yeah, its common practice to mention external dependencies in the README and/or by deduction. I would go even further, like special entries in the JSON that explicitly mentions the need for alien stuff. With DUB for example, you find a generic line like -> "libs": ["sqlite3"] inside the json config file... but we say/label "lib" for any piece of code that is not meant to be use as an end-user application, be it coded in the native language or another one. Maybe -> "alien-libs": ["sqlite3"] would be clear enough, at least to consult the doc for the meaning of alien-libs. :)

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 1 Feb 2016 18:44:47 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

>1. The fact that a new user is not "naturally" directed to AdaIC [...]

I'd agree with that, but there's nothing new about that. If you like herding cats...try an Ada community initiative. Not recommended for the faint of heart...

[...]

>Not to mention that some referenced website "are yelling" 1997 abandoned code/project.

So what? Good Ada code doesn't suddenly become useless because it's old. Good libraries don't NEED any maintenance, and Ada's been around since the early 1980s. Obviously, it depends on what you're looking for; if it is something relatively new or changeable, dead is not good. If it's an FTP library, it probably doesn't matter when it was created.

I'm not going to try to judge "goodness" of code, 'cause very little of it meets MY standards.

> [...]

People who are over-interested in "presentation" are not likely to be interested in Ada, IMHO.

> [...]

I think you overestimate the effort that most Ada programmers will put into such things...after all, we've had AdaHome, AdaPower, AdaIC, and probably others which such things, and they've all (except AdaIC) died from neglect.

>1. There is a website, [...]

Which requires the author to do something; many of them can't even be bothered to post about their libraries here (like AdaID) or send a link to AdaIC? Why do you think they would use some more complex website?

>2. Every package code base has to reside on some "handled" version control system ecosystem [...]

Most likely, only one or two. Much too hard to create something that works with everything, and like as not the volunteers will run out of energy long before.

>3. On your machine you need an executable e.g dub.exe

One would hope that this would be handled by Gnoga and not something that has to be installed. As you as you require a program, you're limited to Windows and Linux and maybe Mac, and someone has to fix that program every time there is an OS update.

> [...]

In order words, yet another tool to learn. One of the reasons I have so much trouble with Linux is that you have to figure out the various package managers before you can do anything -- but I usually just want to get whatever I need done and move on.

...

BTW, this is *exactly* what I thought you meant. I find it a combination of overkill and likely harm to the Ada community (by excluding large portions of, by the extra work involved at a minimum).

[...]

>It does not matter if not ALL sources are listed.

Actually, it does. A tool that includes 30% of the available source will make the listing on AdaIC look robust!

> Its a tool for the future, a community policy: new packages should be setup this way so we can build more effectively, together.

Don't buy it. Especially once whatever it does becomes obsolete (and change for the sake of change, the mantra of this century, will make that sooner rather than later).

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Tue, 2 Feb 2016 10:45:04 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...] many of them can't even be bothered to post about their libraries here (like AdaID) or send a link to AdaIC? Why do you think they would use some more complex website?

Well, to be fair "posting about their libraries here" *IS* requiring the authors to be doing something.

A good tool, IMO, would handle checking and submitting. Probably without much more difficulty than something like PowerArchiver for creating ZIP archives.

> [...]

If the "submission tool" ensures that anything submitted is compilable, we could process/store it in a DB. The hardest part would probably be going from the intermediate-representation to the DB, something like that could be done by a tool that reads the type definitions, creates the tables, and stores all the objects.

[...]

But we don't need to expand the scope to include non-Ada portions of the dependencies; this would mean we wouldn't have to worry about binaries and OSes.

Naturally, having some sort of way to link to binaries/installers would be nice, but it's not requisite.

[...]

All of those systems could, in theory, be tied into (or handled by) the "submission tool" -- where an author could click an "update and build" which would sync his environment with the VCS and build then, if successful, push the new version onto the Ada-specific repository.

[...]

Couldn't we handle these cases if we did it ourselves? I mean we don't need to have the package-manager itself handle building and installing, instead provide an API for those specific tools to do that, so we could limit what we're interested in to what all an Ada project needs.

[...]

> Don't buy it. [...]

I think I could buy it -- the biggest issue, I think, seems to be a method for specifying the project (things like alternate bodies, etc), if we could do that then the rest is essentially protocol details, right?

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Tue, 2 Feb 2016 12:46:49 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

True; my point was that while we can't really have a system agnostic tool that requires no human action, we can make things easy.

[...]

> Why not use an Ada package for building an Ada package registry ?...

This is the ides I would advocate (making it as much self-hosting as possible); however, I don't think that it's a good idea to tie ourselves to a file-system or build-tool (those being sufficiently outside the scope of package management, as well as introducing their own problems [e.g. case-sensitive FS vs case-insensitive]).

I think having an IR amenable to DB-storage would fit quite well here -- we could use the DB itself to help ensure consistency; we could also use it to provide some VCS functionality. (Thinking of programs as 'text' [and 'files'] is going to work against the level of sophistication we can put in our tools.)

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 2 Feb 2016 13:32:14 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

So perhaps you are thinking about a specialized archive manager (for instance a special version of AZip, to stay with Ada stuff :-)), which would grab packages or libraries from an Ada-centric repository, running on the programmer's computer ?

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Tue, 2 Feb 2016 20:21:49 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

Closer but not quite.

The way I envision it would be essentially three programs:

(1) the 'server', which contains the library, stored in an IR;

(2) the 'client', which pulls the info from the library and inserts it into the Ada environment,

(3) the 'project-manager', which takes the library/module on the hosting computer, compiles it, and submits it to the 'server'.

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Wed, 3 Feb 2016 10:39:04 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

(2a) the 'client', which also produces a library archive ready for pushing, one that follows a simple "standard".

The setup is easier when there is a small, agreed-upon set of requirements, I think, such as

- P, by default, can be built with

```
pragma Restrictions
    (No_Implementation_*)
```

- [optional] P is a leaf, it can be built with

```
pragma Restrictions (No_Dependence)
```

- there are test/example programs for P which can be built after P has been added to the normal Ada library

Note that the first item has "by default", which is an opportunity for vendors to put incentives into non-default editions (think: edition for Win32, edition for proprietary OS, ...).

If this scheme is simple, then, still, complexity may be seen as an opportunity by sales personnel, since the technical staff is payed for handling that complexity on behalf of the vendor's customers. No other vendor can handle the same instance of complexity. Which will be different when it is an instance of simplicity. Yet, the is a "by default" edition, mentioned earlier, for a freemium library, maybe.

At all costs, avoid the complexity of Linux style package management whenever there is a chance of it becoming Ubuntu style package management. ("How to force a dense graph of half-maintained packages, even when 'Recommended' dependences are turned off.") Unless, of course, you can sell Ubuntu, too.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 2 Feb 2016 16:51:20 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> Well, to be fair "posting about their libraries here" *IS* requiring the authors to be doing something.

Surely. But sending a link here or to AdaIC is about the minimum something I can imagine (anything less and you don't really want to make the library public in the first place -- which is irrelevant to this discussion).

> [...]

Harm from two things:

(1) Working only with a limited number of Ada implementations (most likely one);

(2) Only having a limited subset of code available;

Both of these could be fixed, but it would be very difficult to do. (A similar example was the job posting site that the AdaIC used to have. It was underpopulated enough so as to be more harmful than valuable, giving the impression of a lack of jobs. We finally got rid of it.)

[...]

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Tue, 2 Feb 2016 20:16:53 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

> (2) Only having a limited subset of code available;

>

> [...]

The second can be ameliorated by the aforementioned "packaging tool", I think, and the first needn't be a problem at all: we don't need to tie ourselves to a particular Ada-implementation or build-system and could easily consider those to be outside the scope of such a project.

[...]

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 1 Feb 2016 09:30:47 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> Anyone thought about doing something similar to DUB, the D package registry (http://code.dlang.org/) ... for Ada?

IMHO we should look to Haskell and how that community have solved this exact problem:

http://docs.haskellstack.org/en/stable/README.html

https://hackage.haskell.org/

It. Just. Works. Compiler and everything.

It is extremely well done.

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Mon, 1 Feb 2016 10:32:37 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

It does seem to work as a collector of solutions. [...] CPAN and PyPI are similar in this respect, and many will appreciate some evaluation procedure that tells them which of these results will just work for them.

Could you say something about portability across implementations of Haskell?

There is the Ada-wide search engine at http://www.adaic.org/ada-resources/ada-on-the-web/

What if this search engine knew about some Ada specific markup which authors of packages could add to the web pages describing packages so that these tags inform the search engine about Ada packages, specifically?

For reference, that are: Facebook style Open Graph markup for <head>, or "data-..." attributes of HTML5, or Dublin Core meta information, ...

From: Thomas Løcke <tl@ada-dk.org>
Date: Tue, 2 Feb 2016 09:54:16 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> Could you say something about portability across implementations of Haskell?

The stack tool is, to my knowledge, heavily focused on the GHC compiler right now, but that is to be expected as GHC is by far the most commonly used Haskell compiler - some would perhaps even argue that it is THE Haskell compiler.

But using stack does not in any way keep you from building your project with UHC, if you're so inclined. Stack doesn't enforce building with stack, it just enables you to do so.

What you get with stack is a dead simple way to make sure that packages _always_ build, even very complicated and large packages. It enables you to define compiler version/dependencies, and be comfortable in knowing that stack will handle those dependencies for you and other stack users.

It provides for a very lean and reliable approach for making and maintaining Haskell projects.

I've used it on Linux, Windows and OSX, and it just works.

> There is the Ada-wide search engine [...]

The problem is that this does not help building complicated libraries / projects. The fact that I can find project X does not necessarily make it straightforward for me to use X, and moving across operating systems this problem becomes pretty huge.

Stack and stack-like tools solve that problem.

I'm under no illusion that this is a simple task to mimic for Ada - I'm just saying that it works exceedingly well for Haskell.

From: Justin Squirek <jsquirek@gmail.com>
Date: Tue, 2 Feb 2016 06:51:48 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

I think the biggest problem facing the community is the actual portability of many of the most useful libraries and not on locating them. [...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 2 Feb 2016 17:06:13 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> The problem is that this does not help building complicated libraries / projects. The fact that I can find project X does not necessarily make it straightforward for me to use X, and moving across operating systems this problem becomes pretty huge.

Well, for the sorts of projects I'm interested in, finding them is 90% of the battle. Building an Ada library almost never requires more than dumping the source code into a directory and running the compiler's build tool (gnatmake, corder, or whatever). If it's more complicated than that, someone's overthought the whole thing and I most likely will forget that library. (After all, the reason I want an Ada library in the first place is so that I can fix it, include it in the source managed by Ada tools, and the like. If that's impractical, it's not helping.)

I realize that when libraries are bindings on third-party components, the situation gets more complex. But that also goes against my overall goal (if possible, write it or get it in Ada, and if not, try to go without).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 8 Feb 2016 18:05:14 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

[...]

I agree with you that Ada needs better publicity of the available tools and the like. The search engine would have gotten more mind-share if I could have come up with a snappy name for it -- but that's been beyond my capabilities. Since it doesn't have a snappy name, there's nothing to look for in Google. (And I'm not sure another search engine is the best place to look for our search engine anyway.)

Constructive suggestions are always welcome.

> [...]

My vision of Ada is of a language in which to construct code that is portable across implementations and across targets. To the extent that a tool, no matter how well-intentioned, interfers with that, then it is harmful to the future of Ada. GNAT /= Ada! What I heard being discussed only seemed to make sense within the context of GNAT on Linux, because that's the only place where "package managers" make sense. Perhaps I over-reacted a bit; I don't think there is much benefit to such a system, but it wouldn't hurt so long as it is not tied to a particular implementation or target. In particular, the setup Tero described would not bother me (I'd still be worried about it providing an image of relatively inactive community by not having much in it, but that might be a risk worth taking if enough of you care).

> [...]

Building complicated tools to do things that aren't really needed for Ada doesn't make it a "sensible proposition". If the "young people" need too much handholding, they're not really ready to engineer software with a professional tool like Ada. (How they get anything done in C++ is beyond me!) They'd be better off with Python or some such language.

[...]

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Mon, 8 Feb 2016 19:50:51 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> My vision of Ada is of a language in which to construct code that is portable across implementations and across targets. [...]

I don't think anyone here wants to interfere with that vision. Especially considering that it is generally agreed that the single free implementation (GNAT) isn't good for Ada.

> GNAT /= Ada!!

Agreed.

And most here don't think that it would be good if it were.

> What I heard being discussed only seemed to make sense within the context of GNAT on Linux, because that's the only place where "package managers" make sense.

I think that it depends on what the definition is. Given that early on in the thread the perl package manager (CPAN) was cited, it seems that the Linux idea of package manager was not intended -- the subject of the thread concurs -- I think what is being talked about could be described as an "online library".

If that is the case, I would recommend:

1) an internal representation, instead of text, which can be verified as consistant/compiled. [This may require a standard way to define an Ada project.]

2) a method (header-field?) marking the license(s) that the contained sources are under [to help w/ filtering, as well as making it feasible for non open source distribution].

3) a method for marking dependencies. (Also a header field?)

4) versioning. (To include changes impacting #2 & #3)

5) that building be handled separately. (Though a standard method for describing an Ada project could definitely help here.)

6) that any installation (of non-Ada items) be handled separately.

While something like integrated tests would be nice, I think they might be a bit beyond the scope of the project. (#5 and #6 are there to both reduce the complexity and the scope of such a repository.)

> In particular, the setup Tero described would not bother me (I'd still be worried about it providing an image of relatively inactive community by not having much in it, but that might be a risk worth taking if enough of you care).

I think any new system would have this problem. The initial populating would be the sticking point... but, arguably, the current situation is worse. (That being the exact impetus for suggesting such a repository.)

> [...]

Does your opinion change if what is being talked about is a library/dependency manager (as described above) rather than the Linux idea of "package manager"?

From: Thomas Løcke <tl@ada-dk.org>
Date: Tue, 9 Feb 2016 09:04:31 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

Package managers make sense on all operating systems I'm aware of. The fact that some don't have a package manager does not mean that they can't or shouldn't have one. I've installed and used the Haskell Stack tool on Windows, OS X and Linux, and it works equally well on all three.

I urge you to take a look at how the Haskell community have solved this particular "challenge". Stack / Cabal / Hackage are wonderful tools, created and maintained by a fairly small community.

http://docs.haskellstack.org/en/stable/README.html

I'm quite sure the Ada community could adapt those tools to match the needs of Ada in the 21st. century. I don't think we have to reinvent the wheel in order to gain the benefits of a proper package manager.

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Tue, 9 Feb 2016 14:33:55 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...] I urge you to take a look at how the Haskell community have solved this particular "challenge". [...] I don't think we have to reinvent the wheel in order to gain the benefits of a proper package manager.

I'm totally behind the idea. Alas, time is as scarce as ever.

Also, I think we're witnessing a new case of 'perfect is the enemy of the good'. Should really a submitted library compile in every OS ever, with every compiler ever, using the most complex dependencies there to be found? Just mark what works and what doesn't and let the Ada masses (:-)) improve on that.

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Tue, 9 Feb 2016 06:58:57 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> Should really a submitted library compile in every OS ever, with every compiler ever, using the most complex dependencies there to be found?

Given that Ada has always allowed the compiler to reject programs dependant upon implementation limits, I think that's a poor argument. However, it can be argued that compiling should not be part of the package manager's job. (OTOH, it makes sense to ensure that no invalid [non-compilable due to syntax or other detectable error] is accepted into the repository.

Dependencies also ought to be indicated/managed so as to allow a "recursive" retrieval -- but, again, this should be limited to only the Ada side of things. (E.G. installing Firebird for a firebird-binding can properly be the responsibility of the user.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 10 Feb 2016 19:46:48 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...] Should really a submitted library compile in every OS ever, with every compiler ever, [...]

No, but it should be crystal clear what assumptions it is making of compilers, target OSes, and hardware. That's where such a repository could be a significant improvement over the current situation. I don't want to start depending on stuff that only works in one situation, but it's pretty hard to tell that for most of the libraries that I've linked to. And even if you only want to work on Linux GNAT, you probably would want to avoid stuff that only works on Windows. And so on. Putting that into some sort of common format would be a worthwhile goal.

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Wed, 10 Feb 2016 21:19:58 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...] crystal clear what assumptions it is making [...]

You're absolutely right. This is why I think that a standard Ada-project format is in order: we would get a standard way to indicate system-dependencies,* multiple/variant bodies, and could use

THAT in the repository to indicate those parameters.

There is, of course the problem of having a library add/drop support for a system (e.g. Lumen & Windows), but that can be handled by the version-control aspect of the system.

* My Ada Project Manager takes the idea of Ada's generics as the basis for handling parameters, so depending on project-x transitively applies the parameter. Creating a standard "root" project for the repository-system would ensure that all the requisite parameters are extant.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 9 Feb 2016 11:08:36 -0700
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

"stack is a project of the Commercial Haskell group, spearheaded by FP Complete. ... While stack itself has been around since June of 2015, it is based on codebases used by FP Complete for its corporate customers and internally for years prior."

In other words, the initial version of stack was created by developers who were paid for their effort. While I have no problem with the idea, I doubt if any Ada company is going to be able to fund such a project. We could perhaps just modify stack for our purposes, but I doubt if we have enough Haskell expertise for that. Many of us are willing to contribute, but without funding or an existing system to adapt it seems unlikely to happen.

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Tue, 9 Feb 2016 13:00:06 -0800
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> Many of us are willing to contribute, but without funding or an existing system to adapt it seems unlikely to happen.

I don't know about that. I think that with the amount of willingness shown on the thread we're close to the 'crystalization point' -- I think the biggest things holding is back is a solid well thought plan.

I am somewhat against using an extant (and presumably general purpose) system, precisely because the real power/utility will come from properties a generalized system is ill-suited for. (Consider how diff is hampered in programmongs by being about text and not semantic differences.)

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Fri, 12 Feb 2016 17:05:47 +0100
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...]

What say the people interested in such a thing move the discussion to a place where things could start rolling? Just as a suggestion:

https://github.com/mosteo/alire/issues/1

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 1 Feb 2016 17:22:28 -0600
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> People using other programming languages have managed to create these package repositories, so it is a shame if Ada programmers cannot manage to do the same.

Not really. People using other programming language *implementations* have managed it. I see no evidence that it is possible to build such a thing that works with multiple implementations, running on multiple host and target systems, with different policies for installation, different project management facilities, and so on.

> You of course need to solve many problems related to this domain, but they are already solved by others, so one should be able to copy the design and just do the Ada implementation.

That wouldn't be an *Ada* repository. It would be a *GNAT* repository! There might be use to such a thing, but call a spade a spade here: don't insult those who use Ada, but don't exclusively use GNAT, by calling such a thing an *Ada* tool.

> [...]

Which was my previous point: in order for that to work, you have to insist on package authors in using a specific version control system, specific installation tools, and most likely a single Ada implementation. A lot of authors are not likely to change their methods of working to use such a system (or have the time to develop complex installation schemes), so you're going to end up with a (smallish) subtest of available libraries. Which would be actively harmful for the future of Ada (or GNAT, if you prefer) - it would appear that there is a lot less available than there really is.

I note that for "simple" libraries: those that use just Ada and/or target OS facilities, distribution simply by source code works well. For instance, we distributed Claw that way -- we provided a special main program that was designed to ensure that an Ada compiler's build tool would completely construct the library. Use gnatmake or corder or whatever did the trick. If one wanted to set up a separate project, one would use the tools of the implementation to do that (but it's not necessary). (If the user doesn't understand how to use the compiler's build tool, they need to understand that before doing anything in any case, they're not ready to build *anything* in Ada.)

For more complex systems that need other libraries, I doubt there is any sensible solution. Unless you're planning to abandon Ada's portability between implementations -- not of much value, IMHO.

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Fri, 5 Feb 2016 21:52:44 +0200
Subject: Re: Ada package registry?
Newsgroups: comp.lang.ada

> [...] I see no evidence that it is possible to build such a thing that works with multiple implementations, running on multiple host and target systems, [...]

NetBSD's pkgsrc should run on multiple platforms (all *BSDs, most Linux distributions, not sure about OS X or Windows) and support multiple programming languages/build systems. And if you want to support multiple host *and target systems* you could always check how OpenEmbedded/Yocto/bitbake combo is doing things.

[...]

> That wouldn't be an *Ada* repository. It would be a *GNAT* repository! There might be use to such a thing, but call a spade a spade here: don't insult those who use Ada, but don't exclusively use GNAT, by calling such a thing an *Ada* tool.

I think you are forgetting who you are replying right now. :)

Most of my Ada software compiles nicely with other Ada compilers (Janus/Ada, ICCAda) and I don't think it would be impossible to add Janus/Ada or ICCAda support to the package repository tool. You just need to make sure you don't try to compile Ada 2005/2012 code with Janus/Ada, which supports only Ada 95.

> [...]

The package repository/tool which I am thinking about, would be only for open source projects/libraries/applications. Open source developers usually use only open source version control systems (cvs, subversion, mercurial, git, darcs, monotone, and fossil) and there are existing practices how to interface with them. Also, old-fashioned zip/tgz release package (fetched over http/https) is easy to support.

What comes to installation methods, the $language community usually provides enough pressure and shaming to make the rogue project to follow the rules or to provide an installation method which is easy enough to integrate into the package management system.

> For more complex systems that need other libraries, I doubt there is any sensible solution. Unless you're planning to abandon Ada's portability between implementations -- not of much value, IMHO.

Not sure what sort of complexity you are thinking about. Just iterate through the dependency chain and build one dependency at time using the selected Ada compiler. When building one package, link the dependencies, and be happy.

If there are C language libraries involved, call gcc or Visual C, and continue as usual.

Building the package repository and related tool might take a lot of wall clock time, since the amount of volunteer Ada programmers with free time is limited, but I don't really see any blocking technical problems. And if there are any, I think then it is time to start making adjustments to the next Ada (or Ada compiler implementation) version!

PragmAda Reusable Components

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Sat, 30 Jan 2016 16:20:50 -0700
Subject: PragmARCs on GitHub
Newsgroups: comp.lang.ada

The PragmAda Reusable Components are now available through GitHub, as well as by downloading a .zip file from the PragmAda web site.

https://github.com/jrcarter/PragmARC.git

The default branch is Ada-07, which is the beta version that uses features of ISO/IEC 8652:2007. Branch master is the Ada-95 version.

[See also “PragmAda Reusable Components”, AUJ 35-3, p. 154. —sparre]

Miscellaneous Libraries

From: darkestkhan <darkestkhan@gmail.com>
Date: Wed, 3 Feb 2016 14:49:55 -0800
Subject: ANN: Libraries and bindings.
Newsgroups: comp.lang.ada

Since Randy brought it up that we should post here about our projects if we want more people to know about them I will do just that. This is not particularly long list but nonetheless some of you may find something from this quite useful. IIRC everything (or almost everything) is licensed under BSD/ISC license. All projects build with gprbuild just fine (didn't test building them

https://github.com/darkestkhan/vt100

"This is wrapper library for vt100 calls in Ada." [why code yet another? only print calls are not bound]

https://github.com/darkestkhan/imago

"Ada binding to DevIL library. This library tries to be as close to original API as possible, while using Ada types whenever possible" DevIL library supports loading/saving image files (and changing data formats) - most formats are supported [including some more obscure ones] (BMP, JPEG, ...).

https://github.com/darkestkhan/oto

Binding to OpenAL. Follows original API.

https://github.com/darkestkhan/xdg

"A small library that should make it at least slightly easier to follow XDG guidelines." [http://standards.freedesktop.org/basedir-spec/basedir-spec-0.7.html]

https://github.com/darkestkhan/cbap

"A small library providing simple callback-based parser for program arguments. This library is meant to be simple to use, and as such doesn't possess any particularly advanced features (like argument combining) that may result in complicating its use (or implementation)."

https://github.com/darkestkhan/lumen2

A fork of lumen - basically my changes and additions went quite far so I couldn't keep it any more at a branch. e.g. Windows support is cut-out due to licensing issues (it was under AGPL instead of ISC), more calls are bound to, etc. [Windows support (in fact binding to entire win api) is on my todo list - but it is impossible ATM for me due to lack of Windows machine]

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Fri, 5 Feb 2016 17:25:17 +0200
Subject: Re: ANN: Libraries and bindings.
Newsgroups: comp.lang.ada

> [...]

Here my (hobby) projects. Like darkestkhan's libraries, most of my projects are distributed under permissive open source ISC license:

Ahven, unit test framework for Ada 95/2005/2012

- http://www.ahven-framework.com/ (web site)
- https://bitbucket.org/tkoskine/ahven (mercurial repository)
- works with Janus/Ada, ICCAda, GNAT
- mature library, users around Europe/US Jdaughter, JSON parser/generator for Ada 95
- https://bitbucket.org/tkoskine/jdaughter
- works with Janus/Ada, ICCAda, GNAT
- under development (no official release) Ladybird, Twitter library/command line client (Ada 95)
- https://bitbucket.org/tkoskine/ladybird
- works with Janus/Ada, ICCAda, GNAT
- under development (no official release) AVR-Ada + Arduino blog
- http://arduino.ada-language.com/
- a blog, not a library
- example code at https://bitbucket.org/tkoskine/arduino-blog

Portable SQLite3 bindings for Ada 95

- https://bitbucket.org/tkoskine/sqlite3-ada
- works with Janus/Ada, ICCAda, GNAT
- under development (no official release)

NFC+PN532 driver for AVR-Ada

- https://bitbucket.org/tkoskine/arduino-pn532
- Reads Mifare Classic tags and NFC type 2/4 tags
- Writes NFC type 2/4 tags
- Emulates NFC type 4 tags
- under development, needs AVR-Ada (avr-gnat)
- I plan to do Raspberry Pi + Linux + native GNAT version at some point

Not Claw Sockets (NC.Sockets) for x86_64 Linux

- https://bitbucket.org/tkoskine/not-claw-sockets-linux-x86
- development has been stalled since 2013

(no time because of other Ada projects)

In addition, I do contributions to AVR-Ada occasionally

- http://sourceforge.net/projects/avr-ada/

Length-Limited Huffman Coding

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sat, 6 Feb 2016 12:57:06 -0800
Subject: Length-Limited Huffman Coding
Newsgroups: comp.lang.ada

Does someone here knows of an Ada implementation of length-limited Huffman coding ? Basically it is the construction of a Huffman tree with a constraint of a limited tree depth (or code length, if you consider the code formed by walking through the tree). References seem to point to two algorithms [1] and [2] which reduces the memory space needed, compared to [1]. Actually [1] would already be OK for my needs ("Dynamic Deflate" compression in Zip-Ada).

[1] A Fast Algorithm for Optimal Length-Limited Huffman Codes Larmore & Hirschberg

[2] A Fast and Space-Economical Algorithm for Length-Limited Coding Katajainen, Moffat & Turpin

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Mon, 8 Feb 2016 13:20:00 -0800
Subject: Re: Length-Limited Huffman Coding
Newsgroups: comp.lang.ada

No need to search, there is an implementation now, translated from katajainen.c (Zopfli project):

http://sf.net/p/unzip-ada/code/HEAD/tree/zip_lib/length_limited_huffman_code_lengths.ads

http://sf.net/p/unzip-ada/code/HEAD/tree/zip_lib/length_limited_huffman_code_lengths.adb

There is an independent test procedure:

http://sf.net/p/unzip-ada/code/HEAD/tree/test/test_llhc.adb

and a minor sighting (so far...) in Zip.Compress.Deflate.

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Thu, 11 Feb 2016 20:44:03 +0200
Subject: Re: Length-Limited Huffman Coding
Newsgroups: comp.lang.ada

> [...]

I am not Randy, but I tested the code with Janus/Ada 3.1.2c (on Windows 10) and it compiled and ran fine:

[...]

Although, Janus/Ada gave one warning:

```
In File
C:\Work\gautier\llhc\LENGTH_LIMITED_HUFFMAN_CODE_LENGTHS.ADB
at line 54
--------------
  53:  type Index_pair is array(Index_type'(0)..1) of Index_type;
  54:  lists: array(0..Index_type(max_bits-1)) of Index_pair;
-----------------------------------------------------^
*WARNING* This component has a non-simple type (6.5.2)
%%%%%%%
```

Cortex GNAT Run Time Systems

From: Simon Wright <simon@pushface.org>
Date: Sun, 07 Feb 2016 22:45:08 +0000
Subject: ANN: Cortex GNAT RTS 20160207
Newsgroups: comp.lang.ada

This release is at Sourceforge[1].

This release includes an RTS for the Arduino Due, arduino-due, and a minimal BSP, arduino-due-bsp.

For the STM32F429I-DISCO, there is one RTS, stm32f429i-disco-rtos, and one BSP, stm32f429i-disco-bsp.

In this release,

- The Containers support generalized iteration ("for all E of C loop"). Note, this is achieved by removing tampering checks. While tampering errors are rare, it would be as well to check algorithms using a fully-featured desktop compiler.
- FreeRTOS is configured to detect stack overflow (if it is detected, the RTS loops inside vApplicationStackOverflowHook()).

The standard packages included (there are more, implementation-specific, ones) are:

- Ada
- Ada.Containers
- Ada.Containers.Bounded_Hashed_Maps
- Ada.Containers.Bounded_Vectors
- Ada.Exceptions
- Ada.IO_Exceptions
- Ada.Interrupts
- Ada.Interrupts.Names
- Ada.Iterator_Interfaces
- Ada.Real_Time
- Ada.Streams
- Ada.Synchronous_Task_Control
- Ada.Tags
- Ada.Task_Identification
- Interfaces
- Interfaces.C
- Interfaces.C.Strings
- System
- System.Assertions
- System.Address_To_Access_Conversions
- System.Storage_Elements
- GNAT
- GNAT.Source_Info

The software is supplied built with for debugging (-g) and with suitable optimisation (-Og), using GNAT GPL 2015 on Mac OS X (it should work out of the box with a Linux-hosted GNAT GPL 2015 cross-compiler, but will need recompiling for another compiler version).

[1] https://sourceforge.net/projects/cortex-gnat-rts/files/20160207/

[See also “Cortex GNAT Run Time Systems”, AUJ 36-3, p. 123. —sparre]

PragmARC.Text_IO

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Sun, 7 Feb 2016 21:50:52 -0700
Subject: Line-Terminator-Independent Text I/O (Mostly)
Newsgroups: comp.lang.ada

The version of the PragmAda Reusable Components for ISO/IEC 8652:2007 now contains PragmARC.Text_IO, a simple text I/O package that can read files with DOS/Windows (CR-LF), Mac (CR), and Unix (LF) line terminators. It can also write files with any of those line terminators.

The PragmARCs are available from the PragmAda website at

https://pragmada.x10hosting.com/pragmarc.htm

or from the GitHub repository at

https://github.com/jrcarter/PragmARC.git

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Sun, 21 Feb 2016 22:49:37 -0700
Subject: New Version of PragmARC.Text_IO
Newsgroups: comp.lang.ada

The initial version of this was adding an extra (native) EOL to files when they were closed. This has now been corrected. The package specification has changed somewhat in the process.

The PragmAda Reusable Components are available from the PragmAda web site or from

https://github.com/jrcarter/PragmARC

[See also “PragmAda Reusable Components”, AUJ 35-3, p. 154. —sparre]

Open Z Wave

From: Tony G. <tonythegair@gmail.com>
Date: Mon, 15 Feb 2016 04:31:05 -0800
Subject: Ada Open Zwave
Newsgroups: comp.lang.ada

Here is a binding to the c++ open zwave binding at www.openzwave.com

The license is the GNAT Modified GPL.

The binding is at https://github.com/tonygair/ada_open_zwave

I am using it to create a home heating / lighting efficiency application to run on a raspberry pi B. The hardware also including a AEON Z2 Stick and Stella Z radio valves.

I am also using POLYORB, the Ada distributed systems annex implementation, and David Botton's truly fantastic GNOGA to make it.

So far it works on the PI and on PC's in Debian and Raspbian Jessie but don't forget to add your the ID you are using it under to the group DIALOUT.

i.e. sudo adduser tony dialout

The heating application will also be released soonish also under the GMGPL license.

I hope it's useful! and improvements to its setup will be gratefully received!

From: Tony G. <tonythegair@gmail.com>
Date: Tue, 16 Feb 2016 02:19:56 -0800
Subject: Re: Ada Open Zwave
Newsgroups: comp.lang.ada

This library was commissioned by myself and designed and implemented by Jeffrey R Carter. Dave Botton also provided some good advice at the start of the project.

Jeffrey R Carter did an excellent job in a difficult project and was a pleasure to work with. I highly recommend his services to other people who also require work done at the limits of the possible.

[See also “Job: Writing a Binding to Open Z Wave”, AUJ 36-2, p. 71. —sparre]

Remote Control of Light Bulbs

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Mon, 15 Feb 2016 21:28:00 +0100
Subject: ANN A-LIFX
Newsgroups: comp.lang.ada

Finally I got the interface package for the LIFX-Bulbs http://www.lifx.com/ in fairly stable shape.

https://github.com/persan/A-LIFX

Wave Files

From: Gustavo <gusthoff.ada@gmail.com>
Date: Sun, 28 Feb 2016 12:42:29 -0800
Subject: Wavefile Reader & Writer Package
Newsgroups: comp.lang.ada

I've been working in my free time on the implementation of a wavefile reader & writer in Ada. Maybe this could be useful for some of you:

https://github.com/gusthoff/wavefiles/

Comments are, of course, welcome!

[See also “Sound recording API for Linux”, AUJ 33-3, p. 144. —sparre]

Stream Tools

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Mon, 29 Feb 2016 07:43:11 +0100
Subject: [ANN] a-tream-tools-1.0.4
Newsgroups: comp.lang.ada

https://github.com/persan/a-stream-tools/releases/tag/a-stream-tools-1.0.4-2016-02-29

News in this release:

- Added send_socket for better integration with gnat_socket.
- Added the capability to read all the remaining elements in the stream without getting End_Error.

[See also “Stream Tools”, AUJ 36-2, p. 64. —sparre]

Ada-related Products

GNAT Pro

From: AdaCore Press Center
Date: Sat Jan 23 2016
Subject: AdaCore Releases GNAT Pro 7.4
URL: http://www.adacore.com/press/gnat-pro-7-4/

New version of Ada Development Environment highlights annual major release of company product line

EMBEDDED WORLD 2016, Nuremberg, Germany, February 23, 2016 — AdaCore, a company offering development and verification tools for reliable, safe and secure software, today released the latest version of its flagship GNAT Pro Ada Development Environment. GNAT Pro 7.4 incorporates new functionality, a number of performance improvements, additional platform support including several new embedded targets, and many other enhancements. GNAT Pro is part of the company’s annual major release cycle for its products, and Q1 2016 will also see new versions of the CodePeer deep static analysis tool for Ada, the SPARK Pro verification environment for high-integrity software, and the QGen model-based development and verification tool for Simulink® and Stateflow® models.

GNAT Pro includes a full Ada compiler, Integrated Development Environments – the GNAT Programming Studio (GPS) and the Eclipse-based GNATbench – a comprehensive toolset including a visual debugger, and an extensive set of libraries and bindings.

GNAT Pro 7.4 continues to build upon the strong foundation of gcc 4.9 while upgrading to the gdb 7.10 debugger technology. It supports Windows 10 as well as several new target platforms, in particular VxWorks 7 (ARM, e500v2, PPC, x86_64), VxWorks 653 3.0, and PikeOS (PowerPC). Among the more than 120 new features are the following enhancements:

- Generating C headers from Ada package specifications, which complements the existing facility (-fdump-ada-spec) for deriving Ada package specs from C header files
- Detecting invalid memory access via libsanitizer on Linux
- Enabling SSE floating point extensions by default on all x86 native ports
- Better performance for the Ada.Containers library, for example in the implementation of "for...of" loops and iterations
- New pragmas to support low-level programming on bareboard targets
- New pragmas to ease porting existing codebases from other Ada compiler environments

“As we do each year with GNAT Pro, this new release brings a wide assortment of new features,” said Cyrille Comar, AdaCore President. “It also makes many existing tools more robust or easier to use, and these are worth mentioning. One example is the GNATtest utility, which automatically generates ready-to-use unit test frameworks. And I’d also like to highlight the distributed feature of GPRbuild, which can now take advantage of server farms as well as multicores. Advanced users have reported incredible savings in build time for huge applications through this enhancement.”

[See also “GNAT Pro”, AUJ 36-2, p. 65. —sparre]

QGen

From: AdaCore Press Center
Date: Sat Jan 23 2016
Subject: AdaCore Releases QGen 2.1
URL: http://www.adacore.com/press/qgen-2-1/

New version of model-based development and verification toolset brings improved performance, new features

EMBEDDED WORLD 2016, Nuremberg, Germany, February 23, 2016 — AdaCore, a company offering development and verification tools for reliable, safe, and secure software, today released the latest version of its model-based development and verification toolset, QGen. QGen provides a qualifiable and customizable code generator from Simulink® and Stateflow® models to the safety-oriented programming languages SPARK (a formally analyzable Ada subset) and MISRA C. QGen 2.1 supports essentially all constructs used for modeling safety-critical control systems. In addition, QGen 2.1 offers a number of other enhancements including optimization of code for switch blocks, the ability to add external code for Lookup tables and Prelookup blocks, support for commented-out / commented-through blocks, and factoring of code for reference models and model libraries. QGen 2.1 is compatible with MATLAB versions 2008b through 2015b.

As a prototype capability, initial support for model-level debugging is available as a supplement to QGen 2.1. Using the GNAT Programming Studio (GPS) IDE, developers can debug both “pure” Simulink®/Stateflow® models and applications that combine manually prepared code with the auto-generated code. At the start of a debug session, side-by-side views of the model and the corresponding generated SPARK/Ada or MISRA C code are displayed. The developer can set breakpoints on individual blocks, which will automatically set breakpoints at the corresponding points in the generated code. Other capabilities include stepping execution one block at a time, viewing the values of signal variables, and stepping into or out of model subsystems. Host (native) debugging, and cross-debugging with any target supported by GDB, will be provided in a subsequent release.

“We designed QGen from the start to support qualified code generation directly from Simulink® and Stateflow® models," said Tucker Taft, Product Architect for QGen, "and this has struck a responsive chord with our customers in the automotive and aerospace industries and other safety-critical software-intensive domains. QGen 2.1 will help reduce the effort in model-based development and verification, and future releases will continue to broaden the QGen product line. Among the planned enhancements is support for transforming high-level requirements into executable and verifiable assertions at the model and code levels."

[See also "QGen", AUJ 36-2, p. 66. —sparre]

SPARK Pro

From: AdaCore Press Center
Date: Wed Jan 27 2016
Subject: AdaCore Releases SPARK Pro 16
URL: http://www.adacore.com/press/adacore-releases-spark-pro-16/

Formal verification toolset helps reduce certification effort for safety-critical and high-security systems

ERTS2 2016, Toulouse, France, January 27, 2016 - AdaCore today announced the latest release of its SPARK Pro integrated development and verification environment, bringing a sound and mathematics-based static analysis technology to the challenges of software verification for high-assurance systems. SPARK Pro 16 provides enhanced coverage of SPARK 2014 language features and now supports the Ravenscar tasking profile, thus extending the benefits of formal verification methods to a safe subset of Ada 2012 concurrent programming features. As another improvement SPARK Pro 16 can generate counterexamples to verification conditions that cannot be proved, thus making it easier for developers to find defects in the functional code or in the supplied contracts. SPARK Pro 16 also improves the handling of bitwise (modular) operations, and the product's proof engine now includes the Z3 SMT solver.

The SPARK Pro technology, which has been jointly developed by AdaCore and its partner Altran, can prove SPARK program properties ranging from absence of run-time errors (exceptions) to compliance with a formally defined requirements specification. SPARK Pro thereby helps reduce the cost of software verification and simplifies the task of certifying the software against safety or security standards. The technology is sound; that is, there are no "false negatives": if SPARK Pro reports that a program is free of a certain kind of error, then that error cannot occur. It also has a very low "false positive" rate, which is important in practice, and its efficient SMT solver technology scales up for usage in large projects. The SPARK language and toolset can be used from the outset on new projects or introduced incrementally into an existing project, allowing a "hybrid" verification approach that combines formal methods with traditional testing techniques.

"With this new version of the SPARK Pro toolset, we get one step closer to our goal: to make it easy for software engineers to start relying heavily on static verification and formal proofs without needing expertise in mathematical logic," said Cyrille Comar, AdaCore President. "Not only are most of the needed proofs conducted completely automatically, but many language restrictions usually associated with such proof capabilities have been lifted. This makes the writing of proven software both efficient and pleasant".

About SPARK Pro

SPARK Pro provides the foremost language, toolset, and design discipline for engineering high-assurance software. It combines the SPARK language and verification tools with AdaCore's GNAT Programming Studio (GPS) and GNATbench Integrated Development Environments.

The SPARK Pro toolset offers static verification that is unrivalled in terms of its soundness, low false-alarm rate, depth and efficiency. The toolset generates evidence for correctness, including proofs of the absence of run-time errors, which can then be used to meet the requirements of safety and security certification schemes, such as DO-178B and DO-178C (airborne systems), EN 50128 (railway systems), and the Common Criteria. SPARK Pro is especially applicable in the context of the Formal Methods supplement to DO-178C.

About SPARK 2014

SPARK 2014 is a formally analyzable programming language especially suited for developing ultra-low defect software in critical applications, for example where safety and/or security are key requirements. The SPARK language has a solid industrial track record with a 25+ year history of successful usage worldwide in a range of industrial applications including civil and military avionics, railway signaling, cryptography, and cross-domain solutions. SPARK 2014 is the next generation of this leading software technology, offering key benefits such as support for hybrid verification (combining formal methods with traditional testing), executable contracts, a larger Ada language subset (for example including generic units), and convergence with Ada 2012 syntax (making it easier to combine Ada and SPARK components). An on-line interactive course on SPARK 2014, part of the AdaCore University curriculum, is available at http://university.adacore.com/courses/spark-2014/.

[See also "SPARK Pro", AUJ 35-2, p. 81. —sparre]

Apex Ada, Object Ada and Ada World

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Wed, 10 Feb 2016 17:30:08 +0100
Subject: PTC Ada products
Newsgroups: comp.lang.ada

Short summary of PTC Ada Group's presentation of today. [PTC has acquired Atego (recursively), also former Rational/IBM Ada products.]

Some emphasized words:

- recommitted [sic] to Ada products after integration with PTC, focused
- licensing is moving towards subscription
- support portal

Three Ada products:

- Apex Ada (Ada 2005, v.5.0)
- Object Ada (Ada 2005, v.9.1+)
- Ada World (Ada 83, limited support, legacy)
- 3rd party additions like ASIS based Ada analyzer

IDEs:

- Linux (Motif), Eclipse, Win32 IDE, ...
- Debugger can now handle DLLs [Note: this is a programmable debugger]

About hosts/targets:

- VxWorks, Solaris, LynxOS, Linux (incl 64 bit), Windows 7+, ...
- Ravenscar
- bare exec
- mostly Intel, Sparc, PowerPC; native and X-compilers
- updates for RTOSs, including support for new ones, are based on customers' requirements, and funding

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Wed, 10 Feb 2016 20:07:59 +0200
Subject: Re: PTC Ada products
Newsgroups: comp.lang.ada

> [...]

> - licensing is moving towards subscription

I asked about ObjectAda from Atego some years ago.

At that time there were two problems:

1) Subscription based licensing, you lose the right to use the compiler when the subscription ends.

2) They only deal with other companies and have no processes how to deal with invidual people

As a Northern European person with a day job, it makes no sense for me to create a company. I also could pay somewhat largish price for perpetual license with 1y support contract, but it makes no sense (for hobbyist) if I cannot use the compiler after that one year.

It was also interesting that one largish client of Atego asked if Atego could simply give me a license for one of their Ada compiler, but that wasn't possible since I did not have that company. (The problem was not money if I understood correctly.)

It would be nice if merge to PTC has brought some changes to this, but I haven't had time to ask about it personally.

Ada and Operating Systems

Mac OS X: GCC for ARM-EABI

From: Simon Wright <simon@pushface.org>
Date: Tue, 26 Jan 2016 17:48:34 +0000
Subject: ANN: GCC 5.2.1 arm-eabi for OS X El Capitan
Newsgroups: comp.lang.ada

This release is on Sourceforge[0].

It must be installed over GCC 5.2.0 (or 5.2.1 if you've built one ...)

This is GCC 5.2.1 for arm-eabi from GCC ARM Launchpad[1], tested on the Cortex-M3 as found on the Arduino Due[2] and the Cortex-M4 as found on the STMicroelectronics[3] STM32F4 Discovery and STM32F429I Discovery boards.

The processors supported are Cortex-M3, Cortex-M4, Cortex-M0+, Cortex-M7, Cortex-R4, Cortex-R5 and Cortex-R7.

The compiler comes with no Ada Runtime System (RTS). See the Cortex GNAT Run Time Systems project[4] for candidates.

The compiler is known to run on El Capitan; it may not run on earlier OS X releases.

[0] https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/5.2.0/arm-eabi/

[1] https://launchpad.net/gcc-arm-embedded (gcc-arm-none-eabi-5_2-2015q4-20151219 release)

[2] http://www.arduino.com

[3] http://www.st.com

[4] https://sourceforge.net/projects/cortex-gnat-rts/)

[See also "Mac OS X: GCC", AUJ 36-4, p. 206. —sparre]

Ada Inside

POSIX Shell for Windows

From: AdaCore Press Center
Date: Wed Dec 9 2015
Subject: AdaCore/gsh
URL: https://github.com/AdaCore/gsh

GSH - A POSIX shell for Windows

General Information

GSH is an implementation of a POSIX shell developed for the Windows platform.

The aim of the project is to provide an efficient UNIX shell instantiation for Windows, for non interactive usage.

GSH can be used to compile projects depending on autotools, UNIX make,... As it targets specifically Windows platform, GSH differs significantly from the most used ones such as bash, zsh. Among others, its implementation does not depend on the 'fork system call' and doesn't emulate that system call (as it's done on Cygwin). This allows better compiling performance (the build times can be up to 3 or 4 times faster than builds performed by projects such as Cygwin).

[...]

Mine Detector

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Thu, 4 Feb 2016 11:51:49 -0700
Subject: Mine Detector for GtkAda 3
Newsgroups: comp.lang.ada

Thanks to Pascal Malaise, there is now a version of Mine Detector that works with GtkAda 3 available from

https://pragmada.x10hosting.com/mindet.html

[See also "Mine Detector", AUJ 36-1, p. 19. —sparre]

LinXtris

From: Pascal Pignard <p.p11@orange.fr>
Date: Mon, 8 Feb 2016 21:49:56 +0100
Subject: LinXtris with Gnoga.
Newsgroups: gmane.comp.lang.ada.gnoga

First port to Gnoga of LinXtris version 0.1a2 from GTKAda (http://sourceforge.net/projects/linxtris).

GTKAda source lines has been left as comments in code to show the translation.

Instructions:

```
$ make linxtris
$ bin/linxtris -data_dir demo/linxtris/
```

Note: option "continuous movement" is quite bad on my computer, maybe some rework is needed on image display.

Muen Separation Kernel

From: Reto Buerki <reet@codelabs.ch>
Date: Mon, 22 Feb 2016 11:52:44 -0000
Subject: [ANN] Muen development version 0.7 released
Newsgroups: comp.lang.ada

We are proud to announce the availability of Muen version 0.7.

The following major features and improvements have been implemented since the last announcement:

Support for Genode VM subjects

Through the close collaboration with Genodelabs in Dresden [1], the Genode OS framework has been ported to run as subject on top of the Muen separation kernel. This allows the robust combination of the static, low-complexity Muen SK with the feature-rich and extensive Genode ecosystem. The result is a flexible platform for the construction of component-based high-assurance systems.

For more information about our work in this area see the Genode release notes [2].

Subject time mechanism

Giving untrusted subjects access to high-resolution time sources is often problematic from a security perspective as it makes way to measure subtle timing differences in execution behavior, enabling side-channel attacks. One mechanism to make such attacks harder is to provide only coarse grained time sources to untrusted code. To this end we implemented a time virtualization mechanism by providing a timeserver component in SPARK 2014, which exports time information with microsecond granularity via shared memory. Only the timeserver has access to the TSC high-resolution timer of the CPU and the Real-Time Clock (CMOS/RTC).

Other subjects derive the absolute and relative time from the exported values without the need to access hardware time sources. For Linux, we implemented a paravirtualized TSC driver and CMOS/RTC emulation in the associated subject monitor (SM).

Hardware and platform policy abstractions

The XML system policy has been augmented with hardware resource and platform description abstractions.

The hardware section describes the hardware resources provided by the target machine and can be automatically generated using the mugenhwcfg [3] tool. By providing an automated mechanism for hardware information collection, the process of supporting new target hardware has been greatly simplified.

Using the platform layer, an unified view of the hardware resources across different physical machines can be achieved. This enables integrators to deploy the same system policy across a wide range of hardware targets.

Linux virtual filesystem and network interface drivers

The muenfs [4] Linux kernel module implements a virtual file system that facilitates user-space access to inter-subject memory channels. Filesystem operations are used to exchange data with other subjects.

The muennet [5] Linux kernel module implements a virtual network interface driver which sends and receives data via shared memory channels. From the perspective of a Linux user-space application, a network interface created using the muennet kernel module behaves just like an ordinary network interface.

These new modules enable applications running on Linux to conveniently communicate and interact with other subjects of a component-based system running on Muen.

Further changes and improvements include:

- Support for Message Signaled Interrupts (MSI)
- Debug server subject written in Ada 2012
- VT subject written in Ada 2012
- Various toolchain improvements and optimizations

One particularly exciting aspect of our work related to the aforementioned Genode framework is that we were able to utilize the base-hw x86_64_muen kernel port to execute 32-bit Windows (7-10) guest VMs using the Genode VirtualBox support on top of Muen. To achieve this, we implemented a VirtualBox hardware execution layer for hw_x86_64_muen. We plan to pursue this line of work in order to properly integrate Windows VM support as a feature of Muen.

The mugenhwcfg tool for automated generation of hardware configuration is the result of a 12-week internship by Chen Chin Jieh, a student from the Nanyang Technological University Singapore. We are very happy with the result and would like to thank him for his contribution to the Muen project.

Further information about Muen is available on the project website [6] and the git repository can be found at [7].

Please feel free to give the latest development version of Muen a try. Feedback is much appreciated!

[1] - http://genode-labs.com/

[2] - http://genode.org/documentation/release-notes/15.08

[3] - http://git.codelabs.ch/?p=muen/mugenhwcfg.git

[4] - http://git.codelabs.ch/?p=muen/linux/muenfs.git

[5] - http://git.codelabs.ch/?p=muen/linux/muennet.git

[6] - http://muen.codelabs.ch/

[7] - http://git.codelabs.ch/?p=muen.git

[See also "Muen Separation Kernel", AUJ 36-1, p. 16. —sparre]

Ada in Context

Iterators for Directories and Environment Variables

From: Yuta Tomino <aghia05@gmail.com>
Date: Fri, 27 Nov 2015 07:25:57 -0800
Subject: Two approaches of iterators for the key-value pairs
Newsgroups: comp.lang.ada

Hello, I'm reading AI12-0009-1 "Iterators for Directories and Environment_Variables" [1]. And there is interesting difference between the new iterator of Ada.Environment_Variables and the existing iterators. [...]

A loop for Ada.Environment_Variables would be like below, according to this AI:

```
for E *of*
   Ada.Environment_Variables.All_Variables
loop
   -- E is Iterator_Element
         (Name_Value_Pair_Type)
   Name (E) -- key
   Value (E) -- value
end loop;
```

On the other hand, as we already know, a loop for Ada.Containers.Hashed_Maps/ Ordered_Maps:

```
for I *in* The_Map_Object.Iterate loop
   -- I is Cursor
   Key (E) -- key
   Element (E), Reference (E).Element.all
    -- value
end loop;
```

If you create new iterator for some key-value pairs, which approach do you like?

[1] http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0009-1.txt?rev=1.4

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 27 Nov 2015 17:30:17 +0100
Subject: Re: Two approaches of iterators for the key-value pairs
Newsgroups: comp.lang.ada

> [...]

>

> If you create new iterator for some key-value pairs, which approach do you like?

Rather this:

```
for Variable in All_Variables loop
   Put_Line (Variable.Name & "=" &
         Variable.Value);
end loop;
```

From: Brad Moore <bmoore.ada@gmail.com>
Date: Fri, 27 Nov 2015 14:52:05 -0800
Subject: Re: Two approaches of iterators for the key-value pairs
Newsgroups: comp.lang.ada

> [...]

My initial implementation of this was as an Iterator, rather than an Iterable container, and in my implementation, I got things to work this way without the Cursor being a tagged type.

My "proof of concept" implementation was for Ada.Directories, but the same approach would also work for Ada.Environment_Variables.

For Ada.Directories, I declared the Cursor type as:

```
type Cursor (Directory_Entry : not null
access constant Directory_Entry_Type) is
private
   with Implicit_Dereference =>
         Directory_Entry;
```

The Implicit_Dereference aspect was added to Ada 2012.

For Ada.Environment_Variables, I would have declared something like:

```
type Cursor (Name_Value_Pair : not null
access constant Name_Value_Pair_Type)
is private
   with Implicit_Dereference =>
         Name_Value_Pair;
```

And

```
function Has_Element (Name_Value_Pair
         : Cursor) return Boolean;

package Environment_Variable_
         Iterator_Interfaces is new
   Ada.Iterator_Interfaces (
         Cursor  => Cursor,
         Has_Element => Has_Element);

function All_Variables return
Environment_Variable_Iterator_Interfaces.
Forward_Iterator'Class;
```

Then one would be able to write;

```
for Variable in All_Variables loop
   Put_Line (Variable.Name & "=" &
         Variable.Value);
end loop;
```

Exactly as Dmitry had requested...

From: Brad Moore <bmoore.ada@gmail.com>
Date: Fri, 27 Nov 2015 15:11:44 -0800
Subject: Re: Two approaches of iterators for the key-value pairs
Newsgroups: comp.lang.ada

[...]

Getting feedback on the preferences of people could influence the direction that the AI takes, and ultimately we want to make the best decisions, so it's good to hear the feedback.

[...]

My sense is that if you don't need a container as part of the abstraction, then perhaps it's better to just provide an Iterator type. All the existing containers already were containers, so for those it made sense to make them Iterable containers. Ada.Directories and Ada.Environment_Variables do not currently have Containers associated with them, so perhaps an argument can be made that they should be Iterators, rather than concoct a Container type so that Iterable container syntax can be used.

Volatile Full Access

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 7 Dec 2015 20:11:15 -0600
Subject: Re: STM32F4 GNAT Run Time System - roadmap
Newsgroups: comp.lang.ada

> [...]

The ARG has decided on a different direction to fix the problem addressed by Volatile_Full_Access; essentially, accesses to non-volatile components of atomic objects have to be accessed with a read-modify-write cycle. (See AI12-0128-1.) Various parts of Annex C will be rewritten to make that make sense.

I think someone here originally proposed that (or at least something like it); it took us quite a while to come around to that view, mainly because of compatibility concerns. But there is also the concern of doing something unexpected.

Anyway, I don't expect that there is much use of that GNAT-specific aspect once the changes are approved (probably will happen next year at our next meeting).

Everything You Know About Storage Space is Wrong

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 26 Dec 2015 17:37:04 -0700
Subject: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

When I started out in S/W development, I learned some rules, such as, "Integer math is much faster than floating point," and, "Memory is scarce."

In the 90s, processors began to typically have an FPU, and floating-point math became as fast as integer, and in some cases, since it could proceed in parallel with the CPU, faster.

When computers began to commonly have RAM in quantities measured in GB rather than KB or MB, memory ceased to be scarce, and things that were previously laughable, such as

```
type Int_Set is array (Integer) of Boolean;
for Int_Set'Component_Size use
        Boolean'Size;
```

became possible (for a 32-bit Integer, Int_Set'Size would be 512 MB). What I knew is wrong.

Today we learn that memory is much faster than persistent storage. That may soon be wrong, too. I've been reading about non-volatile memory research, and it seems that in a few years NV RAM will be available as fast current RAM and as persistent and durable as current disks.

This will no doubt revolutionize computer architectures and programming languages. Instead of computers with distinct memory and storage, there will probably be computers with lots of NV RAM (1-10 TB?) but no disk.

People will no doubt still want a hierarchical naming system for data stored in that memory, but presumably S/W will map variables onto these "files". So instead of the current "open, loop over read/modify/write, close" paradigm, we might have something like

```
type R is record ...
type L is array (Positive range <>) of R;
F : mapped L with File_Name => "name";
All_Records : for I in F'range loop
        -- or "of F"
```

where the bounds of F will be determined from "name". A mechanism will be needed for collections of heterogeneous data as well. F would be equivalent to a Direct_IO file with in-out mode.

I would think that the Ada 2X project should be thinking about these things, and wonder what others here think about them.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sun, 27 Dec 2015 08:55:38 +0100
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

> [...]

Time to resurrect Multics?

(For the education of the young generation: Multics was an OS of the 60s-70s, where everything was organized as "segments" of virtual memory. There was a command to list "named segments" (equivalent of files), which was naturally named... "ls".)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 27 Dec 2015 09:46:31 +0100
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

> [...]

There will be no files and no I/O.

The idea of a memory-mapped object-oriented system is nothing new. On the contrary, it is more than 20 years old.

BTW, we always will have multiple speed memory. When the memory of each computer will get a unique place in the global address space and thus networking I/O will be eliminated, remote memory mapping will remain much slower than the local one. Similarly, locally shared writable memory will always be slower than the single-ported one and that slower than cache etc.

> [...]

> F : mapped L with File_Name => "name";

> [...] F would be equivalent to a Direct_IO file with in-out mode.

No, it will be equivalent to a container library.

BTW, Direct_IO stems from block-oriented devices, namely disks. Once there will be no I/O there will be no need to slice data into same sized blocks.

> [...]

Well, long ago I wrote here about requirements the language must have in order to support such persistent memory. The most important one is getting away from the trusted model. E.g. operations of a persistent protected object must go through the supervisor mode in order to keep protected members out of the caller's memory space. And you won't get anywhere without proper interfaces and an elaborated type system.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sun, 27 Dec 2015 10:36:59 -0700
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

[...]

> No, it will be equivalent to a container library.

Yes, thinking more about the idea, when current S/W writes a file, it often has no idea how big that file will be until it's finished. The equivalent would seem to be a memory-mapped unbounded container that persists, with a name, after the program ends.

Another thought I've had is the need to wipe non-mapped objects at program termination for security.

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Mon, 4 Jan 2016 15:44:57 +0100
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

[...] And we'll have problems with the fact that rebooting will not be tabula rasa ;) New jokes ahead?

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Mon, 28 Dec 2015 10:57:16 +0100
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

> [...]

Maybe the word "distributed" needs to include more features of the language and its library.

Considering economy, the increase in resources does not seem to always entail new algorithms or corresponding new features of languages(*). I have seen it simplify scaling when we could just reuse the same program in a larger address space.

A program system that serves many instead of one will save the cost of a few computers or their parts, so virtualization covers some use cases already, without programs needing to change.

[...]

(*) “We don’t have better algorithms. We just have more data.” -- P. Norvig

From: Bruce B.
<brbarkstrom@gmail.com>
Date: Tue, 5 Jan 2016 05:52:07 -0800
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

There are a three articles in the new Comm. ACM that pick up on this theme:

- Greengard, S.: Better Memory, pp. 23-25
- Mamavati, M., Schwazkopf, M., and Warfield, A.: Non-Volatile Storage
- Helland, P.: Immutability Changes Everything

The first two of these are on NVRAM and architectural changes that are similar to the comments in this thread. The last one deals more with database design when you don't have to worry so much about versions because you can just store everything.

This thread might also want to pick up on the security implications of using block ciphers instead of just the usual encryption.

From: Jacob Sparre Andersen
<jacob@jacob-sparre.dk>
Date: Mon, 29 Feb 2016 13:14:08 +0100
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

> [...]

Some of the rules which (apparently) aren't wrong yet:

http://www.cse.msu.edu/~cse320/Documents/FloatingPoint.pdf

From: Jacob Sparre Andersen
<jacob@jacob-sparre.dk>
Date: Mon, 29 Feb 2016 13:27:36 +0100
Subject: Re: Everything You Know Is Wrong
Newsgroups: comp.lang.ada

> [...]

> F : mapped L with File_Name => "name";

> [...]

>

> I would think that the Ada 2X project should be thinking about these things, and wonder what others here think about them.

I think it looks somewhat similar to some of my experiments on simplified persistence in Ada:

http://www.jacob-sparre.dk/persistence/ae2010-slides.pdf

http://www.jacob-sparre.dk/programming/persistent_containers-2015-paper.pdf

Considering the difficulties of implementing persistence in a simple way with a library, I think it is worthwhile experimenting with something like this - and considering it for the next major revision of Ada.

Conference Calendar

Dirk Craeynest

KU 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.

2016

April 02-08 19th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2016), Eindhoven, the Netherlands. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FOSSACS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems).

> April 03 13th **International Workshop on Formal Engineering approaches to Software Components and Architectures** (FESCA'2016). Topics include: (semi-)formal techniques and their application that aid analysis, design and implementation of software applications; formal modelling of component-based, timed and hybrid systems; temporal properties and their formal verification; interface compliance and contractual use of components; static and dynamic analysis; industrial case studies and experience reports; etc.

April 04-08 31st ACM **Symposium on Applied Computing** (SAC'2016), Pisa, Italy.

> April 03-08 **Track on Software Verification and Testing** (SVT'2016). Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, etc.
>
> April 03-08 **Track on Software Engineering** (SE'2016). Topics include: software architecture, and software design patterns; standards; maintenance and reverse engineering; quality assurance; verification, validation, testing, and analysis; safety, security, and risk management; dependability and reliability; fault tolerance and availability; formal methods and theories; component-based development and reuse; empirical studies, and industrial best practices; applications and tools; distributed, embedded, real-time, high-performance, highly dependable systems; etc.
>
> ☺ April 04-08 **Track on Programming Languages** (PL'2016). Topics include: compiling techniques, domain-specific languages, 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.
>
> ☺ April 04-08 **Track on Multicore Software Engineering, Performance, Applications and Tools** (MUSEPAT'2016). Topics include: software engineering for multicore (CPU or GPU); specification and modeling of multicore systems; programming models, languages, compiler techniques and development tools for multicore; parallel and distributed testing and debugging; evolving sequential software to leverage multicore and manycore hardware; performance and optimization of multicore software; domain- and platform-specific multicore software issues (e.g., issues in scientific computing); etc.
>
> ☺ April 04-08 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2016). Topics include: aspects and components; code generation and optimization; distribution and concurrency; formal verification; integration with other paradigms; interoperability,

versioning and software evolution and adaptation; language design and implementation; modular and generic programming; runtime verification; secure and dependable software; static analysis; testing and debugging; type systems; etc.

April 05-08 — 13th **Working** IEEE/IFIP **Conference on Software Architecture** (WICSA'2016), Venice, Italy. Topics include: re-factoring and evolving architecture design decisions and solutions; techniques and tools for technical debt management; managing architecture risk over the life cycle of a system; architecture description languages and model driven architecture; software architecture modelling, analysis methods and tools; software architecture for legacy systems and systems integration; open architectures, product-line architectures, software ecosystems, systems of systems; software architects' roles and responsibilities; training, education, and certification of software architects; industrial experiments and case studies; etc.

April 05-08 — 10th **Conference for Component-Based Software Architectures** (CompArch'2016), Venice, Italy. Topics include: re-factoring and evolving architecture design decisions and solutions; techniques and tools for technical debt management; managing architecture risk over the life cycle of a system; architecture description languages and model driven architecture; software architecture modelling, analysis methods and tools; software architecture for legacy systems and systems integration; open architectures, product-line architectures, software ecosystems, systems of systems; software architects' roles and responsibilities; training, education, and certification of software architects; industrial experiments and case studies; etc.

April 06-08 — 8th **International Symposium on Engineering Secure Software and Systems** (ESSoS'2016), London, UK. Topics include: automated techniques for vulnerability discovery and analysis; programming paradigms, models, and domain-specific languages for security; verification techniques for security properties; security by design; static and dynamic code analysis for security; processes for the development of secure software and systems; embedded software security; etc. Includes paper: "Progress-Sensitive Security for SPARK".

April 10-15 — 9th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2016), Chicago, Illinois, USA. Topics include: security testing, embedded software testing, testing concurrent software, testing large-scale distributed systems, testing in multi-core environments, quality assurance, model checking, testing of open source and third-party software, software reliability, formal verification, experience reports, etc.

April 11-13 — 18th **International Real-Time Ada Workshop** (IRTAW'2016), Benicàssim, Spain. In cooperation with Ada-Europe.

April 27-28 — 11th **International Conference on Evaluation of Novel Approaches to Software Engineering** (ENASE'2016), Rome, Italy. Topics include: comparing novel approaches with established traditional practices and evaluating them against software quality criteria, software and systems development methodologies, software process improvement, software product line engineering, architectural design and frameworks, software quality management, software change and configuration management, application integration technologies, geographically distributed software engineering, formal methods, model-driven engineering, etc.

April 27-29 — XIX **Iberoamerican Conference on Software Engineering** (CIbSE'2016), Quito, Ecuador. Topics include: languages, methods, processes, and tools; reverse engineering and software system modernization; software evolution and maintenance; model-driven engineering; proof, verification, and validation; quality, measurement, and assessment of products and processes; formal methods applied to software engineering; software product families and variability; software reuse; etc.

☺ May 14-22 — 38th **International Conference on Software Engineering** (ICSE'2016), Austin, Texas, USA. Deadline for early registration: April 4, 2016.

> May 15 — 4th FME **Workshop on Formal Methods in Software Engineering** (FormaliSE'2016). Topics include: integration of FMs in the software development life cycle, ability of formal methods to handle real-world problems, formal methods in a certification context, "lightweight" or usable FMs, application experiences, formal approaches to safety and security related issues, cyber physical systems, scalability of FM applications, rigorous software engineering approaches and their tool support, formal approaches to safety and security related issues, case studies developed/analyzed with formal approaches, etc.

☺ May 17-20 — 19th IEEE **International Symposium On Real-Time Computing** (ISORC'2016), York, UK. Topics include: object/component/service-oriented real-time distributed computing (ORC) technology; programming and system engineering (ORC paradigms, languages, model-maintenance, time-predictable systems, ...), system software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...), applications (medical devices, intelligent transportation systems, industrial automation systems, embedded systems, ...), system evaluation (timing, dependability, fault detection and recovery time, ...), cyber-physical systems, etc.

May 23-27 — 30th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2016), Chicago, Illinois, USA.

May 30 - Jun 02 — 12th **International Conference on Open Source Systems** (OSS'2016), Gothenburg, Sweden. Topics include: adoption, use, acceptance of FLOSS; expanding scientific research and technology development methods through openness; security of FLOSS; interoperability, portability, scalability of FLOSS; open standards; reuse in FLOSS; FLOSS for education; FLOSS in the public sector; etc.

June 01-03 — 20th **International Conference on Evaluation and Assessment in Software Engineering** (EASE'2016), Limerick, Ireland. Topics include: any aspect of empirical software engineering.

June 01-05 — 12th **International Conference on integrated Formal Methods** (iFM'2016), Reykjavík, Iceland. Topics include: hybrid approaches to formal modelling and analysis; i.e., the combination of (formal and semi-formal) methods for system development, regarding modelling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.

June 05-07 — 15th **International Conference on Software Reuse** (ICSR'2016), Limassol, Cyprus. Topics include: COTS-based development and reuse of open source assets; generative development; domain-specific languages; software composition and modularization; model-driven development; reengineering for reuse; software product line techniques; quality assurance for software reuse, such as testing and verification; reuse of non-code artifacts (process, experience, etc.); transition to software reuse; industrial experience with reuse; software evolution and reuse; etc.

June 07-09 — 8th **NASA Formal Methods Symposium** (NFM'2016), Minneapolis, Minnesota, USA. Topics include: identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems; model checking; static analysis; model-based development; design for verification and correct-by-design techniques; applications of formal methods in the development of autonomous systems cyber-physical, embedded, and hybrid systems, ...; use of formal methods in: assurance cases, automated testing and verification, ...; etc.

June 10-14 — 40th **Annual** IEEE **Computer Software and Applications Conference** (COMPSAC'2016), Atlanta, Georgia, USA. Event includes: symposiums on Computer Education and Learning Technologies (CELT), Embedded & Cyber-Physical Environments (ECPE), IT in Practice (ITIP), Novel Applications & Technology Advances in Computing (NATA), Security, Privacy and Trust in Computing (SEPT), Software Engineering Technology and Applications (SETA), etc.

♦ June 13-17 — 21st **International Conference on Reliable Software Technologies - Ada-Europe'2016**. Pisa, Italy. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN, and the Ada Resource Association (ARA).

> June 17 — 3rd **International Workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering** (De-CPS 2016). Topics include: Industrial challenges and experience reports on co-engineering for multiple dependability concerns in CPS engineering; modeling and analysis of Cyber-Physical Systems (CPS) and IoT; tools and methodologies to guarantee dependability-related properties, including real-time and mixed-criticality cohabitation; challenges posed for CPS design and verification by multi-core processors; smart factoring, industry 4.x; platforms for IoT - CPS. Deadline for submissions: April 30, 2016.

June 13-17 — ACM **Conference on Programming Language Design and Implementation** (PLDI'2016), Santa Barbara, California, USA. Topics include: all areas of programming language research, including the design, implementation, theory, and efficient use of languages.

☺ June 19-23 — ISC2016 - **Workshop on Exascale Multi/Many Core Computing Systems** (MuCoCoS'2016), Frankfurt, Germany. Topics include: multi/many core languages, system software and architectural solutions for extreme-scale systems towards Exascale; methods and tools for preparing applications for Exascale; adaptive run-time systems for extreme-scale; etc.

☺ June 28-30 **International Conference on Reliability, Safety and Security of Railway Systems** (RSSR'2016), Paris, France. Topics include: safety in development processes and safety management; combined approaches to safety and security; system and software safety analysis; formal modelling and verification techniques; system reliability; validation according to the standards; tool and model integration, toolchains; domain-specific languages and modelling frameworks; model reuse for reliability, safety and security; etc. Deadline for submissions: May 1, 2016 (posters).

July 04-08 **Software Technologies: Applications and Foundations** (STAF'2016), Vienna, Austria. Successor of the TOOLS federated event. Topics include: practical and foundational advances in software technology, including formal foundations of software technology, testing and formal analysis, graph transformations and model transformations, model driven engineering, and tools. Deadline for submissions: April 15, 2016 (doctoral symposium), May 2, 2016 (project showcases).

July 05-07 10th **International Conference on Tests And Proofs** (TAP'2016). Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the connection between proofs (and other static techniques) and testing (and other dynamic techniques); verification and analysis techniques combining proofs and tests; program proving with the aid of testing techniques; deductive techniques to support testing: generating testing inputs and oracles, supporting coverage criteria, and so on; program analysis techniques combining static and dynamic analysis; testing and runtime analysis of formal specifications; model-based testing and verification; using model checking to generate test cases; testing of verification tools and environments; applications of testing and proving to new domains, such as security, configuration management, and language-based techniques; case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc.

July 5-8 28th **Euromicro Conference on Real-Time Systems** (ECRTS 2016), Toulouse, France. Topics include: middleware; operating systems; runtime environments; virtualization and temporal isolation; software architecture; programming language & compiler support; component-based approaches; modelling and formal methods; probabilistic analysis; quality of service support; reliability, security and survivability; mixed-criticality systems; scheduling and schedulability analysis; worst-case execution time analysis; validation and verification techniques.

July 17-19 10th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2016), Shanghai, China. Topics include: theoretical aspects of software engineering, such as abstract interpretation, component-based systems, cyber-physical systems, distributed and concurrent systems, embedded and real-time systems, formal verification and program semantics, integration of formal methods, language design, model checking and theorem proving, object-oriented systems, run-time verification and monitoring, software architecture, software testing and quality assurance, software security and reliability, static analysis of programs, type systems and behavioural typing, tools exploiting theoretical results, etc.

July 17-23 28th **International Conference on Computer Aided Verification** (CAV'2016), Toronto, Ontario, Canada. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems, algorithms and tools for verifying models and implementations, program analysis and software verification, verification methods for parallel and concurrent systems, testing and run-time analysis based on verification technology, applications and case studies in verification, verification in industrial practice, formal models and methods for security, etc.

☺ July 18-22 30th **European Conference on Object-Oriented Programming** (ECOOP'2016), Rome, Italy. Topics include: theory, design, implementation, optimization, and analysis of programming languages that enable or enforce abstractions across various programming styles, from object-orientation to reactivity to spreadsheets; innovative and creative solutions to real problems; evaluations of existing solutions in ways that shed new insights; etc. Deadline for submissions: April 11 - May 10, 2016 (workshop papers), May 20, 2016 (doctoral symposium).

☺ July 17 1st **Workshop on Programming Models and Languages for Distributed Computing** (PMLDC'2016). Topics include: new approaches to distributed programming that provide efficient execution and the elimination of accidental nondeterminism resulting from concurrency and partial failure. Deadline for paper submissions: May 6, 2016.

☺ July 18 — 11th **Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems** (ICOOOLPS'2016). Topics include: techniques for the implementation and optimization of a wide range of languages including but not limited to object-oriented ones; implementation and optimization of fundamental languages features (from automatic memory management to zero-overhead metaprogramming); runtime systems technology; compilers (intermediate representations, offline and online optimizations, ...); empirical studies on language usage; resource-sensitive systems (real-time, low power, mobile, cloud); tooling support, debuggability and observability of languages as well as their implementations; etc. Deadline for submissions: April 11, 2016 (abstracts), April 15 (papers).

July 24-26 — 11th **International Joint Conference on Software Technologies** (ICSOFT'2016), Lisbon, Portugal. Topics include: all areas that are either related to new software paradigm trends or to mainstream software engineering and applications, such as software metrics, agile methodologies, risk management, quality control and assurance, software standards and certification, software and systems integration, software testing and maintenance, model-driven engineering, software and systems quality, software and information security, formal methods, programming languages, middleware technologies, parallel and high performance computing, etc. Deadline for submissions: April 21, 2016 (position papers), May 20, 2016 (Doctoral Consortium papers).

August 01-03 — IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2016), Vienna, Austria. Merger of SERE (International Conference on Software Security and Reliability) and QSIC (International Conference on Quality Software). Topics include: reliability, security, availability, and safety of software systems; software testing, verification and validation; metrics, measurements, and analysis; software vulnerabilities; formal methods; benchmark, tools, and empirical studies; etc. Includes IEEE International Workshop on Safety and Security in Cyber-Physical Systems (SSCPS), on Trustworthy Computing (TC), etc. Deadline for submissions: April 22, 2016 (workshop papers, student papers), April 29, 2016 (fast abstracts).

August 02-05 — 11th IEEE **International Conference on Global Software Engineering** (ICGSE'2016), Orange County, California, USA. Theme: "Software Bridging Distances Between People". Topics include: industrial offshoring and outsourcing experiences, lean and agile development, methods and processes, mining software repositories and software analytics, open source software communities, security and privacy, software evolution and maintenance, strategic issues in distributed development, tools and infrastructure support, etc.

Aug 31- Sep 02 — 42nd **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2016), Limassol, Cyprus. Topics include: information technology for software-intensive systems; embedded software engineering (ESE); model-based development, components and services (MOCS); software process and product improvement (SPPI); teaching, education and training for dependable embedded and cyberphysical systems (TET-DEC); cyber-physical systems (CPS).

September 12-14 — 5th **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SoMeT'2016), Larnaca, Cyprus. Topics include: state-of-art and new trends on software methodologies, tools and techniques; software methodologies and tools for robust, reliable, non-fragile software design; software development techniques for legacy systems; software evolution techniques; agile software and lean methods; formal methods for software design; software maintenance; software security tools and techniques; formal techniques for software representation, software testing and validation; object-oriented, aspect-oriented, component-based and generic programming, multi-agent technology; Model Driven Development (DVD), code centric to model centric software engineering; etc.

☺ September 26-29 — 21st **International Workshop on Formal Methods for Industrial Critical Systems** & 16th **International Workshop on Automated Verification of Critical Systems** (FMICS-AVoCS'2016), Pisa, Italy. Topics include: design, specification, refinement, code generation and testing of critical systems based on formal methods; methods, techniques and tools to support automated analysis, certification, debugging, learning, optimization and transformation of critical systems, in particular distributed, real-time systems and embedded systems; automated verification (model checking, theorem proving, SAT/SMT constraint solving, abstract interpretation, etc.) of critical systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); tools for the development of formal design descriptions; case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; impact of the adoption of formal methods on the development process and associated costs; application of formal methods in standardization and industrial forums. Deadline for submissions: April 18 (abstracts), April 25 (papers).

October 19-21	24th **International Conference on Real-Time Networks and Systems** (RTNS 2016), Brest, France. Topics include: real-time system design and analysis: task and message scheduling, modelling, verification, evaluation, model-driven development, worst-case execution time estimation, distributed systems, fault tolerance, quality of service, security; infrastructure and hardware for real-time systems: wired and wireless communication networks, fieldbuses, networked control systems, control/computing co-design, sensor networks, power-aware techniques; software technologies for real-time systems: compilers, programming languages, middleware and component-based technologies, operating systems, databases; applications: automotive, avionics, space, railways, telecommunications, process control, multimedia. Deadline for submissions: July 25, 2016.
☺ Oct 30-Nov 04	ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2016), Amsterdam, the Netherlands. Topics include: all aspects of software construction, at the intersection of programming, languages, systems, and software engineering. Deadline for submissions: April 1, 2016 (Onward! research papers and essays), June 10, 2016 (DLS - Dynamic Languages Symposium), June 17, 2016 (abstracts SLE - Software Language Engineering and GPCE - Generative Programming: Concepts & Experiences), June 24, 2016 (papers SLE and GPCE), June 30, 2016 (doctoral symposium), July 8, 2016 (posters).
Nov 29-Dec 2	37th **IEEE Real-Time Systems Symposium** (RTSS 2016), Porto, Portugal. Topics include: operating systems, networks, middleware, compilers, tools, scheduling, QoS support, resource management, testing and debugging, design and verification, modeling, WCET analysis, performance analysis, fault tolerance, security, power and thermal management, embedded platforms, and system experimentation and deployment experiences. Deadline for submissions: May 4, 2016.
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

21st International Conference on Reliable Software Technologies
Ada-Europe 2016
13-17 June 2016, Pisa, Italy

Advance Information

The 21st International Conference on Reliable Software Technologies (Ada-Europe 2016) will take place in Pisa, Italy. This conference is the latest in a series of annual international conferences started in the early 80's, under the auspices of, and organization by, Ada-Europe, the international organization that promotes the knowledge and use of Ada and Reliable Software in general into academia, research and industry.

Ada-Europe 2016 provides a unique opportunity for dialogue and collaboration between academics and industrial practitioners interesting in reliable software.

The conference will span a full week, including tutorials and a central three-day technical program with the latest advances in reliable software technologies and Ada. The core program features 3 keynote talks, 12 refereed scientific papers, 8 industrial presentations and one special session on Ada and Parallelism. The program of the conference is complemented with presentations from projects and the "ITS EASY Post Graduate School", co-located with the conference, and the workshop on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering". Half-day and full-day tutorials will be provided on Monday and Friday.

Week Overview

Monday	Tuesday	Wednesday	Thursday	Friday
Tutorials	Keynote Talk *Alan Burns*	Keynote Talk Valerio Giorgetta	Keynote Talk Marc Duranton	Tutorials & Workshop
	Regular session *Concurrency & Parallelism*	Industrial session *Use of Ada*	Industrial session *Reliable Software*	
	Vendor Session	Presentations *ITS EASY Post Graduate School & Projects*	Regular session *Program Correctness & Robustness*	
	Special session *Ada & Parallelism*	Regular session *Testing & Verification*	Regular session *Real-Time Systems*	
	Ada-Europe General Assembly Welcome cocktail	Conference banquet Best paper award	Best presentation award Closing session	

Keynote talks

Each day of the core program will be opened a keynote talk delivered by one the following eminent speakers:

- **Alan Burns**, University of York, UK, "*Why the Expressive Power of Languages such as Ada is needed for Future Cyber Physical Systems*"
- **Valerio Giorgetta**, Magneti Marelli, Italy, "*Challenges for the Automotive Platform of the Future*"
- **Marc Duranton**, CEA, France, "*The HiPEAC Vision*"

Tutorials

Bracketing the conference on Monday and Tuesday, the program includes eight tutorials:

- A Semi-formal Approach to Software Development, W. Bail, Monday morning
- Ada 2012 (Sub)types and Subprogram Contracts in Practice, J. Sparre-Andersen, Monday morning
- Software Test and Verification Techniques for Dependable Systems, W. Bail, Monday afternoon
- Towards Energy Awareness and Predictability in the Linux Kernel, J. Lelli, Monday afternoon
- Embedded ARM Programming with Ada 2012, P. Rogers, Monday full day
- Access Types and Memory Management in Ada 2012, J.P. Rosen, Friday monrning
- Using Gnoga for Desktop/Mobile GUI and Web development in Ada, J.P. Rosen, Friday afternoon
- Parallelism in Ada, C, Java and C#, Today and Tomorrow, B. Moore & S. Michell, Friday full day

Co-Located Workshop

The conference week features the third International Workshop on Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering (De-CPS 2016), following the success of the inaugural workshop in 2014 and of its second edition in Madrid in 2015.

The workshop will take place Friday, June 17th, from 09:30 to 17:30.

About the Venue

Pisa is located in Tuscany, close to the coast and just 80 km from Florence. It is a university city with a population of nearly 100,000. Once a Marine Republic, Pisa stretches along the shores of the Arno River, and occupies a place of honour amongst the most exclusive of art cities. Its glorious past offers authentic wonders to the tourist, and there is a lot more to see than just the leaning Tower of Pisa, its most popular 'product'.

June is full of events in Pisa, including in the conference week the Saint Patron's festivities (San Ranieri) with the **Luminara** on the night of June 16. This is definitely worth seeing! Book your hotel in advance - highly recommended as hotels will be in short supply.

The conference will be hosted at the Scuola Superiore Sant'Anna, located in the heart of Pisa, just a walk away from Campo dei Miracoli.

Organization

Program Committee

Industrial Committee

Sponsors

AdaCore
The GNAT Pro Company

ANSYS

ESTEREL
Technologies

Ellidiss
Software
TNI Europe Limited

PTC®
Developer Tools

RAPITA
SYSTEMS LTD

VECTOR
software

The conference is supported and sponsored by

Ada europe

In Cooperation with:

acm In-Cooperation

SIGAda SIGBED SIGPLAN

Ada Resource Association

http://www.ada-europe.org/conference2016

FOR IMMEDIATE RELEASE

Ada 2012 Language Standard Corrigendum Approved by ISO

Milestone marks smooth continuation of Ada language standardization process

EMBEDDED WORLD 2016, Nuremberg, Germany, February 23, 2016 – The Ada Resource Association (ARA) and Ada-Europe today announced that an update to the Ada 2012 language standard, formally known as Technical Corrigendum 1 to ISO/IEC 8652:2012, has been approved and officially published by the Geneva-based International Organization for Standardization (ISO). Comprising a variety of clarifications and minor corrections driven by implementation and user experience, the Corrigendum was developed under the auspices of Working Group ISO/IEC JTC1/SC22/WG9, in particular by WG9's Ada Rapporteur Group (ARG), and was issued on February 1, 2016. This work was supported in part by the ARA and Ada-Europe.

The publication of the Corrigendum highlights the steady and orderly evolution of the Ada programming language. New versions of the standard are published by ISO at roughly ten-year intervals. Between releases, the ARG reviews the standard for completeness, correctness, and unambiguity, and also considers and analyzes proposed updates ranging from minor wording changes to the addition of major new features. Especially in the case of new features, the ARG performs a careful analysis of the tradeoffs among the design choices, taking into account the requirements of all the stakeholders (existing Ada users, potential new users, compiler implementors, third-party tool providers, educators and researchers, etc.) This process has worked successfully since the language's inception more than thirty years ago, resulting in precisely defined standards that are issued in a timely fashion and that meet the evolving needs of the Ada community.

A consolidated Ada 2012 Language Reference Manual, consisting of the Ada 2012 standard as updated by changes from the Corrigendum, is available online: www.ada-auth.org/standards/ada12_w_tc1.html/.

"In this phase of the language standardization process, the focus is on attention to detail, so the Corrigendum has 'fine-tuned' the wording to make sure that the standard is correct," said Dr. Joyce Tokar, WG9 Convenor. "It has also enhanced the control provided by contract-based programming, so the Ada programmer can not only specify the preconditions and predicates that apply to inputs, but also identify which particular exceptions should be raised when a precondition or predicate fails. The preconditions and predicates can thus fully specify an API's requirements, and the consequences of failure when these requirements are not met. The Corrigendum represents an important contribution to the Ada community."

About Ada 2012

Ada 2012 has brought significant enhancements to Ada, most notably in the area of "contract-based programming." Features here include the ability to specify preconditions and postconditions for subprograms, and invariants for private (encapsulated) types. These take the form of Boolean expressions that can be interpreted (under programmer control) as run-time conditions to be checked. The contract-based programming features fit in smoothly with Ada's Object-Oriented Programming model, and support the type substitutability guidance supplied in the Object-Oriented Technologies and Related Techniques Supplement (DO-332) to the avionics software safety standard DO-178C / ED-12C.

Other Ada 2012 improvements include enhancements to the containers library, increased expressiveness through features such as conditional expressions and more powerful iterators, and support for multicore platforms (task affinities, and the extension of the Ravenscar profile – standardized in Ada 2005 as an efficient and predictable tasking subset for high-integrity real-time systems – to multiprocessor and multicore environments).

A technical summary of Ada 2012, together with an explanation of the language's benefits and a set of links to further information, is available at www.ada2012.org, a website maintained by the Ada Resource Association.

About the Ada Resource Association

The Ada Resource Association (ARA) is a non-profit organization chartered to support the continued evolution of the Ada language and its infrastructure, to serve as a source of information about Ada and its usage, and to promote Ada as a language for effective software engineering. To these ends, the ARA maintains the Ada Information Clearinghouse website www.adaic.org and has provided funding for the development and maintenance of the Ada language standard and the Ada Conformance Assessment Test Suite. For information about the ARA, including sponsorship opportunities, please visit www.adaresource.com. The ARA is headquartered in Oakton, VA (US).

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of the Ada programming language in academia, research and industry in Europe. Its flagship event is the annual international conference on reliable software technologies, a high-quality technical and scientific event that has been successfully running in the current format for the last 20 years. Ada-Europe has member organizations all over the continent, in Belgium, Denmark, France, Germany, Spain, Sweden, and Switzerland, as well as individual members in many other countries.

For information about Ada-Europe, its charter, activities and sponsors, please visit: www.ada-europe.org. Ada-Europe is headquartered in Brussels, Belgium.

Organization Contacts

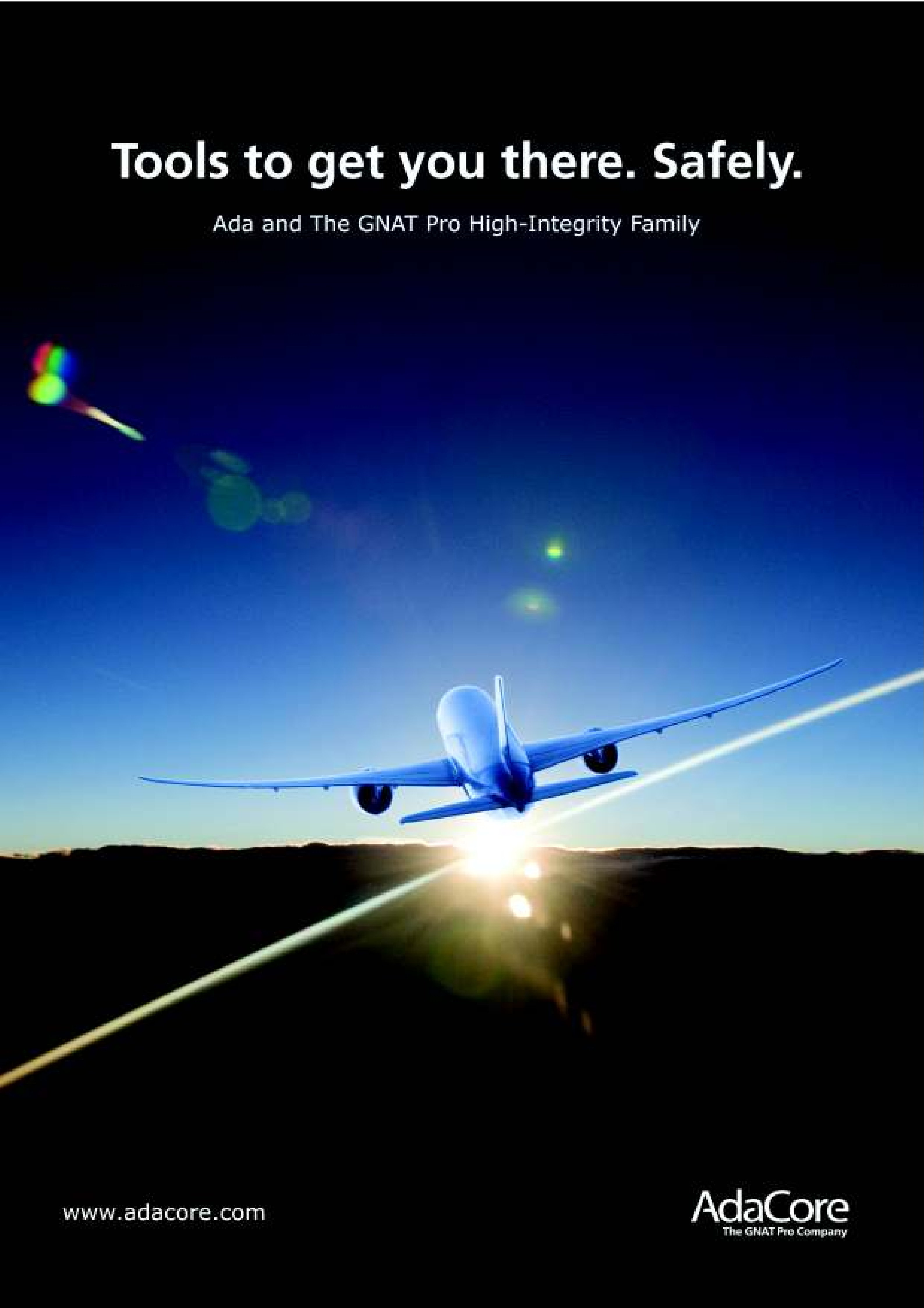
Ada Resource Association
Ben Brosgol, ARA President
brosgol@adacore.com

Ada-Europe
Tullio Vardanega, Ada-Europe President
president@ada-europe.org

Press Contacts

Ada Resource Association
Jenna Beaucage
Rainier Communications
Tel: +1-508-475-0025 x124
jbeaucage@rainierco.com

Ada-Europe
Dirk Craeynest, Ada-Europe Vice-president
c/o KU Leuven, Department of Computer Science
dirk.craeynest@cs.kuleuven.be


Tools to get you there. Safely.
Ada and The GNAT Pro High-Integrity Family
www.adacore.com
AdaCore
The GNAT Pro Company

Update for Ada 2012

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 rationale for the changes made to Ada 2012 as a consequence of a review carried out in 2015.

Keywords: rationale, Ada 2012, corrigendum.

1 Introduction

The first version of Ada (Ada 83) was developed by a team led by the late Jean Ichbiah and funded by the USDoD. The development of Ada 95 was done under the leadership of Tucker Taft and also funded by the USDoD. Then came Ada 2005 and Ada 2012 [1, 2] which were developed on a more modest scale and largely done by voluntary effort with support from within the industry itself by bodies such as the Ada Resource Association and Ada-Europe.

The Ada Rapporteur Group (ARG) is a team of experts nominated by the national bodies represented on WG9 (ISO/IEC JTC 1/SC22/WG9, the working group for Ada) and the two liaison organizations, ACM SIGAda and Ada-Europe. In the case of Ada 2005, the ARG was originally led by Erhard Plödereder and then by Pascal Leroy. For Ada 2012, it was led by Ed Schonberg. Since the Ada 2012 standard was issued the ARG has been led by Jeff Cousins. The editor, who at the end of the day actually writes the words of the standard, continues to be the indefatigable Randy Brukardt and the convenor of WG9 is Joyce Tokar.

The changes made to Ada 2012 by the update are presented in the same areas as in the Ada 2012 Rationale [3] to ease any comparison. In each area the relevant Ada Issues (AIs) are listed and this is then followed by a brief discussion. It will be observed that many issues concern corner cases which will not be of much interest to the average user. Moreover, the changes are usually corrections to the descriptive text. But just occasionally we get the excitement of a new aspect, a new type or subprogram and even a new bit of syntax!

Note that some Ada Issues were mentioned in the Postscript section of the Ada 2012 Rationale published after the standard was approved (and in Programming in Ada 2012 [4] by the author). They are given here for completeness and are marked with *. But note that they might have changed again since the Rationale was written.

The updated version of Ada 2012 was approved by WG 9 in June 2015, by SC 22 in December 2015, and published by ISO in February 2016. It is formally ISO/IEC 8652:2012/Cor 1:2016, entitled Technical Corrigendum 1 for Ada 2012, or TC1 for short.

2 Contracts and aspects

This is perhaps the most exciting area of Ada 2012 and covers matters such as the new notation for aspects which replaces many uses of pragmas and also the introduction of contracts. Contracts include pre- and postconditions, type invariants, and subtype predicates.

The following Ada issues cover this area:

32 Questions on 'Old

41 Type_Invariant'Class for interface types

42 Type invariant checking rules

44* Calling visible functions from type invariant expressions

45* Pre- and Postconditions are allowed for generic subprograms

49 Invariants need to be checked on the initialization of deferred constants

54* Aspect Predicate_Failure

68 Predicates and the current instance of a subtype

71* Order of evaluation when multiple predicates apply

77 Has_Same_Storage on objects of size zero

96 The exception raised when a subtype conversion fails a predicate check

99 Wording problems with predicates

104 Overriding an aspect is undefined

105 Pre and Post are not allowed on any subprogram completion

113 Class-wide preconditions and statically bound calls

116 Private types and predicates

131 Inherited Pre'Class when unspecified on initial subprogram

133 Type invariants and default initialized objects

149 Type invariants are checked for functions returning access-to-type

150 Class-wide type invariants and statically bound calls

154 Aspects of library units

These changes can be grouped as follows.

There are a number of clarifications on pre- and postconditions in general (45, 105), some additional rules on the use of the aspect Old (32), and a trivial omission on the aspect Has_Same_Storage (77).

There are also clarifications regarding types and the use of class wide conditions and invariants (113, 150).

A number of additional rules are required describing the places where type invariants are checked (41, 42, 44, 49, 133, 149).

A new aspect Predicate_Failure and rules concerning checking order are added (54, 71, 96). A number of clarifications to the rules for predicates are added (68, 99, 116).

Finally, there are minor clarifications regarding aspects in general (104, 154).

===

AI-45 notes that pre-and postconditions (both specific and classwide) are allowed on generic programs but they are not allowed on instances of generic subprograms. This avoids awkward maintenance problems that might arise if such conditions were allowed on both a generic and an instance. On a similar matter, it was the intent that pre- and postconditions should always be visible to the user and so go on specifications and not bodies which are completions. However, the wording forgot to mention the new expression functions which as well as standing alone can also be completions (see the example of Non_Zero in AI-49 below). Such expression functions and null procedures acting as completions cannot have pre- and postconditions either (AI-105).

The attribute 'Old which is used in a postcondition to indicate the initial value of a parameter (or maybe a global) is a bit tricky. AI-32 adds some more (fairly obvious) detail such as that X and X'Old have the same type even if it is anonymous.

A trivial point about the aspect Has_Same_Storage which is useful in preconditions is that the expression X'Has_Same_Storage(Y) returns False if X or Y or both occupy zero bits. (AI-77).

AI-131 concerns Pre'Class. If the initial definition of a subprogram does not specify Pre'Class then the corresponding subprograms of derived types just inherit True as one would expect. It also notes that one cannot specify Pre'Class for an overriding subprogram of a type unless Pre'Class is specified for some ancestor. This is because it would be ored with True and thus have no effect.

AI-113 and AI-150 address a problem with Pre'Class, Post'Class and Type_Invariant'Class and parameters. Typically we might have something like

```
procedure P(S: in out T; ...)
  with Pre'Class => expression involving S
```

where T is a tagged type. The procedure P can be called with a statically bound parameter or indeed with a dynamically bound classwide parameter. The wording is clarified that the actual types of the actual parameters are always used. The reader is referred to the text of the Ada Issues which have a large example which should be helpful.

It is well-known that type invariants are not intended to be foolproof but to be helpful in catching many flaws (unlike pre- and postconditions which are meant to be perfect). As time has passed more little quirks have been found regarding when type invariants should or should not be checked. Remember that invariants apply to private types. When we have a full view of the type (as in a subprogram in the body of the package declaring the type) then we can change an object of the type temporarily to a state where the invariant does not apply (this is often necessary for intermediate stages in manipulation). However, when we leave the full view by for example returning from a subprogram in the package body then checks are applied on parameters and so on.

AI-44 states that type invariants are not checked on **in** parameters of functions but are checked on **in** parameters of procedures. This is necessary to avoid infinite recursion which would arise if an invariant itself calls a function with a parameter of the type. Moreover, a classwide invariant could not be used at all without this modification.

AI-49 adds that invariants need to be checked on the initialization of deferred constants (other initializations were already covered). An example is as follows

```
package R is
  type T is private
    with Type_Invariant => Non_Zero(T);
  function Non_Zero(X: T) return Boolean;
  Zero: constant T;
private
  type T is new Integer;
  function Non_Zero(X: T) return Boolean is
    (X /= 0);
  Zero: constant T := 0;
end R;
```

Currently, this is not caught but the declaration of Zero should raise Assertion_Error. Note the neat use of an expression function for the completion of Non_Zero.

AI-133 concerns default initializations and type invariants. Remember that initialization by default occurs in various circumstances. If a record type has components with an initial value as in

```
type R is
  record
    C: Integer := 99;
  end record;
```

then when we declare an object of type R without any initialization thus

```
X: R;
```

then we are assured that X.C will have the value 99. Moreover, in Ada 2012 we can give default values for other types by the new aspects Default_Value and Default_Component_Value (these are discussed in Section 3). But we cannot give defaults for existing predefined types such as Integer since we can only give the default when the type is declared.

The question arises as to whether or not the rule about default initializations being checked for the type invariant applies to an object declared inside the package body. Normally, of course, we can do what we like inside the body so we might expect such initializations not to be checked. However, since the default initialization mechanism is the same for clients outside the body as it is for the writer of the body, it was felt that it would be better if they were all checked for consistency and simplicity. An error might thus be detected earlier.

But there is an exception. If the partial view of the type has unknown discriminants then the user cannot declare objects of the type anyway and so no check could be performed externally. For uniformity no check is performed internally either.

AI-149 adds checks for functions whose return type is an access type with a designated type having a part of the type concerned (parameters were already covered). Further similar checks are added by AI-42 concerning type extensions. It is all a bit subtle and maybe other problems are lurking. One is tempted to quote a wee bonny poem of Sir Walter Scott

O what a tangled web we weave,
When first we practise to deceive!

A more exciting change (AI-41) is to allow Type_Invariant'Class on interface types. This can be used to ensure that any type derived from the interface satisfies certain properties. The following example is given

```
type Window is interface
  with Type_Invariant'Class =>
    Window.Width * Window.Height = 100;
function Width(W: Window) return Integer is abstract;
function Height(W: Window) return Integer is abstract;
```

This ensures that any type derived from the interface Window will provide functions giving the height and width of the windows such that their product is exactly 100. This is a display window of 100 pixels (poor quality I fear).

If we derive a type from several such interfaces then the Type_Invariant'Class is of course the conjunction of the individual invariants.

A new aspect Predicate_Failure is introduced by AI-54 (with a wording correction in AI-96). A related issue concerning the order of evaluation of predicates is discussed in AI-71. These were discussed in detail in the Ada 2012 Rationale and in Section 16.5 of Programming in Ada 2012. A short extract will suffice here.

The expected type of the expression defined by the aspect Predicate_Failure is String and gives the message to be associated with a failure. So we can write

```
subtype Open_File_Type is File_Type
  with
    Dynamic_Predicate => Is_Open(Open_File_Type),
    Predicate_Failure => "File not open";
```

If the predicate fails then Assertion_Error is raised with the message "File not open".

We can also use a raise expression (see AI-22 in the next section) and thereby ensure that a more appropriate exception is raised. If we write

```
Predicate_Failure =>
   raise Status_Error with "File not open";
```

then Status_Error is raised rather than Assertion_Error with the given message. We could of course explicitly mention Assertion_Error thus by writing

```
Predicate_Failure =>
         raise Assertion_Error with "A message";
```

Finally, we could omit any message and just write

```
Predicate_Failure => raise Status_Error;
```

in which case the message is null.

A related issue is discussed in AI-71. If several predicates apply to a subtype which has been declared by a refined sequence then the predicates are evaluated in the order in which they occur. This is especially important if different exceptions are specified by the use of Predicate_Failure since without this rule the wrong exception might be raised. The same applies to a combination of predicates, null exclusions and old-fashioned subtypes.

There are a number of minor wording omissions and corrections with the description of predicates. AI-116 notes that one cannot give a specification of an aspect such as Dynamic_Predicate to both the full view and private view of a type (there is such a rule for most aspects but it was forgotten for predicates). A subtle point covered by AI-68 is that within a predicate the current instance (such as T in the expression Non_Zero(T) in the example of AI-49) acts like a value rather than an object. (This prevents certain undesirable uses of T such as applying a number of object attributes.) AI-99 confirms that **not** is an allowed operation in a Static_Predicate and also corrects some wording in relation to tasks and protected objects.

An interesting topic is addressed by AI-154. It may be recalled that rather than writing

```
package P is
  pragma Pure(P);
...
```

we can instead write

```
package P
  with Pure is
...
```

or more pedantically even

```
package P
  with Pure => True is
...
```

or really foolishly

```
package P
  with Pure => False is
...
```

Using a pragma rather than an aspect specification is the preferred style for library unit pragmas. But there might be some benefit in using an aspect specification since one could change the state of a whole group of packages in one blow by a structure such as

```
package State_Control is
  Purity: constant Boolean := True;
  ...
end State_Control;
...
with State_Control;
package P
  with Pure  => State_Control.Purity is
  ...
end P;
...      -- and so on for several packages
```

We could then make Purity false for adding some debugging material during development and then set it to true for final production.

Observe that AI-154 illustrates that one could be silly enough to try to write

```
package P with Pure => Purity is
  ...
  Purity: constant Boolean := True;          -- illegal
end P;
```

which raises the question of when is the wretched static expression Purity evaluated. The answer is immediately of course just like the pragma and so the example is illegal because Purity has not yet been elaborated. Note that aspects such as type invariants in

```
type Stack is private
  with Type_Invariant => Is_Unduplicated(Stack);
```

require the elaboration to be deferred. But this cannot be done with Pure because it controls the application of some Legality Rules.

The final AI-104 regarding aspects simply replaces a confusing sentence by a (confusing?) user note. Aspects such as Constant_Indexing can be inherited; the aspects themselves cannot be redefined but the functions they denote can be modified by overriding or by overloading.

3 Expressions

The introduction of contracts triggered the need for more flexible forms of expressions in Ada 2012. These are conditional expressions (if and case), quantified expressions, and expression functions. In addition membership tests were made much more flexible.

The following Ada issues cover this area:

22* Raise expressions

39* Ambiguity in syntax for membership expression removed

40 Resolving the selected_expression of a case_expression

50 Conformance of quantified expressions

62 Raise expression with failing string function

84 Box expressions in array aggregates

100 A qualified expression makes a predicate check

103 Expression functions that are completions in package specifications

141 Add raise expression to Introduction

147 Expression functions and null procedures can be declared in a protected_body

152 Eliminate ambiguities in raise expression and derived type syntax

157 Missing rules for expression functions

158 Definition of quantified expressions

These changes can be grouped as follows.

The most important change in this update is perhaps the introduction of raise expressions (22, 62, 141, 152); it is convenient to discuss a change to the syntax for membership test at the same time (39).

A number of changes relate to expression functions (103, 147, 157). There are also some clarifications regarding quantified expressions (50, 158).

There are miscellaneous changes to qualified expressions (100), case expressions (40), and array aggregates (84).

===

The introduction of raise expressions by AI-22 was deemed important enough to be mentioned in the Introduction to the revised RM. It was discussed in some detail in the postscript section of the Ada 2012 Rationale which was written after the Ada 2012 standard was published. However, the discussion there needs updating since the syntax rules have been modified as a consequence of ambiguities mentioned in AI-39 and AI-152. So here is a more integrated description.

The raise expression, is added by analogy with if statements and the raise statement. Thus as well as

```
if X < Y then
  Z := +1;
elsif X > Y then
  Z := –1;
else
  raise Error;
end if;
```

we can also write

```
Z := (if X<Y then 1 elsif X>Y then –1 else raise Error);
```

The syntax for raise expression is now as follows

raise_expression ::=
  raise *exception_*name [**with** *string_*simple_expression]

Note that unlike in a raise statement, the string expression has to be a simple_expression rather than an expression in order to avoid ambiguities involving logical operations.

A raise expression is a new form of relation (as will be seen in the syntax in a moment) and has the same precedence and so will need to be in parentheses in some contexts. But as illustrated above it does not need parentheses when used in a conditional expression which itself will have parentheses.

Raise expressions will be found useful with pre- and postconditions. Thus if we have

```
procedure Push(S: in out Stack; X: in Item)
  with
    Pre => not Is_Full(S);
```

and the precondition is false then Assertion _Error is raised. But we can now alternatively write

```
procedure Push(S: in out Stack; X: in Item)
  with
    Pre => not Is_Full(S) or else raise Stack_Error;
```

and of course we can also add a message thus

```
    Pre => not Is_Full(S) or else
      raise Stack_Error with "wretched stack is full";
```

Another issue concerns what happens if the string expression in a raise expression (or indeed in a raise statement) itself raises an exception; it could be a function call which returns the string. The answer is that the one caused by the string expression is propagated instead of the one given in the raise expression or statement (AI-62).

On a closely related topic the syntax for membership tests has been found to cause ambiguities (AI-39).

Thus

```
A in B and C
```

could be interpreted as either of the following

```
(A in B) and C              -- or
A in (B and C)
```

This is cured by changing the syntax for relation to

relation ::=
 simple_expression [relational_operator simple_expression]
 | *tested_*simple_expression [**not**] **in** membership_choice_list
 | raise_expression

and changing membership choice to use simple_expression as well thereby requiring that we have to insert parentheses in the above example

membership_choice ::=
  *choice_*simple_expression | range | subtype_mark

Thus a membership_choice no longer uses a choice_expression. However, the form choice_expression is still used in discrete_choice.

When first written, AI-22 showed the syntax for a raise expression using *string_*expression just as in raise statement. However, this caused ambiguities as mentioned earlier so it was changed to *string_*simple_expression by AI-152. Curiously enough it was also necessary to change the syntax of digits constraint and delta constraint to use simple expression as well. The AI has the following bizarre example

```
Atomic: String := "Gotcha";
type Fun is new My_Decimal_Type digits
    raise TBD_Error with Atomic;
```

This could be parsed as either

```
type Fun is new My_Decimal_Type digits
    (raise TBD_Error with Atomic);
```

or

```
type Fun is new My_Decimal_Type digits
    (raise TBD_Error) with Atomic;
```

So we now have

digits_constraint ::=
  digits *static_*simple_expression [range_constraint]
delta_constraint ::=
  delta *static_*simple_expression [range_constraint]

It seems cruel to have to change delta constraint which one might have thought was peacefully buried in Annex J for obsolescent features.

These potential ambiguities are unlikely to impact the normal user. If the compiler complains then the judicious insertion of some parentheses will undoubtedly cure the problem.

Expression functions were added in Ada 2012. Remember that an expression function takes the form

```
function F ( ... ) return T is
  (expression of subtype T);
```

A good example was given earlier thus

```
function Non_Zero(X: T) return Boolean is
  (X /= 0);
```

Remember that such functions can act as a complete function or as a completion of a traditional function specification.

A number of points were overlooked in the definition of Ada 2012. One has already been mentioned namely that function expressions acting as completions cannot have pre- and postconditions (see AI-105 in Section 2).

Another point is that expression functions and indeed null procedures can be used in the body of a protected type as a completion of a protected function (AI-147). This requires a modification to the syntax which becomes

```
protected_operation_item ::=
          subprogram_declaration
        | subprogram_body
        | null_procedure_declaration
        | expression_function_declaration
        | entry_body
        | aspect_clause
```

An interesting situation arises if the result expression of an expression function can be written as an aggregate as for example

```
function Conjugate (C: Complex) return Complex is
   ((C.Rl, –C.Im));                    -- double parens
```

Remember that the conjugate of a complex number *C* has the same real part but the imaginary part changes sign so that the conjugate point in the Argand plane is the reflection of *C* in the real axis.

The original rules say that the expression of an expression function is simply an expression in parentheses. However, this is ugly if the expression already has parentheses as occurs with an aggregate as above. Now Ada dislikes double parentheses so we have rules for if expressions that they have to be in parentheses unless the context already supplies parentheses as in the case of a subprogram call with a single parameter. Consequently, AI-157 concludes that the second lot of parentheses are unnecessary so we can just write

```
function Conjugate (C: Complex) return Complex is
   (C.Rl, –C.Im);                      -- single parens
```

As a result the syntax is revised to

```
expression_function_declaration ::=
        [overriding _indicator]
        function_specification is
                (expression)
                [aspect_specification] ;
      | [overriding _indicator]
        function_specification is
                aggregate
                [aspect_specification] ;
```

The above example shows a record aggregate. The same applies to array aggregates in parentheses. But of course if the result is given as a string then the parentheses are necessary. So we can have either of

```
function Piggy return String is
   ('P', 'I', 'G');
function Piggy return String is
   ("PIG");
```

There are also changes to the freezing rules which will probably leave the reader cold. These are in AI-157 and AI-103. A simple one is that expression functions acting as completions only freeze the expression and nothing else and null procedures never freeze anything.

There are a couple of minor points regarding quantified expressions.

AI-158 clarifies the result of a quantified expression where the array concerned has zero elements. In the case of the existential qualifier **for some**, the result is False whereas in the case of the universal qualifier **for all**, the result is True. So consider

```
B := (for all K in A'Range => A(K) = 0);
```

which assigns true to B if every element of A is zero. If A doesn't have any elements then the result is still true. Similarly

```
B := (for some K in A'Range => A(K) = 0);
```

assigns true to B provided at least one element of A is zero. If A doesn't have any elements then the answer is false. This all seems pretty obvious but the wording was deemed to require clarification for those not having a Fields Medal in mathematics.

Another quirk is discussed by AI-50 which is concerned with conformance. Remember that the parameters in the specification and body of a subprogram have to conform. The introduction of quantified expressions means that such an expression could occur as the default value in a subprogram specification; thus using the example above we might have

```
procedure P(B : Boolean :=
        (for all K in A'Range => A(K) = 0));
```

The corresponding text in the procedure body has to conform and additional rules are required to ensure this. The new thing is that quantified expressions introduce declarations such as that of K in the parameter list and we have to say that these two (technically different) declarations are the same in specification and body and specifically that the two defining identifiers are the same and are used in the same way.

A minor omission concerns qualified expressions (AI-100). Remember the difference between a qualified expression and a conversion. Qualification (which takes a quote) just states that an expression has the given (sub)type and is often used for resolving ambiguities. Conversion (which does not have a quote) actually changes the type (if necessary). Qualification also checks any relevant subtype properties. But on the addition of subtype predicates, although it was added that they were checked on type conversions, it was forgotten to add that any subtype predicates should also be checked on qualification.

A very minor omission is covered by AI-40 which says that case expressions and case statements resolve in exactly the same way – that is have the same rules for type matching.

Finally, AI-84 concerns the use of the box notation <> in array aggregates. This was added in Ada 2005 and indicates that a component takes its default value which is the same as the default value for a stand-alone object.

However, Ada 2012 added the aspects Default_Value and Default_Component_Value. So we might write

```
type My_Integer is new Integer
        with Default_Value =>888;
```

```
type My_Array is array (Integer range <>) of My_Integer
      with Default_Component_Value => 777;
```

If we declare

```
X: My_Array(1 .. 10);
```

then the value of X(1) will be 777 of course using the aspect Default_Component_Value.

But if we write

```
X: My_Array(1..10) := (others => <>);
```

then very surprisingly X(1) was 888 rather than 777. The rules for <> were not updated to note that if the Default_Component_Value has been given then that applies rather than the stand-alone value. This is put right by AI-84 so that the value is now 777 in both cases.

4 Structure and visibility

One of the most dramatic changes in Ada 2012 concerns subprogram parameters and is that functions can have parameters of all modes. Other areas covered here include incomplete types and discriminants.

The following Ada Issues cover this area:

65	Descendants of incomplete views
74	View conversions and out parameters passed by copy
94	An access definition should be a declarative region
95	Generic formal types and constrained partial views
97	Tag of the return object of a simple return expression
101	Incompatibility of hidden untagged record equality
109	Representation of untagged derived types.
132	Freezing of renames-as-body
137	Incomplete views and access to class wide types

These changes can be grouped as follows.

A number of issues concern views. There are clarifications of incomplete views (65, 137) and omissions concerning view conversions (74) and constrained views (95).

An amusing issue concerns the definition of a declarative region (94).

Miscellaneous issues concern renaming (132), untagged record equality (101), untagged derived types (109), and the tag of return objects (97).

===

An odd situation is discussed by AI-65. A type T3 can be a descendant of T1 but yet inherits no characteristics of T1 because of an intervening type T2. Consider

```
package P is
  type T1 is private;           -- partial view
...C: constant: T1;
private
  type T1 is new Integer;       -- complete view
  C: constant := 37;     -- my favourite number
end P;
with P;
package Q is
  type T2 is new P.T1;
end Q;
with Q;
package P.Child is
  type T3 is new Q.T2;
private
                 -- what can we do with T3 here?
end P.Child;
```

In this example T3 is derived from T2 and T2 is derived from T1. The fact that T2 is derived from Integer is not visible to the declaration of T3. Nevertheless the conversion rules allow a value of type T1 to be converted to T3 in the private part of the child package. But the fact that T3 is an integer type is not visible.

We say that T3 is effectively a descendant of an incomplete view of T1. (Note "effectively"; it's not technically an incomplete view but behaves in some ways as if it were.) So we can convert C but not 73 to type T3 in the private part of C.Child.

```
...X: T3 := T3(P.C)        -- OK
   Y: T3 := T3(73);        -- No, T3 is not visibly numeric!
```

It was meant to be like this in Ada 95; Ada 2005 meddled with it and Ada 2012 made a confusing "improvement". Hopefully the clarifications made now will be the end of the story.

It is helpful to remember the distinction between a partial view and an incomplete view.

> A partial view is the view given by a private type declaration in contrast to the full view given by the full declaration in the private part. As in type T1 above.
>
> An incomplete view is the view given by an incomplete declaration such as occurs with access types. Thus

```
type Cell;                      -- incomplete view
type Link is access Cell:
type Cell is                    -- completion
  record
    Next: Link;
    ...
  end record;
```

The use of the concept of incomplete views was much extended in Ada 2005 by the introduction of the limited with clause. It was extended again in Ada 2012 by allowing incomplete types to be completed by types other than access types and allowing incomplete views as parameters.

There are many rules concerning access types that designate incomplete views. AI-137 clarifies that they also apply to access to class wide types.

AI-95 concerns an omission/confusion with regard to generic untagged formal types and partial views. Briefly, within a generic body we assume the worst as to whether or not a formal subtype has a constrained partial view. In particular we assume that untagged formal private and derived types do indeed have a constrained partial view.

As Ada has grown there have been further lexical amusements such as functions returning access to functions. Thus we can now have

```
type T is
  access function( ... ) return
    access function( ... ) return
      access function( ... ) return ...
```

ad infinitum. To be more specific the rules seem to prohibit

```
type T is
  access function(A: Integer) return
    access function(A: Float) return Boolean;
```

because here we have two instances of A in the declarative region for T. There is no real reason why this should not be permitted so the definition of declarative region is extended to include an access definition (AI-94).

A somewhat different topic is addressed by AI-74 and concerns parameters of mode **out** which have always been the source of troubles. The basic problem is that such parameters can become undefined. Consider this simple procedure to find the two roots of a quadratic equation

```
procedure Quadratic(A, B, C: in Real;
      Root_1, Root_2: out Real; OK: out Boolean) is
  D: constant Real := B**2 – 4.0*A*C;
begin
  if D < 0.0 or A = 0.0 then
    OK := False; return;
  end if;
  Root_1 := (–B + Sqrt(D)) / (2.0*A);
  Root_2 := (–B – Sqrt(D)) / (2.0*A);
  OK := True;
end Quadratic;
```

If the equation has complex roots then no values are assigned to Root_1 and Root_2 so they are likely to contain rubbish. So if we call Quadratic thus

```
Quadratic(AA, BB, CC, R1, R2, State);
```

then because of the copy in and out rules for parameters of elementary types, the variables R1 and R2 which might have had respectable values will now contain rubbish.

Of course if we had made the parameters Root_1 and Root_2 of mode **in out** then the original values of R1 and R2 would have been retained if no assignments were made to Root_1 and Root_2.

However, if we had been wise and used the Default_Value aspect introduced in Ada 2012 thus

```
type Real is new Float
  with Default_Value := 0.0;
```

then the behaviour is different. In this case Root_1 and Root_2 will behave essentially as if they were of mode **in out** and will remain unchanged. Note carefully that they will not take the default values of 0.0 and so the existing values of R1 and R2 will not be disturbed. Of course if we had declared a local variable R_Temp of type Real then it would take the initial value of 0.0.

This technique of initially copying in parameters of mode **out** has existed in Ada for access types since Ada 83. Remember that access types always have a default initial value of **null** and so this copying in behaviour is identical. Incidentally, this copying in is done "in the raw" without making any subtype checks such as range constraints; again this follows the behaviour of access types. Note also that the whole purpose of an **out** parameter is to give it some value without concern for the original value of the actual parameter and so gratuitously checking the original value of the actual could be irritating if it raised an exception. Another point is that the default value applies to the type and not to the subtype.

However, do remember that we cannot give a default value to the predefined types such as Float so this is a good reason for declaring our own types.

Other problems arise when an actual parameter is a view conversion and this is the real topic of AI-74. Consider the following simple example

```
procedure Inc(X: in out Integer) is
begin
  X := X + 1;
end Inc;
...
F: Float;
...
F := 3.14;
Inc(Integer(F));
```

Remember that the behaviour is that the value of F is converted to type Integer (and thus becomes 3) and this is the initial value of the parameter X which is then incremented to 4 and finally converted to 4.0 and copied back into F. This is as in Ada 83.

But problems arise if the parameter is an **out** parameter and not an **in out** parameter. Consider

```
procedure P(X: out My_Integer) is ...
...
Y: Long_Float;
...
P(My_Integer(Y));
```

Now suppose we have given Default_Value for My_Integer. An important goal of Default_Value is to ensure that junk values do not arise. This is done by treating **out** parameters essentially as **in out** parameters as illustrated by Quadratic. But now we are in trouble because we are unlikely to be able to convert the giant floating value Y to the type My_Integer.

This problem is overcome by saying that if the aspect Default_Value is given for the type of the formal parameter then there must be an ancestor of both the target type and the operand type of the view conversion and the operand type itself must also have the aspect Default_Value. The conversion is then bound to work.

AI-132 concerns expression functions and freezing again (see the brief mention of AI-103 in the previous Section). If we have an expression function such as

```
function F(...) return T is
  (expression of subtype T);
```

then it can occur in a renaming as body thus

```
function G( ... ) return T renames F;
```

This AI points out that this renaming freezes the expression of the expression function F.

The redefining of equality has always been a bother. Originally there were different rules for composition of tagged and untagged types. The difference was removed in Ada 2012 in order to make composition more uniform. However, a quirk in the rules meant that a hidden definition of equality for an untagged record type as in

```
package P is
  type PT is private;
private
  type PT is record ... end record;          -- untagged
  function "=" (L, R: PT) return Boolean;
end P;
```

was not permitted. This was a mistake and accordingly this restriction is removed by AI-101.

There are omissions regarding aspect specifications and derived types. One of the advantages of the introduction of aspect specifications is that they occur with the entity to which they apply. This means that the traditional linear elaboration does not always apply because the aspect might refer to things that have not yet been declared. AI-109 clarifies the situation with regard to the freezing of the representation of untagged types.

Finally, AI-97 addresses a minor error in the description of the tag of an object in a return statement. The introduction of the extended return statement where we have

```
return R: T do
  ...
end return;
```

needed clarification because T might not be identical to the return type given in the function specification (it might be a subtype; perhaps the function has an indefinite type and the return is definite, perhaps classwide and specific, and so on). So the rules were rewritten to cover the extended return. Unfortunately the rules were written in a way that was incorrect for an old-fashioned return statement. This is now put right.

5 Tasking and real-time facilities

The major topic in this area is providing features for multiprocessors and increasing control for scheduling.

The following Ada Issues cover this area:

1 Independence and representation clauses for atomic objects

33* Sets of CPUs when defining dispatching domains

48 Default behavior of tasks on a multiprocessor with a specified dispatching policy

51 The Priority aspect can be specified when Attach_ Handler is specified

52 Implicit objects are considered overlapping

55 All properties of a usage profile are defined by pragmas

73 Synchronous Barriers are not allowed with Ravenscar

81 Real-time aspects need to specify when they are evaluated

82 Definition of "dispatching domain"

90 Pre- and postconditions and requeues

98 Problematic examples for ATC

107 A prefixed view of a By_Protected_Procedure interface has convention protected

114 Overlapping objects designated by access parameters are not thread-safe

117 Restriction No_Tasks_Unassigned_To_CPU

129 Make protected objects more protecting

These changes can be grouped as follows.

First there are some clarifications and additions to dispatching domains which were added in 2012 (33, 48, 82). There are also some changes to the definition of the Ravenscar profile (55, 73, 117) and clarifications to some real-time aspects (51, 81). These are all in the Real-Time Systems annex (D).

The examples of the use of ATC (asynchronous transfer of control) need further explanation (98).

The question of being thread-safe in the face of overlapping objects has always been tricky and the text in the opening part of Annex A is modified (52, 114).

There are some improvements to the ability to control concurrent access in the core language (107, 129).

The required support for aspects such as Pack and their interactions with atomicity is rationalized (1). Note that this AI is a hangover from Ada 2005.

Finally, there are changes to the core language regarding pre- and postconditions and requeue (90).

===

As defined in Ada 2012 a dispatching domain consists of a set of processors whose CPU values are contiguous. However, this is unrealistic since CPUs are often grouped together in other ways. Accordingly, AI-33 adds a type CPU_Set and two functions to the package System.Multiprocessors.Dispatching_Domains thus

```
type CPU_Set is array (CPU range <>) of Boolean;
function Create(Set: CPU_Set)
                              return Dispatching_Domain;
function Get_CPU_Set(Domain: Dispatching_Domain)
                              return CPU_Set;
```

Moreover, the original functions Create and Get_Last_CPU are modified to be

```
function Create(First: CPU; Last: CPU_Range)
                              return Dispatching_Domain;
function Get_Last_CPU(Domain: Dispatching_Domain)
                              return CPU_Range;
```

The changes enable Last to be zero thereby allowing for null domains. Remember that the type CPU_Range has lower bound of zero whereas the subtype CPU has lower bound of one. If a domain is empty then Get_Last_CPU returns zero and Get_First_CPU returns one.

A minor editorial change is that many instances of Dispatching_Domain (which is a type name) are changed to dispatching domain (the concept) by AI-82. An important clarification concerns the behaviour in the absence of any use of CPU and dispatching domains. The summary of AI-48 says that in the absence of any setting of the CPU of a task and the creation of any dispatching domains, a partition that specifies a language-defined dispatching policy will allow all tasks to run on all processors.

With regard to Ravenscar, the whole essence of its intention is to enable the use of a very simple runtime system. Accordingly, the newly added synchronous barriers should not be allowed and so the additional restriction

```
No_Dependence => Ada.Synchronous_Barriers
```

is added to the definition of the Ravenscar profile by AI-73. Furthermore, in the case of multiprocessors, in order to permit analysis we need to ensure that all tasks (including the environment task) are assigned to a specific CPU and especially that no task is assigned zero which indicates Not_A_Specific_CPU. So a new restriction is introduced, namely

```
No_Tasks_Unassigned_To_CPU
```

and this is also added to the definition of the Ravenscar profile (AI-117). Moreover, Ravenscar requires that the CPUs are denoted statically so another restriction is introduced

```
No_Dynamic_CPU_Assignment
```

and this is also added to Ravenscar (AI-55).

There are some clarifications concerning aspects in the Real-Time annex. One is simply to say that the real-time aspects of a type are evaluated when an object of the task type is actually declared (AI-81). This applies to the aspects Priority and Interrupt_Priority and also to CPU. Remember that the aspects do not necessarily have to be static, in particular they could be discriminants of the type and different for different objects. Thus

```
task type Slave(N: CPU_Range)
  with CPU => N;
Tom: Slave(1);   Dick: Slave(2);   Harry: Slave(3);
```

It is easy to be confused regarding priorities and interrupt priorities. Typically, a protected procedure used as an interrupt handler would have aspects giving the priority and interrupt to be handled thus

```
procedure Handler
  with Attach_Handler => Some_Interrupt,
       Interrupt_Priority => Hot_Priority;
```

However, if the procedure Handler is in a protected type Monitor then the priority could be given on Monitor itself using the aspects Priority or Interrupt_Priority. If no interrupt priority or priority aspect is specified, the priority is implementation-defined but in the range of interrupt priority (AI-51).

The mechanism for Asynchronous Transfer of Control (ATC) uses a form of select statement thus

```
select
  delay ... ;            -- triggering alternative
  ...
then abort
  Do_Something;          -- abortable part
end select;
```

This depends upon there being places in the abortable part that are abort completion points. The examples given in the RM in 9.7.4 rely upon this and some extra explanation is added (AI-98).

There has always been "boilerplate" in paragraph 3 of Appendix A about reentrancy. The obvious example is that

```
task T1 is
begin
  Put(A_File, "Text");
end T1;
task T2 is
begin
  Put(A_File, "More Text");
end T2;
```

is not required to work (being unsafe use of a shared variable, namely A_File). But if we change T2 to

```
task T2 is
begin
  Put(B_File, "More Text");
end T2;
```

then it is required to work provided A_File and B_File are indeed truly different files. The wording in paragraph 3 is improved to be more explicit with regard to parameter passing (52, 114). It now becomes

"The implementation shall ensure that each language-defined subprogram is reentrant in the sense that concurrent calls on any language-defined subprogram perform as specified, so long as all objects that are denoted by parameters that could be passed by reference or designated by parameters of an access type are nonoverlapping."

An important difference between protected functions and protected procedures (and entries) is that protected functions can be accessed concurrently. The principle is that such functions should be used for interrogating state and not to change it. However, in the case of calling functions inside a container, they do often change state. Accordingly, to enable containers to be used by parallel tasks and to impose control of access by protected objects, it is necessary to be able to make protected functions behave like protected procedures and so prevent multiple access. This can be done by a new aspect Exclusive_Functions which can be given for a protected type or a single protected object. Thus we write

```
protected type PT
  with Exclusive_Functions;
  ...
```

and then all protected functions declared within PT have exclusive access. Note carefully that the aspect is not permitted on individual protected functions but on the protected type (or object) as a whole (AI-129).

The usual convention for a prefixed view of a subprogram is Intrinsic which means that 'Access cannot be applied. However, an awkward situation has been discovered in the case of a subprogram with aspect Synchronization being By_Protected_Procedure. (Remember that the possible values for the aspect Synchronization are By_Entry, By_Protected_Procedure, and Optional.) In that case the convention is deemed to be protected so that Access can be applied (AI-107).

AI-1 is a hangover from Ada 2005. The essence of the problem is that the language is inconsistent regarding the pragma (now aspect) Pack. On the one hand the text of the RM regarding packing says that entities have to be squeezed up really tightly. On the other hand alignment properties, atomicity and so on ought to be respected. The revised text clarifies that Pack should not do anything that violates other requirements.

Finally, a bother with requeue is addressed by AI-90 (requeue has been the source of many bothers in the past so another one is not unexpected). This time the problem concerns pre- and postconditions. Suppose we have entries E1 and E2 with pre- and postconditions. And suppose that E1 does a requeue onto E2. The current text is unclear as to what exactly is checked and when. Do we avoid checking any postcondition on E1 and do we bypass any precondition on E2? Certainly not is the brief answer. Basically we require that the postcondition on E2 implies that on E1 by saying that they must fully conform. And moreover any precondition on E2 is indeed checked when the call is requeued. Remember that parameters are passed on unchanged and that the requeue statement does not have any explicit parameters itself.

6 Iterators, pools, etc.

This area covers the new iterators introduced in Ada 2012 plus access types and storage pools but also various miscellaneous features.

The following Ada Issues cover this area:

3 Specifying the standard storage pool

27 Access values should never designate unaliased components

36 The actual for an untagged formal derived type cannot be tagged

38 Shared_Passive package restrictions

43 Details of the storage pool used when Storage_Size is specified

46 Enforcing legality for anonymous access components in record aggregates

47 Generalized iterators and discriminant-dependent components

67 Accessibility of explicitly aliased parameters of procedures and entries

70 9.3(2) does not work for anonymous access types

72 Missing rules for Discard_Names aspect

76 Variable state in pure packages

85 Missing aspect cases for Remote_Types

89 Accessibility rules need to take into account that a generic function is not a function

93 Iterator with indefinite cursor

120 Legality and exceptions of generalized loop iteration

124 Add Object'Image

136 Language-defined packages and aspect Default_Storage_Pool

138 Iterators of formal derived types

142 Bad subpool implementations

145 Pool_of_Subpool returns null when called too early

148 Dangling references

151 Meaning of subtype_indication in array component iterators

These changes can be grouped as follows.

A number of issues concern default and standard storage pools in general (3, 43, 138) and some issues concern the newly introduced subpools (142, 145, 148).

Several issues concern clarifications and omissions regarding generalized iterators (47, 93, 120, 138, 151).

As ever there are issues regarding accessibility rules, anonymous access types and related topics (27, 46, 67, 70, 89).

There are some clarifications and omissions about package state such as Pure and Shared_Passive (38, 76, 85).

Finally, there are miscellaneous issues on derived types (36), Discard_Names (72), and Object'Image (124).

===

Remember that when we declare an access type we can specify which storage pool it is to use. If we do not specify one then the default is used. Originally this default was just the "standard pool". The pragma Default_Storage_Pool was introduced in Ada 2012. It enables the user to specify which pool is to be used by default if none is specified for the access type. Thus we might write

```
pragma Default_Storage_Pool(My_Pool);
```

Moreover, the parameter can be null thus

```
pragma Default_Storage_Pool(null);
```

which ensures that we must always specify the pool to be used and prevents any allocation by default. AI-3 enables us to go back to the standard pool by writing

```
pragma Default_Storage_Pool(Standard);
```

This additional argument means that there is a minor syntax change thus

storage_pool_indicator ::=
*storage_pool*_name | **null** | Standard

Note that the indicator Standard has nothing to do with the package Standard as such.

AI-43 makes subtle changes to the behaviour of the aspect Storage_Size as applied to storage pools. Briefly, the pool used by an access type that has Storage_Size given must not allocate additional storage when the original amount is exhausted and no other type can use the same pool unless requested. So we might have

```
type T is access ...
for T'Storage_Size use 1000;
type S is access ...
for S'Storage_Pool use T'Storage_Pool;  -- share pools
```

Note that if we do give the aspect Storage_Size for a type then that implies the (implementation-defined) storage pool for the type and so we cannot also give the aspect Storage_Pool for that type. Contrariwise if we do give the pool explicitly by for example

```
for T'Storage_Pool use My_Pool;
```

then the storage size is determined by the behaviour of My_Pool and the aspect Storage_Size cannot be given explicitly (the writer of the pool will have had to declare a function Storage_Size as part of the implementation of the pool and that will act as the attribute.)

AI-136 concerns the use of default storage pools with language defined generic units. After some discussion it is concluded that the effect of specifying the aspect Default_Storage_Pool on an instance of a language-defined generic unit is implementation-defined. One consequence of this is that one cannot rely upon using the aspect Default_Storage_Pool to change the storage pool used by a container such as a linked list if the container is an instance of the language-defined container Doubly_Linked_List.

Three AIs concern subpools which were introduced in Ada 2012 and clarify a number of omissions. AI-142 simply says that Allocate_From_Subpool could be erroneous if not implemented in accordance with the given rules. AI-145 says that the function Pool_Of_Subpool returns **null** if called before calling the procedure Set_Pool_Of_Subpool (pretty obvious). AI-148 tidies up the loose wording regarding what happens when we deallocate subpools (all objects that were in them cease to exist of course so beware dangling references as usual).

There are some omissions regarding iterators which were added in Ada 2012. AI-138 concerns the inheritance of aspects such as Constant_Indexing and Iterator_Element. Remember that a type such as List in Doubly_Linked_Lists has aspects thus

```
type List is tagged private
  with Constant_Indexing => Constant_Reference,
       Variable_Indexing => Reference,
       Default_Iterator => Iterate,
       Iterator_Element => Element_Type;
```

If we derive a type from List then we cannot change Iterator_Element into something other than Element_Type. We say that these aspects are non-overridable (they could be confirmed).

The other AIs in this group (47, 93, 120, 151) concern the new generalized iterators and address a number of curious omissions.

It might be recalled that if we have an array of type T thus

```
type ATT is array (1 .. N) of T;
The_Array: ATT;
```

then rather than express iteration as

```
for I in The_Array'Range loop
  The_Array(I) := 99;    -- do something to The_Array(I)
end loop;
```

we can more briefly use **of** rather than **in** and write

```
for E of The_Array loop
  E := 99;               -- do something to component E
end loop;
```

Optionally we can give the subtype of E thus

```
for E: T of The_Array loop ...
```

AI-151 says that any subtype given must statically match that of the component of the array (obvious really). Adding T is essentially a comment to aid the reader but the kindly compiler checks that it is correct.

The other AIs of this group essentially come down to the same thing. Generalized iteration enables us to write something in a shorthand way. When the shorthand is expanded, what is done using the resulting long form must not be illegal. For example AI-47 shows how we might appear to make the array object vanish, AI-93 says that exceptions might be raised and AI-120 covers problems with limitedness and constantness.

Access types and particularly the anonymous access types introduced in Ada 2005 are often a source of problems. AI-27 clarifies the behaviour of value conversions of composite objects. AI-67 clarifies the accessibility of explicitly aliased parameters. AI-46 addresses the issue of the legality of record components of anonymous access types. AI-70 covers issues of the master of tasks created by anonymous access types.

AI-89 is more interesting. It suggests that Ada programmers should carefully remember the golden rules "a generic function is not a function", "a generic procedure is not a procedure", and "a generic package is not a package". The details of the AI are a bit elusive but revolve around the above rules.

Another group of issues concern matters such as the state of packages. AI-76 shows how an apparently pure package could seem to have its state changed via tricks such as using a self-referential type. Such trickery is deemed erroneous.

AI-85 notes that we cannot permit giving the aspects Storage_Size or Storage_Pool for remote access to class wide types which are given in a package with the aspect Remote_Types.

AI-38 concerns packages that are Shared_Passive. Various rules concerning the misuse of access types are strengthened.

A curious error in the matching rules for generic parameters has long been overlooked and is corrected by AI-36. If a formal parameter of a generic unit is derived untagged, then a corresponding actual parameter must also be untagged. Thus if we have

```
generic
  type T is private;
  type TT is new T;
package P ...
```

then we cannot instantiate P with a tagged type for TT. This has been wrong ever since Ada 95.

The pragma Discard_Names was introduced in Ada 95. It tells the compiler to throw away tedious tables of names at runtime associated with things such as Image and Value. AI-72 points out that Ada 2012 forgot to say that Discard_Names is now an aspect and can be given as such. So if we have an enumeration type with lots of long identifiers such as

```
type Greek is (alpha, beta, gamma, ... , omega);
```

then rather than separately giving

```
pragma Discard_Names(Greek);
```

we can add the aspect when the type is declared thus

```
type Greek is (alpha, beta, gamma, ... , omega)
  with Discard_Names;
```

Finally, AI-124 proudly announces the extension of the attribute Image to apply to objects as well as to types.

At the moment if a slovenly programmer wants to avoid the majesty of the full might of Integer_Text_IO to print out the value of N of some integer type such as My_Nice_Integer_Type (perhaps for diagnostic purposes) then they write

```
Put(My_Nice_Integer_Type'Image(N));
```

And now thanks to AI-124, this becomes

```
Put(N'Image);
```

Note that GNAT users have been writing N'Img for a long time.

7 Predefined library

The main improvements in the standard library in Ada 2012 concern containers and these are addressed in the next section. There were also other additions such as UTF encoding packages, extensions to directories and the package Locales.

The following Ada Issues cover this area:

28*	Import of variadic C functions
30	Formal derived types and stream attribute availability
31	All_Calls_Remote and indirect calls
34	Remote stream attribute calls
37*	New types in Ada.Locales cannot be converted to/from strings
88	UTF_Encoding.Conversions and overlong characters on input
102	Stream_IO.File_Type has Preelaborable_Initialization
106	Write'Class attribute
121	Stream-oriented aspects
130	All I/O packages should have Flush
135	Enumeration types should be eligible for convention C
146	Should say stream-oriented attribute

These changes can be grouped as follows.

A number of issues concern streams, their aspects and attributes (30, 34, 102, 106, 121, 146).

Two issues concern interfacing to the C language (28, 135).

Minor issues concern I/O packages and Flush (130), the aspect All_Calls_Remote (31), and UTF_Encoding (88).

Finally, there is a major revision to the new package Ada.Locales (37).

===

AI-121 points out that the stream-oriented attributes such as 'Read and 'Write can be given using an aspect specification as well as by an attribute definition clause. Thus for a type Date we can declare a procedure Date_Write and associate them using

```
for Date'Write use Date_Write;
```

Alternatively, we can declare Date as

```
type Date is record ... end record
  with Write => Date_Write, ... ;
```

and then declare the procedure Date_Write.

AI-146 just corrects a bit of wording; strictly we always talk about stream-oriented attributes and not stream attributes. AI-30 clarifies the use of stream-oriented attributes with untagged formal derived types but ironically refers to them as stream attributes. AI-34 concerns the use of streams and Remote_Types packages; in summary, dereferencing a remote access-to-classwide type to make a dispatching call to a stream-oriented attribute such as 'Write is not allowed.

AI-106 clarifies the way in which a class wide stream-oriented aspect is given. For example

```
type My_Type is abstract tagged null record
  with Write'Class => My_Write;
```

AI-102 adds the pragma Preelaborable_Initialization to the type File_Type in the package Ada.Streams.Stream_IO. It points out that the package was made preelaborable in Ada 2012 so that it was more useful but the corresponding change to the private type File_Type was forgotten.

Two issues concern interfacing to C. AI-28 discusses the import of variadic C functions (that is functions with a variable number of parameters). In Ada 95, it was expected that such functions would use the same calling conventions as normal C functions; however, that is not true for some targets today. Accordingly, this AI adds additional conventions to describe variadic C functions so that the Ada compiler can compile the correct calling sequence. The other issue (AI-135) concerns enumeration types and makes them eligible for convention C provided certain range conditions are satisfied.

With regard to input-output in general AI-130 adds the procedure Flush to the packages Sequential_IO and Direct_IO so that any internal buffers can be flushed in the same way as in Text_IO and Stream_IO.

A rather specialized change is made by AI-31 concerning the aspect All_Calls_Remote. It states that the aspect applies to all indirect or dispatching remote subprogram calls to the RCI (remote call interface unit) as well as to direct calls from outside. All indirect or dispatching calls should go through the PCS (partition communication subsystem).

A number of packages for UTF encoding were added in Ada 2012. AI-88 addresses a minor issue regarding overlong characters on input, that is as a parameter to a function Encode or Convert. It confirms that such overlong encodings do not raise Encoding_Error.

Finally, AI-37 discusses a curious difficulty found in attempting to use the seemingly innocuous new package Ada.Locales. It was mentioned in the Ada 2012 Rationale which we repeat.

The types Language_Code and Country_Code were originally declared as

```
type Language_Code is array (1 .. 3) of Character
                                   range 'a' .. 'z';
type Country_Code is array (1 .. 2) of Character
                                   range 'A' .. 'Z';
```

The problem is that a value of these types is not a string and cannot easily be converted into a string because of the range constraints and so cannot be a simple parameter of a subprogram such as Put. If LC is of type Language_Code then we have to write something tedious such as

```
Put(LC(1)); Put(LC(2)); Put(LC(3));
```

Accordingly, these types are changed so that they are derived from the type String and the constraints on the letters are then imposed by dynamic predicates. So we have

```
type Language_Code is new String(1 .. 3)
  with Dynamic_Predicate =>
     (for all E of Language_Code => E in 'a' .. 'z';
```

with a similar construction for Country_Code.

Readers might like to continue to contemplate whether this is an excellent illustration of some of the new features of Ada 2012 or simply an illustration of static strong or maybe string typing going astray.

8 Containers

The container library was considerably enhanced in Ada 2012. A few issues have arisen since.

The following Ada Issues cover this area:

35	Accessibility checks for indefinite elements of containers
69	Inconsistency in Tree container definition
78	Definition of node for tree container is confusing
110	Tampering checks are performed first

These changes can be grouped as follows.

AI-69 and AI-78 both address the same issue regarding the fact that the root node of a tree has no element.

AI-35 concerns problems with accessibility checks necessary to prevent dangling references when using the indefinite containers.

AI-110 addresses the question of when tampering checks are performed.

===

It is fundamental to the organization of trees that each node of the tree has an associated element containing a value except the root node which has no such associated element. Both AI-69 and AI-78 make various corrections to the wording such as to point out that an iterator never visits the root node.

AI-35 addresses the question of accessibility checks when manipulating indefinite containers (these containers were introduced in Ada 2005). Certain operations of instances of the indefinite container packages require accessibility checks to prevent dangling references. The term "perform indefinite insertion" is defined and then this is used in the description of the various operations. Thus in the case of Indefinite_Doubly_Linked_Lists we are told that Append, Insert, Prepend, and Replace_Element that have a parameter of the Element_Type perform indefinite insertion.

AI-110 concerns the order of making various checks. The conclusion is that tampering checks are always performed before any other checks such as that for capacity.

9 Conclusions

A number of presentation AIs (56, 80, 134, 159) which cover mostly trivial typos have not been discussed. One amusing example will suffice. Paragraph 10 of clause A.18.25 on bounded multiway trees says that the function Copy is declared as

```
function Copy(Source: Tree; Capacity: Count_Type := 0)
  return List;
```

Clearly List is an alternative spelling for Tree!

Altogether, the changes made by the update are relatively minor. There is no need to feel that your valuable copies of the Ada 2012 Reference Manual., the Ada 2012 Rationale or indeed Programming in Ada 2012 are somehow now useless. If you stumble across any of these minor blemishes then the Ada compiler will undoubtedly be your guide.

Finally, I need to thank all those who have helped in the preparation of this paper and especially Randy Brukardt, Dirk Craeynest, Jeff Cousins and Joyce Tokar.

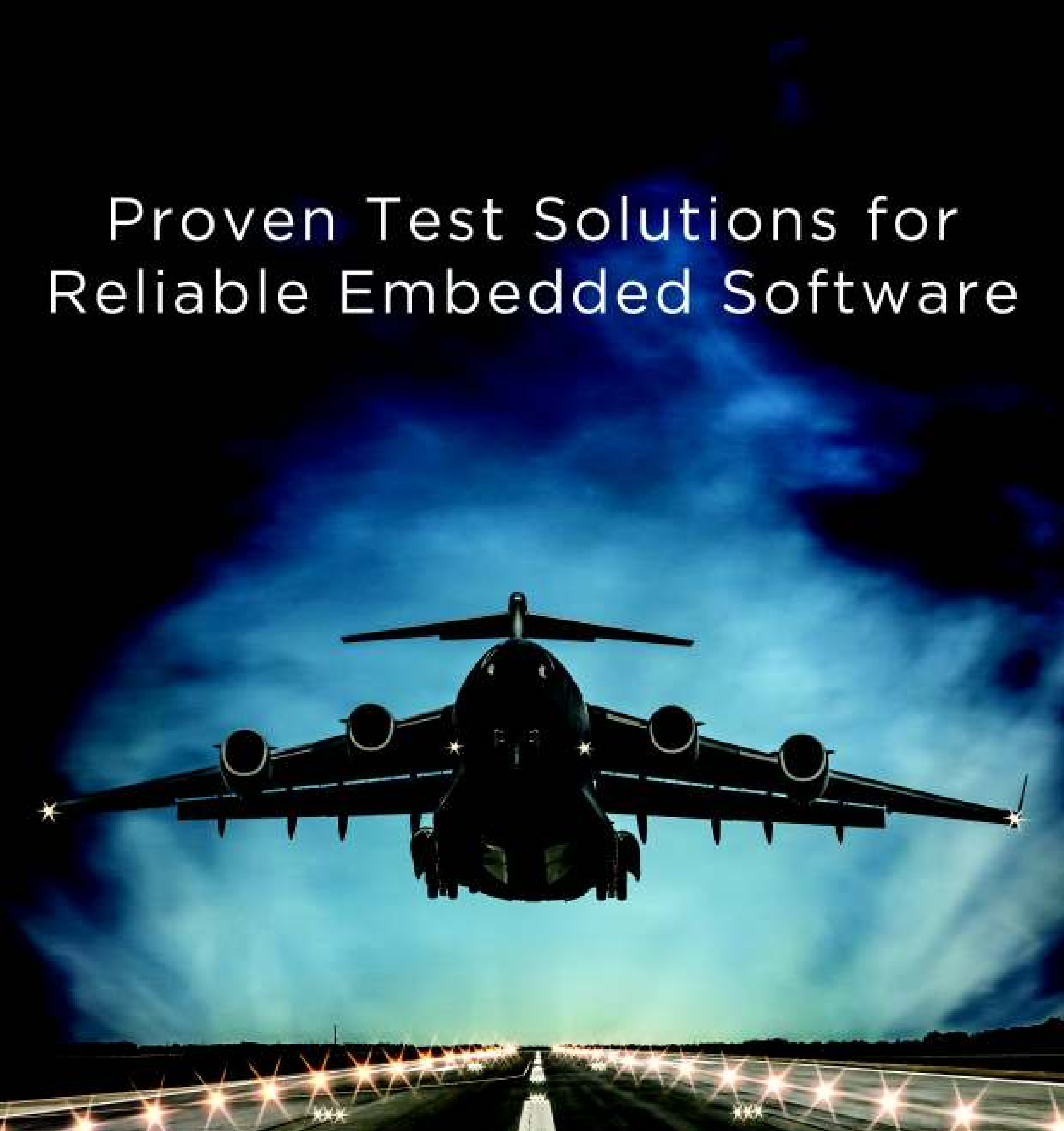
References

[1] ISO/IEC 8652: 2012E, *Ada Reference Manual.*

[2] S. T. Taft et al (eds.) (2012) *Ada 2012 Reference Manual*, LNCS 8339, Springer-Verlag.

[3] John Barnes (2013), *Ada 2012 Rationale*, LNCS 8338, Springer-Verlag

[4] John Barnes (2014), *Programming in Ada 2012*, Cambridge University Press.



VECTOR software

vectorcast.com

"VectorCAST is unique in that it provides us with the ability to increase the reliability and quality of our flight software."

-- Honeywell

VectorCAST is a TÜV SÜD Certified Software Tool for Safety Related Development

Vector Software, Inc.
Golden Cross House | 8 Duncannon Street | London WC2N 4JF UK | +44 203 603 0120 | sales@vectorcast.com

Overview of the 17th International Real-Time Ada Workshop

20-22 April 2015
Bennington, Vermont, USA

Contents *

Program Committee

Workshop Participants

Mario Aldea Rivas, University of Cantabria, Spain
Patrick Bernardi, Australian National University, Australia
Alan Burns, University of York, UK
Carl Brandon, Vermont Tech, USA
Robert Dewar, AdaCore, USA
Tristan Gingold, AdaCore, USA
Michael González Harbour, University of Cantabria, Spain
Stephen Michell, Maurya Software, Canada
Brad Moore, General Dynamics, Canada
Luis Miguel Pinho, Polytechnic Institute of Porto, Portugal
Juan Antonio de la Puente, Technical University of Madrid, Spain
Jorge Real, Universitat Politècnica de València, Spain
Pat Rogers, AdaCore, USA
José Ruiz, AdaCore, France
Sergio Sáez, Universitat Politècnica de València, Spain
Joyce Tokar, Pyrrhus Software, USA
Tullio Vardanega, University of Padua, Italy
Andy Wellings, University of York, UK
Juan Zamorano, Technical University of Madrid, Spain

Sponsors

Ada europe

AdaCore The GNAT Pro Company

* The Proceedings of the 17th International Real-Time Ada Workshop are published in the April 2015 issue of ACM Ada Letters.

Session Summary: Conformance Issues

Chair: Alan Burns
Rapporteur: Juan Antonio de la Puente

1 Introduction

The aim of the session was to discuss the role of the IRTAW group in generating test cases for Ada real-time support and updating the ISO/IEC technical reports related to high integrity systems and vulnerabilities in Ada.

2 Conformance tests

The basis for the discussion was the paper by Alan Burns and Andy Wellings on *Testing Conformity to the Real-Time Annex* [2]. Alan Burns started by recalling the requirement from the Ada Rapporteur Group (ARG) to extend the current tests in ACATS (Ada Conformity Assessment Test Suite) [1] in order to properly cover the real-time features defined in Annex D of the Ada 2012 standard. His presentation went on with a summary of the current status of the ACTS tests, and the need to complete those related to the real-time annex.

The group agreed that it is its responsibility to provide tests for the Annex D features, even though some of them may not be amenable to being tested. An example is the requirement of an efficient implementation when tasking restrictions are used.

A recommendation that all future proposals of new features should be provided with a test case was also agreed upon. The test case should be designed after the ARG initially considers the issue and starts working on the details.

As a practical approach to writing the tests, an attempt will be made to hold a seminar on test writing for interested people. Ada-Europe will set up a web page with information about the process, and AdaCore offered to provide an environment where participants can collaborate in the development of new tests.

3 Technical reports

The second part of the session was motivated by Joyce Tokar's position paper on updating ISO/IEC TR 15942 [6]. She started the discussion by widening the topic to the following three ISO/IEC technical reports that may need updating:

- TR 15942 — Guide for the use of the Ada programming language in high integrity systems [3];
- TR 24718 — Guide for the use of the Ada Ravenscar profile in high integrity systems [4];
- TR 24772 — Guidance to avoiding vulnerabilities in programming languages through language selection and use. Annex C. Vulnerability descriptions for the language Ada [5].

After a detailed account of the origins and history of the first two documents, a discussion followed on the effort required for updating them. Since the need for this effort was not clear, it was decided to start a survey in order to decide if such an update is important to industrial users, and in this case try to find volunteers to do the job.

The group also agreed on continuing support to the work of WG9 on the Ada annex to the vulnerabilities report (TR 24772).

4 Conclusions

In summary, the group concluded that updating the ACATS tests and contributing to the Ada annex to the vulnerabilities report are the most urgent tasks in the standards area. On the other hand, the availability of resources for updating the high-integrity reports and the interest of the work itself need further clarification.

References

[1] Ada Conformity Assessment Authority (2014), *Ada conformity assessment test suite (ACATS)*. URL `http://www.ada-auth.org/acats.html`.

[2] A. Burns and A. Wellings (2015), *Testing conformity to the real-time annex*, In Ada Letters vol.35 (1), pp. 17-25, ACM.

[3] ISO/IEC (2000), *TR15942:2000 — Information Technology — Programming Languages — Guide for the Use of the Ada Programming Language in High Integrity Systems.*

[4] ISO/IEC (2005), *TR 24718:2005 — Information Technology — Programming Languages — Guide for the use of the Ada Ravenscar profile in high integrity systems.*

[5] ISO/IEC (2014), *TR 24772 — Information Technology — Programming Languages — Guidance to avoiding vulnerabilities in programming languages through language selection and use.* Draft 3rd ed.

[6] J. Tokar (2015), *Update of ISO/IEC TR 15942, programming languages — Guide for the use of the Ada programming language in high integrity systems*, In Ada Letters vol.35 (1), pp. 93-94, ACM.

Session Summary: Fine-grained Parallelism

Chairs: Luis Miguel Pinho and Stephen Michell
Rapporteur: Brad Moore

Session Goals

The main goals of this session were to:

- *Present an overview of a model for fine-grained parallelism in Ada based on the notion of tasklets;*
- *Present and discuss a general execution model that would support parallelism constructs being considered for possible inclusion in a future version of the Ada standard;*
- *Present and discuss a real-time model that provides consistency with the general model while providing enough flexibility to accommodate a wide range of real-time systems with the intent of supporting real-time analysis and maintaining or improving the safety features of the language.*

1 Real-Time Fine-Grained Parallelism in Ada

This session presented a model for fine grained parallelism that could integrate into the existing Ada real-time tasking model to extend capabilities to provide better support for multicore parallelism. Luís Miguel Pinho gave the presentation which covered this broad topic, and identified various issues that the model was designed to address, while also highlighting open issues for further discussion and research.

The model extends the existing Ada tasking model in some significant ways, and thus new terminology was needed to facilitate understanding. In particular, the notion of tasklets was introduced which allows an Ada task to potentially contain many light weight threads of execution. These light weight threads provide parallelism using fully strict fork-join constructs. The concept is that tasklets may be created explicitly using parallelism syntax constructs, or implicitly by the compiler, when the compiler has sufficient knowledge of the semantics of the code to allow this to happen.

An important point is that an Ada task with no parallelism, is itself still considered to be a tasklet. In other words, all executing Ada code is considered to be carried out by tasklets. Tasklets are a logical entity only, and not visible in the text of an Ada program.

Miguel presented two main parallelism syntactic constructs, parallel blocks, and parallel loops, which together provide a good coverage of the needs that one encounters when looking for improved performance for multicore systems. To provide the compiler with better knowledge for avoiding data races, Miguel then described two new aspects, Global and Potentially_Blocking that together point out where such data races can exist. The idea being that if the compiler can know where sections of code exist that are free from data races, the compiler can generate implicit parallelism to make better use of the available cores.

Next, Miguel went on to describe the Executor model, where Executors are the agents of execution that carry out the underlying tasklets of an application. Miguel mentioned that it can help to think of Executors as OS threads, although the model allows for other mechanisms to be used or developed.

A problematic area involves allowing potentially blocking calls to be allowed in tasklets. To accommodate this, Miguel then introduced the notion of progress guarantees. Three forms of progress guarantee were described: Immediate Progress, Eventual Progress, and Limited Progress, which can be mapped to implementation strategies with regard to Executor allocation. At a basic level, executors are either allocated immediately as needed, eventually if not immediately, or analysis is needed to ensure that the number of executors planned for the execution is sufficient to prevent deadlock.

Other issues raised with respect to real-time centred around the topics of priority and other real-time task attributes. In particular, tasklets execute at the priority (or deadline) of the associated task. Boosts in priority resulting from protected actions only affect the tasklets that are within the protected action, and all other tasklets of the same task continue at the base priority/deadline of the associated task.

Another concern is in dealing with how parent tasklets wait for child tasklets to complete. The model suggests that the parent should execute child tasklets if available, while waiting for children to complete, and then if child tasklets are not available, the parent should spin as if executing the parent tasklet.

A number of possible restrictions and controls were then identified which would allow a real-time programmer to control and understand the underlying parallelism, which would be needed to facilitate analysis. During the presentation, a number of open issues were presented and discussed which are described below.

2 Discussions

Availability to General Programmer

One of the first discussion points was whether parallelism constructs should be made available to general programmers. If Ada is to be serving the safety critical niche, would adding new parallelism capabilities provide too many ways for a general programmer to introduce logic errors and data races, and generally weaken Ada's position in this market? If the compiler is given better capabilities to determine where parallelism can be generated implicitly, would there still be a need for explicit parallelism?

It was pointed out that parallelism is needed for certain algorithms, and many such algorithms need to be rewritten to accommodate parallelism. Compilers will not be able to implicitly generate such parallelism for all of these algorithms. It was mentioned that another need is specifically for real-time. The real-time programmer will need control and understand the parallelism in order to provide better real-time analysis. Real-time control may be needed to force the implementation to deviate from the default behaviour as different analysis methods may work better with different run-time behaviours. Steve Michell also noted that there is a trend towards higher levels of complexity in real-time systems. He gave the example of vehicle braking systems and collision avoidance systems, which require input from vision systems or sensor networks, highlighting the fact that the need is a growing one. Even where parallelism can be implicitly generated, it is beneficial to be able to explicitly state the parallelism as a design intention by the programmer. Thus both explicit and implicit parallelism is needed. Andy Wellings suggested that the area of parallelism could involve a fair bit of academic work, as it would be desirable to ensure that worthwhile features are added to the language without introducing undesirable features. It is not an all or nothing set of capabilities.

Syntax for Specifying Parallel Constructs

Another discussion point asked the question whether parallelism capabilities should be confined to approaches involving libraries, or whether the Ada language should be extended with new syntax to accommodate this parallelism. On the one hand, it is desirable to avoid adding further complexity to the language, but on the other hand, the amount of new syntax needed seems to be relatively small, yet provides better safety and appears to be less error prone than relying on libraries. In particular, it was noted that safety is increased since the compiler can interpret the usage of parallelism and verify the semantics and indicate problems with the programmer code, such as pointing out problems with data races. It was generally agreed that adding parallelism support to the language using a model and syntactic constructs such as those presented would be useful and important for current multicore platforms as well as provide better support for the future where the number of available cores is expected to increase as many-core platforms become more prevalent. Andy Wellings felt that the need for parallel loops was clear, but wondered if the need for parallel blocks was as important, or could be omitted if necessary. It was noted that the parallel block construct does provide capabilities that are difficult to accomplish with loops, such as recursive parallelism.

Joyce Tokar raised a question about whether guidance should be provided in some form to help programmers work with and understand the new concepts. She noted that slicing parallelism across arrays can be dangerous and difficult to get right if there are too many options, and there can be too many ways for a programmer to introduce logic errors and data races, or introduce behaviours that worsen instead of improve performance. It was generally agreed that guidance in some form would be helpful for programmers.

Exception Propagation

On the topic of exception propagation, the question was raised whether the choice of propagation should be well defined, or arbitrary. For example, in the case of a parallel block construct that is being executed sequentially, what should happen if an exception occurs in the first block? Are the other blocks expected to execute? Miguel's response indicated that there is no expectation for the subsequent blocks to begin execution. If blocks have already begun execution, then it would be allowed that they would execute until completion, but an implementation could choose to abort those tasklets early if an exception was raised in one of the blocks, as it was noted that all parallel execution is considered failed if an exception occurs. Andy Wellings at first did not like that behaviour (see discussion below on model of parallelism), but after discussion warmed to the idea. Pat Rogers asked whether if it might be desirable to propagate all exceptions raised in some sort of aggregate form, rather than only propagating only one of them. The group eventually agreed that the choice of exception to be propagated when multiple blocks generate exceptions should be arbitrary, and up to the implementation. Michael González Harbour asked about other forms of transfer of control. For example, what happens if a return statement occurs first, and then an exception occurs in another tasklet as work is finished? The group generally felt that the same approach would be applied and the choice of transfer mechanism out of the parallelism construct would be chosen arbitrarily. While it was agreed that arbitrary propagation is likely a good solution, more research would be needed to determine if a better solution is possible. It was noted that transfer of control was expected to be allowed to leave a parallelism construct, but a transfer of control could not be used to enter a parallelism construct, which is consistent with how transfers of control are currently defined with respect to blocks statements in Ada.

Model of Parallelism

One topic that generated a fair amount of philosophical discussion was whether the parallelism constructs represent sequential code that is being parallelized, or whether the constructs indicate that parallel code is being written. The answer to this question is important in understanding the models. While to some extent this applied to both parallel loops and parallel blocks, it was felt that it mostly applied to parallel blocks. Andy Wellings initially felt that if it is sequential code being parallelized, then he was not happy with the use of the parallel keyword in the construct. In languages such as Parasail, parallelism is the default, and you need to specify where the code needs to be sequential. The model in Ada is the other way round. He initially suggested that "parallelizable" would be a more accurate description. Miguel noted that other frameworks in other languages already use the "parallel" keyword in a similar manner as being proposed, so it shouldn't be too difficult for people to understand. It was also mentioned that "parallelizable" would be awkward to both say and write. After discussion, Andy Welling's concerns were mostly addressed and he mentioned he felt comfortable going forward with "parallel" as a keyword.

Executor Model

The next topics of discussion were centred around the Executor model. A number of interesting questions were raised. The first being, can a task rendezvous with itself? This led into the part of the presentation describing the three progression models: immediate progress, eventual progress, and limited progress. It was explained that the progress models allowed this to occur safely, since potentially blocking operations result in the tasklet making the attempt to rendezvous to be processed by a separate executor. This guarantees that the executor performing the accept is not the same one that is requesting the rendezvous, thus avoiding deadlock. Ada already has the notion of the same task attempting multiple rendezvous in parallel on the same entry. This can occur with the asynchronous transfer of control capability.

Some clarification was needed to understand the progress models. For immediate progression, it was unclear what was meant if no cores were available. It was explained that availability of a core means that it is idle or executing lower priority tasklets. Making progress means that a tasklet can compete for scheduling. It does not mean that it can execute. In other words, it means that the tasklet has an executor allocated to it. Miguel explained that immediate progress is essentially global scheduling. Andy Wellings felt that "progress" is the wrong word to describe the intent. It really means availability of an executor. Tullio Vardanega suggested that it may be helpful to consider and describe how bounded vs unbounded executor pools map to the progress models. Miguel said he would revisit the use of terminology to see if there are possible improvements that could be made.

Miguel presented an example of a loop where one of the iterations in the loop releases a protected object that cause other iterations of the loop to block. The model as presented, suggested that the example would succeed if executing in parallel but deadlock if sequential. Therefore, it was asked if the code deadlocks when executing sequentially, should it be expected to work when executing in parallel? This was another philosophical question, which was deemed an open issue. Tullio Vardanega had concerns about the limited progress model. In particular, he felt that being able to determine the number of executors needed for a system is not going to be an easy thing to do. Static determination of executor counts needed for limited progress remains an open issue.

Accept Statements in Parallel Constructs

It was then asked whether accept statements would be allowed in tasklets. What should happen if each block of a parallel block construct in a task has an accept statement on the same entry? This was a situation that hadn't been considered, but after discussion, the group felt that accept statements should probably not be allowed from inside parallel constructs, as there were both safety concerns and implementation difficulty concerns about supporting a feature with questionable usefulness. This should be relatively easy to enforce since accept statement cannot occur inside procedure bodies, and are only allowed in the main task body.

Andy Wellings asked whether tasklet interaction would be allowed in abort-deferred regions. This required more deliberation, and after regrouping the group arrived at the conclusion that adding parallelism within abort-deferred regions should probably be disallowed. The main problem being that a protected object has a lock to prevent data races within the protected object. If generating parallelism within the protected object were allowed, then another lock could eventually be needed to protect the variables inside the protected object. The main concerns here were of introducing too much complexity for implementers to support.

Potentially Blocking Operations in Tasklets

Andy Wellings asked whether both parallel loops and parallel blocks have needs to support potentially blocking scenarios. The cumulative sum problem was given as an example where parallelism using potentially blocking operations could be useful. While an example for parallel blocks was not given, it should be noted that I/O is another case where potentially blocking operations are needed, which can be generally useful both within parallel loops and parallel blocks.

Real Time Controls

A number of real-time controls were identified to allow programmers to better analyze the run-time behaviour. It was noted that No_Implicit_Parallelism cannot eliminate parallelism at the hardware level, so it should be described as applying only to software parallelism generated by the compiler. It was also asked whether all the controls were necessary. Is it necessary to be able to control both the number of tasklets as well as the number of executors? It was explained that both can play a role towards providing analyzability of the run-time behaviour. In the case of parallel loops for example, one needs to know both the number of chunks (i.e., tasklets) that were generated for the loop, as well as how many executors are assigned to execute those chunks.

Task Attributes in Tasklets

Another area of consideration was with regard to priorities, task deadlines, and other task attributes. It was generally felt that assigning priorities and deadlines at the tasklet level would be too onerous and cumbersome. The task attributes generally are shared among all tasklets associated with a given task. This led to questions about what should happen if two tasklets of the same task call Set_Priority, or modify other task attributes. Once again, it was felt that the choice of which modification should ultimately take place should be chosen arbitrarily by the implementation. Set_Priority in particular was noted as being problematic, and different alternatives were considered. Currently in Ada, Set_Priority does not always immediately take effect, so it may make sense to postpone the effect until non-parallel code, or restrict the call to be disallowed within parallelism constructs. Another alternative would be to provide guidance to suggest that Set_Priority should not be called from within parallel code. Andy Wellings disagreed, as you cannot specify programmers not do stupid things. Rather he felt we should just state what happens when the call is made from within parallelism constructs. It was noted that having the compiler reorder the calls to Set_Priority should not be allowed because the compiler is not supposed to generate implicit parallelism if there is no visible effect, and reordering calls to Set_Priority would have this effect. It was generally felt that deferring the call or disallowing it when executing in a parallel setting would make the most sense. Treatment of the Set_Priority call remains an open issue though. Jorge Real asked how it would be possible to determine if one was within a parallelism construct. This may not be easy to determine if nested subprogram calls are involved, and making such a determination should not significantly impact performance which would defeat the purpose. More thought is needed in this area, which is another open issue.

Timers and Interrupts

Yet another set of issues raised was with regard to interrupts and timer events. Should it be possible to create parallelism tasklets from within an interrupt handler or timer event? Suppose ten timing events are created which expire at the same time. Can these events execute in parallel? While this is likely an implementation specific capability, the proposed model does not disallow such possibilities. Execution timers in particular are problematic however. Currently there is one execution budget per task. If the task executes in parallel, how can the execution time be tracked? One possible solution would be to only update the execution time at the end of the parallel activity. So far this deferred model seems to be the only viable solution that wouldn't have significant impact on performance, but clearly more research is needed in this area. Another possibility would be to have a set of budget values per core, but Tullio thought this would lead to too much complexity. If the parent tasklet is spinning, then you have a mechanism to determine how much time is spent in the core. While that might be a useful technique used in developing a solution, more thought is needed.

Other model variants

While the real-time models presented provide a fair amount of flexibility, it was acknowledged that other sensible models may be possible, or other variations of the presented model may also be viable. Examples given include other preemptive models where preemption doesn't occur if child tasklets are executing, or where preemption is deferred until the end of the parallelism.

3 Summary

The following summarizes the positions taken by the workshop during this session:

- There was general agreement that support for parallelism features should be added to the language in the form of parallelism constructs involving syntax for parallel loops and parallel blocks, but that some form of guidance document would also be needed to help programmers make the best use of the available cores and avoid various pitfalls.
- The parallel keyword is likely the best choice to use in defining the syntax for these new parallelism constructs.
- When multiple transfers of control occur in parallel, likely the choice of transfer should be arbitrarily chosen by the implementation.
- Parallelism constructs should probably be disallowed within abort-deferred regions.
- Parallelism constructs should probably disallow accept statements within the construct.
- More thought and research is needed to address the open issues described in [1] and above. In particular; Multiple parallel updates to task attributes need to be worked out;
  - The choice of arbitrary selection for transfer of control out of parallel constructs should be further scrutinized to see if the behaviour should be more well-defined;
  - The terminology used for describing the progress models should be revisited to see if better terminology can be used, to facilitate understanding of the concepts; Efficient mechanisms to determine if code is executing in parallel should be investigated, particularly if parallelism constructs are to be disallowed within abort-deferred regions;
  - Methods to efficiently track and implement execution timers should be explored.

References

[1] L. M. Pinho, B. Moore, S. Michell, S. T. Taft (2015), *Real-Time Fine Grained Parallelism in Ada*, In Ada Letters vol. 35 (1), pp. 46-58.

Session Summary: Language Abstractions

Chair: Andy Wellings
Rapporteur: Jorge Real

1 Introduction

This session reviewed the support for two classes of language abstractions for which position papers had been submitted. The workshop first considered the ability to monitor and control the affinity of protected handlers for both interrupts and timing events, based on the ideas and results presented in papers [5] and [3]. Then we considered the incorporation of the concept of cyclic tasks directly into the language, according to the proposal in [1].

2 Support for event-handler affinity

The session chair set the context of the issues at hand. There is a lack of support in Ada for controlling the affinity of event handlers (namely timing-event, execution-time and group-budget handlers) [5]. Similarly, interrupt handlers cannot currently have their dispatching domains set either.

Implementation of the required support for these facilities is highly dependent on the type of the underlying execution platform. Andy identified four possible scenarios:

Bare machine There is no operating system as such: the Ada run-time environment has full control over interrupts and devices.

Real-time kernel In this case, the kernel, run-time and application all run in the same address space, with same privileges.

Real-time kernel with loadable kernel modules Here all or part of the application can run as a kernel module, hence sharing address space and privileges with the kernel.

User-mode application on top of a real-time kernel The Ada run-time environment and the application run in separate address space to the kernel and run with limited privileges.

The chances for a particular implementation to efficiently support event-handler affinity will very much depend on the ability to control the underlying hardware resources. This support is generally limited when the run-time relies on an existing OS kernel, but on real-time embedded systems, programmers should have control over (or at least knowledge about) which processor will execute which handlers, so that the associated interference can be bound and properly considered in the schedulability analysis.

The Ada Real-Time Annex philosophy is to allow the specification of packages which, since the Annex is optional, may or may not be supported on all systems. Examples of these are asynchronous task control, EDF scheduling or group budgets. So encouraging development and implementation of essential real-time programming abstractions seems like a reasonable goal.

So the core issue is whether or not the language should define different acceptable implementation approaches so that there is more commonality on the various execution platforms. This would probably only apply to low-level features, such as interrupts, event handlers or representation aspects.

Currently a function exists in package `Ada.Interrupts` to obtain the CPU to which an interrupt is connected. The proposal from Wellings and Burns [5] suggests that, in order to control the execution of interrupt handlers on a multiprocessor system, this package should also provide mechanisms to set the dispatching domain (and potentially an individual processor) for an interrupt. A standard exception should be raised if the operation is not supported on a particular platform. In addition, the package `Ada.Interrupts.Names` should declare standard names for all the reserved interrupts required by the Ada run-time system. For example, clock interrupts that service timing events and those that allow tasks to be released when a delay expires (either relative or absolute). The implementation should also document which reserved interrupts result in which of the event handlers being executed. And for those implementations where extra tasks are introduced to execute the event handlers, those tasks should have the same affinity as the associated interrupt.

The position paper submitted by Sáez et al [3] showed the implementation of timing-event affinities for a particular implementation of Ada on top of Linux. In that implementation, the run-time introduces tasks to execute timing-event handlers. Even though this is not in line with the spirit of timing event handlers, which should ideally be executed by the clock interrupt handler, the proposed experimental implementation takes advantage of this by using one task per processor (using task affinities) so that they execute on a known CPU. In this manner, there is a degree of control about where (on which CPU) the interference of the handler will occur. The paper [3] gives an alternative implementation for `Ada.Real_Time.Timing_Events` to support this feature.

2.1 Summary of Workshop position

The Workshop's conclusions after discussion of this part of the session were:

- Ada interrupt handlers should be able to have dispatching domain (or individual processor) set, with raising a standard exception if this feature is not supportable by the underlying platform.

- It was considered that reserved interrupts could still be invisible, but documentation should indicate which dispatching domain is in charge of executing their handlers. There was however no consensus around this aspect.
- It was agreed that Ada should include mechanisms to set the dispatching domain or processor where a timing-event handler executes. There was no consensus however regarding execution-time and group-budget handlers.
- We need to work on the motivation and vision of the issue, towards the production of a related Ada Issue for its consideration by the Ada Rapporteur Group.

3 Support for the concept of cyclic tasks

Cyclic tasks (both periodic and aperiodic) are fundamental patterns in real-time and embedded systems. There are however no abstractions in Ada to model them, and therefore programmers need to care not only about the logic of their tasks, but also about their particular release mechanisms. There have been contributions in this regard in the last few years. Previous editions of this workshop have considered different versions of a library of standard real-time utilities to capture these patterns. An initial version was proposed in [6], which was later extended to cover multi-moded systems [2]. More recent work, in the context of the Ada-Europe International Conference on Reliable Software Technologies, elaborated on this basis to adapt the library to multiprocessor systems [4]. But until today, the IRTAW has been wary of suggesting language changes in this regard.

The second half of this session aimed at revisiting this language constraint, based on the position paper by Patrick Bernardi [1]. In that paper, Bernardi proposes a cyclic task syntax, which can be either time- or event-triggered. Cyclic tasks may also specify how to handle deadline-miss and budget-exhaustion events by means of exceptions.

3.1 Summary of Workshop position

Upon examination of Bernardi's proposal, the workshop agreed that, after some rework and refinement, this could be a good starting point for a proposal to the Ada Rapporteur Group. Points in favour are the the proposal makes more about the cyclic nature of real-time systems, takes advantage of implementation experience, and naturally resolves initialisation issues, since cyclic tasks can be released after their initialisation phase. The workshop, however, identified several outstanding issues that need clarification or further consideration:

- The proposal suggests that a deadline-miss exception should be raised in a tardy task. This would be an asynchronous exception, which is not supported in Ada.
- The proposal does not cover the handling of minimum inter-arrival time violations for sporadic tasks.
- Need to clarify whether deadlines relate to the design's release time or to the actual release time of the tasks.
- Need to revisit and refine the syntax for releasing an aperiodic or sporadic task.
- Need to consider the situation of a program trying to release a task with periodic behaviour.
- In its current state, the proposal does not allow to identify that a task is cyclic from its specification: one has to read the body.
- Need to assess the flexibility of the model, making sure that all aspects are covered.

The workshop agreed on the need to generate an Ada Issue from a revised version of [1] covering these open issues.

References

[1] P. Bernardi (2015), *Incorporating Cyclic Task Behaviour into Ada Tasks*, Ada Letters vol. 35 (1), pp. 59-73, ACM.

[2] J. Real and A. Crespo (2010), *Incorporating Operating Modes to an Ada Real-Time Framework*, Ada Letters vol. 30 (1), pp. 73-85, ACM.

[3] S. Sáez, J. Real, and A. Crespo (2015), *Implementation of Timing-Event Affinities in Ada/Linux*, Ada Letters vol. 35 (1), pp. 80-92, ACM.

[4] S. Sáez, S. Terrasa, and A. Crespo (2011), *A Real-Time Framework for Multiprocessor Platforms Using Ada 2012*, In S. Romanovsky and T. Vardanega, editors, 16th International Conference on Reliable Software technologies – Ada-Europe 2011, volume 6652, Springer.

[5] A. Wellings and A. Burns (2015), *Interrupts, Timing Events and Dispatching Domains*, Ada Letters vol. 35 (1), pp. 26-31, ACM.

[6] A. J. Wellings and A. Burns (2007), *A Framework for Real-Time Utilities for Ada 2005*, Ada Letters vol. 27 (2), pp. 41-47, ACM.

Simulating Next-Generation Cyber-Physical Computing Platforms

Paolo Burgio
University of Modena and Reggio Emilia, Italy; email : $paolo.burgio@unimore.it$
Carlos Alvarez, Eduard Ayguadé, Antonio Filgueras, Daniel Jimenez-Gonzalez, Xavier Martorell and Nacho Navarro
Barcelona Supercomputing Center, Spain; email : $\{name.surname\}@bsc.es$
Roberto Giorgi
University of Siena, Italy; email : $giorgi@dii.unisi.it$

Abstract

In specific domains, such as cyber-physical systems, platforms are quickly evolving to include multiple (many-) cores and programmable logic in a single system-on-chip, while including interfaces to commodity sensors/actuators. Programmable Logic (e.g., FPGA) allows for greater flexibility and dependability. However, the task of extracting the performance/watt potential of heterogeneous many-cores is often demanded at the application level, and this has strong implication on the HW/SW co-design process. Enabling fast prototyping of a board being designed is paramount to enable low time-to-market for applications running on it, and ultimately, for the whole platform: programmers must be provided with accurate hardware models, to support the software development cycle at the very early stages of the design process. Virtual platforms fulfill this need, providing that they can be in turn efficiently developed and tested in a few months timespan. In this position paper we will share our experience in the sphere of the AXIOM project, identifying key properties that virtual platforms modeling next-generation cyber-physical systems should have to quickly enable simulation-based software development for a these platforms.

1 Introduction

As the technological scaling for semiconductors predicted by Moore's law hit the so-called *power wall*, and energy consumption became a primary concern for the market of electronic devices, computing platforms shifted to many-core heterogeneous designs [1, 2, 3, 4]. These platforms are perfectly suited to meet the requirements especially of next-generation cyber-physical systems (CPS), where a huge number of peripherals interacting with the surrounding environment are coupled to a computing board delivering high performance/watt through many-core SMPs and hardware accelerators. Sensors and actuators will be integrated in the design through *ad-hoc* bridges/circuits, or more flexible reprogrammable logic (e.g., FPGAs), composing a system made of several communicating nodes with one or more centralized controllers running on general purpose SMP cores. Hardware accelerators are application-specific circuits which increase the power efficiency of portions (*kernels*) of applications by orders of magnitude. The consequence is that, today, software developers must write code that runs on multiple cores and uses the hardware resources available in the platform, in a productive and effective manner: extracting the tremendous performance/watt potential of such a complex platform essentially becomes also a software development problem. Dependability is also improved when adopting programmable logic: for example, systems based on programmable logic can execute a function in a deterministic way, without the need of a continuous push-pull to/from caches. Most systems based on caches tend to offer a good average performance but may fail to respect a hard deadline in the worst case. Moreover, if the specific architecture fails, reconfiguration of the FPGA can help. Concepts like Data-Flow Threads (DF-Threads) [5, 6] can enable the repetition of the execution of a failed thread.

Virtual platforms are the key to fight this problem, as they enable fast software prototyping at the very early stages of the design cycle of a board, where hardware is not yet 100% available. Computer architects are well aware of this, and in recent years a number of simulator infrastructures have been developed [7, 8, 9], and eventually commercialized, that model a generic or specialized computing fabric with also high accuracy (e.g., cycle-accurate [9, 10]). Unfortunately, correctly modeling the behavior of an hardware platform is time-costly: fully cycle-accurate simulators[1] can be orders of magnitude slower than the corresponding hardware counterparts [11]. For this reason, recently, some virtual platforms were proposed (such as Qemu [11]), for pure functional simulation. They can be successfully adopted in an initial phase to enable functional testing/debugging of the alpha versions of applications, and to quickly exploring the hardware/software partitioning of applications into kernels. Then, software developers might resort to slower and fully cycle-accurate simulators in advanced stages of debugging, until the first prototypes of the board are available.

In this position paper we describe our preliminary analysis on building a virtual platform for simulating cyber-physical systems, in the context of the AXIOM project [12]. AXIOM explores energy-efficient, many-core platforms for next-generation cyber-physical systems. We will briefly describe the guidelines that drive the development of the AXIOM board, in section 2. In section 3 we decompose a simulator for many-core heterogeneous platforms in its basic building blocks, and for each of them we discuss in detail the main issues in simulating it, and how it can (should) be accurately modeled in the quickest way possible. We will do this bringing our expertise on already existing simulation platforms,

[1]cycle accurate virtual platforms mimic the behavior of each component of a system at every clock cycle

both industry [7, 8] and academical solutions [10]. Finally, section 4 draws some conclusions.

2 Requirements for a cyber-physical system: The Axiom project

We are entering the *cyber-physical age*, where both objects and people will become nodes of the same digital network for exchanging information. This vision is also referred to as *"Internet of Things"* (IoT) becauses the general expectation is that "things" or systems will become somewhat smart as people, allowing a tight system-to-human and device-to-environment interaction. As a consequence, we expect that such cyber-physical systems (CPS) will at least react in real-times, consume the least possible energy for a given task, scale up through modularity, and allow for an easy programmability across performance scaling. The whole set of these expectations impose scientific and technological challenges that AXIOM project (Agile, eXtensible, fast I/O Module[12, 13]) tries to address, exploring new hardware/-software architectures for CPSs.

Communities [14, 15, 16] that are using CPSs are devising more and more the need for more powerful embedded platforms that could be: i) easy programmable through an almost standard software toolchain; ii) be customizable with programmable logic (i.e., FPGAs), iii) be extensible to one or more boards (e.g., two robotic arms that need to be closely synchronized toward a single real-time task); iv) provide an easy way to integrate sensors (e.g., through widely available Arduino [15] shields). Current solutions providing enough energy-efficient computational power for fulfilling this needs are starting to rely more and more on multi- and many-core architectures (e.g., UDOO [14] and RaspberryPi2 [16] rely on a quad-core and GPUs) . For example, some current research projects (such as ADEPT [17] or FP7 P-SOCRATES [18]) are already investigating how to join efforts from the high-performance computing (HPC) and the embedded computing domains, which are both focused on high power efficiency, while GPUs and new dataflow platforms such as Maxeler's [19], or in general FPGAs, are claimed as the most energy efficient.

AXIOM research mainly targets designs coupling power-efficient multiple cores, such as ARM ones, and FPGA accelerators on the same die as in the Xilinx Zynq [1], and produce prototypes of single-board computers, similar to UDOO [14], Arduino [15] and RaspberryPi [16]. This architecture includes capability to high-speed board-to-board interconnects and controllers for commodity CPS peripherals such as *Arduino Shields*. AXIOM partners will start the development using a virtual platform: this paper reports the preliminary results of such investigations. At the same time, the tested parts, when ready, are progressively migrated on the FPGA prototype (a Xilinx ZC-706). As a consequence, the AXIOM project requires a virtual platform which simulates general-purpose cores, programmable logic (for accelerators), and peripherals ASIC circuits that integrate sensors and actuators. Figure 1 shows the scheme of a computing platform including two general-purpose cores, FPGA logics and a few peripherals/sensors connected to it.

From the software viewpoint, the AXIOM system will interact with and react to the surrounding environment by properly managing actions in real-time through an operating system (such as Linux), a well-known parallel programming model:

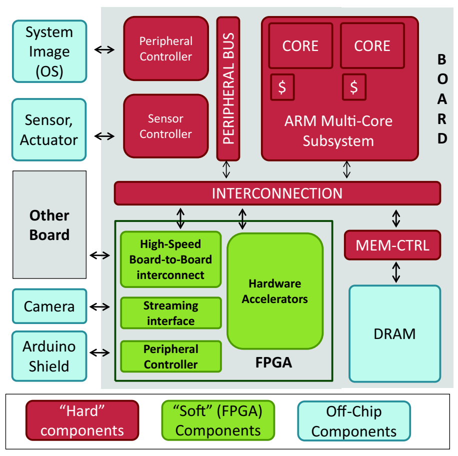


Figure 1: Heterogeneous computing platform with sensors

OmpSs [20]. By using OmpSs, applications will be hardware/software partitioned, i.e., decomposed in parallel tasks that can be mapped on multiple software execution units (OS threads) and/or hardware execution units, e.g. the accelerators in the FPGA. This provides a huge number of options for mapping tasks to resources, considering the device on which a task is mapped, the size of the input data, the data transfer time, or the different speed of the devices in executing the task. Tools and techniques for quickly finding the optimal HW/SW partitioning of applications according to performance and power metrics, and to validate them against real-time constraints, are therefore crucial for the project. The issue is that, when all tasks are mapped either on hardware or software, a complete FPGA synthesis flow for hardware accelerators can spend from hours to days, depending on the size of the computational kernels to process. With virtual platforms, on the contrary, new accelerator elements can be quickly added in the simulation environment, and we can run a timing accurate full system simulation of the applications partitioned on the SMP cores and FPGA accelerators in a matter of minutes to few hours.

3 Virtual Platform requirements

This section describes how to build a virtual platform for a computing system such as the one targeted by AXIOM. Starting from AXIOM specifications, we will first describe its basic building blocks, and discuss how a proper design for each of them will enable fast prototyping of the target board. We will bring our expertise, previously gained using/developing two simulator for heterogeneous systems, namely COTSon [7] in the TERAFLUX project [21, 22], and a prototype built after the open-source academical VirtualSoC [9] by University of Bologna: HC-VSoC [10, 23]. We will also refer to other existing simulator infrastructure of potential interest.

From AXIOM specifications, the simulator must enable quick software prototyping of a system *whose hardware architecture is not 100% defined at early stages of the project*. We identify these four key requirements:

1)immediate availability of at least a first functional version of the simulator, to let the software development cycle start;
2) possibility of defining architectures and their timing model for cycle accurate evaluations, to be selectively used in combination with functional models;

3) the virtual platform must be capable of integrating multiple modules (such as proximity sensors), that *generate/simulate events coming from the surrounding environment*, hence whose behavior must be random, or driven by user/parametrizable;
4) we must be capable of easily putting new hardware models in the design, and to replace (fast and inaccurate) behavioral models of its components, e.g., sensors and actuators, with more timing-accurate (and slower) versions.

Requirements 1) and 2) match the experience of the COTSon [7] simulator, while requirements 3) and 4) match the experience of the HC-VSoC [10, 23] simulator, which models platforms with user-defined hardware accelerators called HWPUs, i.e., whose functionality is defined by the end-user. HC-VSoC architects reduce the problem complexity by specifying a clearly-defined common communication contract and infrastructure for all the blocks modeled in the systems. SMP cores and HWPUs are equipped with a memory shell that supports that communication protocol. Also the COTSon [7] virtual platform, whose primary design requirement was to build a highly scalable architecture, employs a similar solution, providing a well-defined communication API.

This brings us to the first component of a virtual platform, the **simulation engine**, which supports/simulates the interaction between modeled blocks. A number of tools for this exist, both coming from industry and academia, and the most known is probably SystemC [24] by Accelera. The SystemC package is a very flexible macro library (C++ language) and a simulation engine, to simulate the behavior of hardware blocks with different levels of abstraction and accuracy, from RTL- to cycle-accurate, to transactional-level modeling. Higly-scalable infrastructures (such as – as explained before – COTSon [7], or OVP [8]) expose a very simple API to integrate hardware blocks in their engines, and come with a few pre-built architecture models. These are the best solution if the architecture models included in these packages partially or totally match the one being developed.

An second component that must be carefully designed at early stages is the **interconnection**, which emulates on- or off-chip connectivity. Designing an interconnection infrastructure with acceptable good tradeoff of simulation speed, scalability, and timing accuracy is not trivial and it is probably the most time-consuming part in developing virtual platforms. In addition, in AXIOM, the interconnection must enable fast integration of the future versions of the hardware models, to meet requirement 4). An example of scalable communication *medium* is the one connecting multiple COTSon [7] nodes, or the one of HC-VSoC [23], whose protocol is called PINOUT. The difference between them is in the way they are implemented: in the former it is exposed as a pluggable model rigorously decoupled as a functional model and a timing model. Hence the simulated hardware blocks access to the interconnection by directly invoking a simple given API. The latter is itself an instantiated as a SystemC modules with its own model of hardware ports, and a timing accuracy given by design. An amenable property of a simulated interconnection (although not a critical requirement for AXIOM), is that it should be possible to customize its internals and the modeled communication delay should be driven, e.g., *via* simulator parameters or configuration files. An example of configuration files for a simulator is shown in Figure 2. It was developed in the PREDATOR FP7 project [25].

Rows and columns in the figure simulate a hierarchical crossbar by specifying the communication delay among each master port (e.g., SMP cores) and each slave ports (memories and peripherals), respectively. In this example, we are modeling an N-to-M crossbar with user-defined delay for each master-to-slave (core-to-mem bank) path: for instance, we note that the M-th slave implements the controller for an off-chip DDR, because the delay that each master "sees" to get to it (specified in the last column) are significantly higher than for other memories.

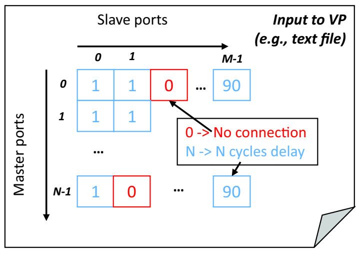


Figure 2: Parametrizable interconnection model

General Purpose cores are the most complex component of a simulator, as they must correctly mimic the functional behavior, e.g., of modern superscalar cores, with branch-prediction units and multiple deep pipelines, or the complex hierarchical shared cache systems and prefetch buffers of next-generation many-core architectures. Luckily, the choice of the instruction-set architecture (ISA) and core model to adopt is usually made at the very beginning of the project, and it does not change in the following. Moreover, most of the simulation infrastructures provide a portfolio of processor models, which is often freely available as a library (see for instance OVP [8]).

The key point in integrating core models in a bigger design is that, in order to support the development of software, *each core model must come with the required toolchain for compiling the code of applications, deploying them on the simulator and – possibly – to support debugging* to do what ultimately is the main purpose of a virtual platform: support software development. In AXIOM, this is reflected in requirement 1).

A few examples can be:

- COTSon [7], which includes x86_64 processor models together with the associated toolchain;
- The HC-VSoC package [10, 23] targets for ARM-based embedded systems, and it comes with a "standard" GNU Compiler Collection (GCC [26]) cross-compiled for it;
- Open Virtual Platforms by Imperas [8] provides a wide portfolio of core models, including ARM (32 and 64 bit), Imagination MIPS (32 and 64 bit), PowerPC, Xilinx Microblaze, and many more.

A project can also adopt a proprietary ISA from a specific provider: they also usually come with a library/software package that simulates a single processing core, using "open" simulation engines (e.g., SystemC), or again with in-house engines or define ISEs (ISA Extensions).

Due to its complexity, the processor model is usually the component from which the development of a virtual platform starts, together with the simulating engine. For this reason, the preliminary version of the platform provided to programmers typically embeds only one or multiple SMP cores, the interconnection model, and a few memories, with limited set of peripherals. Using this, software developers for an heterogeneous platform (such as AXIOM's) can immediately compile, deploy and test the "host/SMP part" of their code.

Programmable logic and peripherals (and sensors/actuators). The platform template targeted by AXIOM embeds on-chip programmable logic, as well as a number of peripherals controllers to interact, e.g., with sensors or Arduino shields. Once the communication infrastructure has been set, and a scalable model of the on-chip interconnection implemented as explained before, it is extremely easy to include in the simulator in-house customized models for peripherals and hardware accelerators. For instance, the PINOUT interface in [23] is implemented in the so-called COMU of HC-VSoC HWPUs. Internally, each of HWPU model can be implemented with a different simulation speed/timing accuracy tradeoff, as required by project specification.

Integration with external components. In the AXIOM project, peripheral components will either interact with the surrounding environment, or connect the board to COTS components or 3rd party subsystems such as the Arduino Shields, and the virtual platform will simulate these behaviors. In the first case, we can employ parameters or proper input files for the simulator that mimic the surrounding environment. For instance, the behavior of temperature sensors can be easily defined *via* simple input text files describing the variation of the temperature in time. The second scenario, in turn, has a great impact on the simulation infrastructure, and raises a potential problem. Simulator developers might need to integrate pre-existing models of the two platforms (e.g., the core model running on SystemC, and the model of an Arduino running on a proprietary simulation engine), which are potentially not designed to communicate each other, or can even be written in different programming language. This possible incompatibility in the communication between simulator models, may require to implement stub functions to transform the information between formats understood by the two components.

Memories. In current virtual platforms, typically memories are implemented as "wrappers" that simply add a delay for accessing big arrays of data modeling the memory banks. For this reason, it is not uncommon that virtual platform developers create their in-house simulation models of memories, when possible. More complex or "fancy" memory models, such as smart memories, can be easily implemented starting from these components.

Support software libraries Applications running on the simulator might employ specific standard libraries, such as `libc` and `libsdtc++`, or custom runtimes, such as `nanos++` [27], or `libhwpu` (in HC-VSoC) to do their work. This is also the case of AXIOM. In this case, the simulator infrastructure must support the same set of required APIs as the "real" board, to ensure code portability.

4 Conclusions

We presented in this paper the approach used by the AXIOM project for flexibly simulating a realistic Cyber-Physical System, soon to be implemented as single board computer. Mainly, besides a FPGA prototype, we developed the preliminary steps through virtual platforms. In particular two platforms had been selected: COTSon and HC-VSoC as they can provide the best support for our design needs. In particular, the inclusion of FPGA in the simulation toolchain provides support for exploring dependability options.

5 Acknowledgment

This work is supported by the AXIOM project, funded by EU H2020 program (grant ICT-01-2014 GA 645496), the Spanish Government, through the Severo Ochoa program (grant SEV-2011-00067) the Spanish Ministry of Science and Technology (TIN2012-34557) and the Generalitat de Catalunya (MPEXPAR, 2014-SGR-1051). Authors would also like to thank the anonymous reviewers for their precious feedback.

References

[1] Xilinx Inc., *Zynq Series*.

[2] G. Kyriazis (2012), *Heterogeneous System Architecture: A Technical Review*.

[3] AMD, *The AMD Fusion Family of APUs*.

[4] R. Giorgi (2015), *Scalable embedded systems: Towards the convergence of high-performance and embedded computing*, in Proceedings of the 13th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing.

[5] R. Giorgi and P. Faraboschi (2014), *An Introduction to DF-Threads and their Execution Model*, in IEEE Proceedings of MPP-2014, (Paris, France), pp. 60–65.

[6] S. Weis, A. Garbade, B. Fechner, A. Mendelson, R. Giorgi, and T. Ungerer (2014), *Architectural support for fault tolerance in a teradevice dataflow system*, Springer International Journal of Parallel Programming, pp. 1–25.

[7] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega (2009), *COTSon: Infrastructure for Full System Simulation*, SIGOPS Oper. Syst. Rev., vol. 43, pp. 52–61.

[8] Imperas Software, *OVP – Open Virtual Platforms*.

[9] D. Bortolotti, C. Pinto, A. Marongiu, M. Ruggiero, and L. Benini (2013), *VirtualSoC: A Full-System Simulation Environment for Massively Parallel Heterogeneous System-on-Chip*, in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International, pp. 2182–2187.

[10] P. Burgio, A. Marongiu, D. Heller, C. Chavet, P. Coussy, and L. Benini (2012), *OpenMP-based Synergistic Parallelization and HW Acceleration for On-Chip Shared-Memory Clusters*, in 15th Euromicro Conference on Digital System Design, DSD 2012, Cesme, Izmir, Turkey, pp. 751–758.

[11] F. Bellard (2005), *QEMU, a Fast and Portable Dynamic Translator*, in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, (Berkeley, CA, USA), pp. 41–41, USENIX Association.

[12] D. Theodoropoulos *et al.* (2015), *The AXIOM project (agile, extensible, fast i/o module)*, in IEEE Proceedings of the 15th International Conference on Embedded Computer Systems: Architecture, MOdeling and Simulation.

[13] C. Alvarez, E. Ayguade, J. Bueno, A. Filgueras, D. Jimenez-Gonzalez, X. Martorell, N. Navarro, D. Theodoropoulos, D. N. Pnevmatikatos, C. Scordino, P. Gai, C. Segura, C. Fernandez, D. Oro, J. R. Saeta, P. Passera, A. Pomella, A. Rizzo, and R. Giorgi (2015), *The AXIOM software layers*, IEEE Proceedings of the 18th EUROMICRO-DSD, pp. 117–124.

[14] UDOO, *Android Linux Arduino in a tiny single-board computer*.

[15] M. Banzi (2008), *Getting Started with Arduino*. Sebastopol, CA: Make Books - Imprint of: O'Reilly Media.

[16] The Raspberry Pi Foundation., *The Raspberry Pi Board*.

[17] The ADEPT Consortium, *ADEPT – Addressing Energy in Parallel Tehcnologies*. [Online]. Available: http://www.adept-project.eu/.

[18] L. M. Pinho, V. Nélis, P. M. Yomsi, E. Quiñones, M. Bertogna, P. Burgio, A. Marongiu, C. Scordino, P. Gai, M. Ramponi, and M. Mardiak (2015), *P-SOCRATES: a Parallel Software Framework for Time-Critical Many-Core Systems*, Microprocess. Microsyst., vol. 39, no. 8, pp. 1190–1203. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2015.06.004.

[19] Maxeler Technologies, *MPT Hardware*.

[20] J. Bueno, L. Martinell, A. Duran, M. Farreras, X. Martorell, R. M. Badia, E. Ayguade, and J. Labarta (2011), *Productive Cluster Programming with OmpSs*, in Proceedings of the 17th International Conference on Parallel Processing - Volume Part I, Euro-Par'11, (Berlin, Heidelberg), pp. 555–566, Springer-Verlag.

[21] R. Giorgi *et al.* (2014), *TERAFLUX: Harnessing dataflow in next generation teradevices*, Microprocessors and Microsystems, vol. 38, no. 8, Part B, pp. 976 – 990.

[22] R. Giorgi and A. Scionti (2015), *A scalable thread scheduling co-processor based on data-flow principles*, ELSEVIER Future Generation Computer Systems, pp. 1–10.

[23] P. Burgio, A. Marongiu, P. Coussy, and L. Benini (2014), *A HLS-Based Toolflow to Design Next-Generation Heterogeneous Many-Core Platforms with Shared Memory*, in 12th IEEE International Conference on Embedded and Ubiquitous Computing, EUC 2014, Milano, Italy, pp. 130–137.

[24] Accelerat Systems Initatives, *SystemC*.

[25] P. Burgio, M. Ruggiero, and L. Benini (2010), *Simulating Future Automotive Systems*, tech. rep., DEIS - University of Bologna.

[26] The Free Software Foundation, *GCC – The Gnu Compiler Collection*.

[27] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas (2011), *OmpSs: A proposal for programming heterogeneous multi-core architectures*, Parallel Processing Letters, vol. 21, pp. 173–193.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU 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

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

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

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch