ADA USER JOURNAL

Volume 34
Number 3
September 2013

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

This September issue of the Ada User Journal closes the publication of the Ada 2012 Rationale chapters, with a last installment, an epilogue, which summarizes many general issues which were considered (or not considered) in the standard revision. I would like to thank John Barnes for producing this valuable guide of the changes in the latest version of the Ada standard. Ada-Europe is now taking the necessary steps to publish the consolidated chapters in a volume of the Lecture Notes on Computer Science series.

The issue follows by publishing a contribution originating from the Industrial Track of the Ada-Europe 2013 conference. In this paper, Daniel Bigelow, from Bigelow Informatics, Switzerland, presents a strategy to convert a large application code from another compilation system to GNAT. The Industrial Track of Ada-Europe conferences provides very valuable work, which the Journal promotes to be extended to regular papers. We are looking forward to present to our readers with more inputs from the conference in later issues.

Also coming from the Ada-Europe 2013 conference, we publish a report on the panel that analyzed the use of heap technologies in real-time systems. The panel included three specialists on the topic: Ludovic Gauthier, from Atego Systems, Inc., USA; S. Tucker Taft, from AdaCore, USA; and James Hunt, from aicas GmbH, Germany, and was moderated by Erhard Plödereder, from the University of Stuttgart, Germany. This paper provides a report of the presentations and discussion of the session, from the session rapporteur, Jørgen Bundgaard.

Following the publication in the June issue of the session reports from the 15th International Real-Time Ada Workshop, which took place in 2011, in this issue we present a summary, provided by Alan Burns, the workshop chair, of the 16th edition of the workshop, which took place in York, UK, last April. We plan to provide our readers with the more detailed session summaries of the workshop, in a forthcoming issue of the Journal.

As usual, the issue also presents the news digest and calendar sections, by their respective editors, providing the readers with a review to the world of Ada, and reliable software in general. The forthcoming events section provides the advance program for the forthcoming SIGAda International Conference on High-Integrity Language Technology (HILT) that will take place November 10-14 in Pittsburgh, USA, and the call for papers for the 19th International Conference on Reliable Software Technologies - Ada-Europe 2014, taking place in Paris, France, June 23-27, 2014. Finally, the Ada Gems section provides a gem on some potentially unexpected behavior on the update of variables, by Emmanuel Briot and Robert Dewar, of AdaCore.

Luís Miguel Pinho
Porto
September 2013
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 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-Belgium Spring 2013 Event

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Sun, 23 Jun 2013 11:33:28
Subject: Ada-Belgium Spring 2013 Event, Sat 29 June 2013
Newsgroups: comp.lang.ada, fr.comp.lang.ada,be.comp.programming

----

Ada-Belgium Spring 2013 Event

Saturday, June 29, 2013, 12:00-19:00

Leuven, Belgium

including at 15:00

2013 Ada-Belgium General Assembly

and at 16:00

Ada Round-Table Discussion

<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/local.html>

Announcement

The next Ada-Belgium event will take place on Saturday, June 29, 2013 in Leuven.

For the sixth year in a row, Ada-Belgium decided to organize their "Spring Event", which starts at noon, runs until 7pm, and includes an informal barbecue, a key signing party, the 20th General Assembly of the organization, and a round-table discussion on Ada-related topics the participants would like to bring up. Afterwards, those interested can once more get practical hands-on experience on packaging Ada software for Debian with Ludovic Brenta, principal maintainer of Ada in Debian.

Schedule

----

* 12:00 welcome and getting started (please be there!)
* 12:15 informal barbecue
* 14:45 key signing party
* 15:00 Ada-Belgium General Assembly
* 16:00 Ada round-table + informal discussions
* 19:00 end

Participation

Everyone interested (members and non-members alike) is welcome at any or all parts of this event.

For practical reasons registration is required. If you would like to attend, please send an email before Wednesday, June 26, 21:00, to Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> with the subject "Ada-Belgium Spring 2013 Event", so you can get precise directions to the place of the meeting. Even if you already responded to the preliminary announcement, please reconfirm your participation ASAP.

If you are interested to become a new member, please register by filling out the 2013 membership application form[1] and by paying the appropriate fee before the General Assembly. After payment you will receive a receipt from our treasurer and you are considered a member of the organization for the year 2013 with all member benefits[2]. Early renewal ensures you receive the full Ada-Belgium membership benefits (including the Ada-Europe indirect membership benefits package).

As mentioned at earlier occasions, we have a limited stock of documentation sets and Ada related CD-ROMs that were distributed at previous events, as well as back issues of the Ada User Journal[3]. These will be available on a first-come first-serve basis at the General Assembly for current and new members. Ada-Belgium sponsor AdaCore provided us some Ada books, and we'll organize a small raffle to hand them out to interested members.

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium/forms/member-form13.html

[2] http://www.cs.kuleuven.be/~dirk/ada-belgium/member-benefit.html

[3] http://www.ada-europe.org/auj/home/

Informal barbecue

The organization will provide food and beverage to all Ada-Belgium members. Non-members who want to participate at the barbecue are also welcome: they can choose to join the organization or pay the sum of 15 Euros per person to the Treasurer of the organization.

Note: if spring would still not have arrived yet in Belgium at this (theoretical) summer day (read: if heavy rain is expected), the barbecue might be replaced with an alternative; but rest assured: food and drinks will be available!

General Assembly

All Ada-Belgium members have a vote at the General Assembly, can add items to the agenda, and can be a candidate for a position on the Board [4]. See the separate official convocation [5] for all details.

[4] http://www.cs.kuleuven.be/~dirk/ada-belgium/board/

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/13/130629-abga-conv.html

Key Signing Party

Wouldn't it be nice if a majority of people used GPG to sign their email every day so that you could send all non-signed email into the spam bin? To make that dream come true, please join and expand the global Web of Trust[6]!

What you should bring with you:

* an official ID card issued by your national government;

* your GPG key fingerprint (i.e. the output of gpg --fingerprint) on small paper slips; a dozen copies or so should be enough.

What you will go home with:

* signatures from all other participants;

* automatic inclusion in the global Web of Trust;

* the ability to digitally sign or encrypt anything you like.

[6] http://en.wikipedia.org/wiki/Web_of_Trust

Ada Round-Table Discussion

This year, we plan to keep the technical part of the Spring event informal as well. We will have a round-table discussion on Ada-related topics the participants would like to bring up. We invite everyone to briefly mention how they are using Ada in their work or non-work environment, and/or what kind of Ada-related activities they would like to embark on. We hope this might spark some concrete ideas for new activities and collaborations.

Afterwards, those interested can get practical information and hands-on experience on "Packaging Ada Software for Debian" [7][8]. See the event's web page for more info.

[7] http://www.debian.org/

[8] http://people.debian.org/~lbrenta/debian-ada-policy.html

[…]

[See also “Ada-Belgium Spring 2012 Event”, AUJ 33-2, p. 73. —sparre]

GNAT Industrial User Day

From: Jamie Ayre <ayre@adacore.com>
Date: Tue, 9 Jul 2013 11:46:43 +0200
Subject: [AdaCore] GNAT Industrial User Day 2013
To:"libre-news@lists.adacore.com

AdaCore is once again happy to invite you to join us for the GNAT Industrial User Day that will take place in Paris on September 25th, 2013.

This year's event will provide information on the recent evolutions in the Ada and SPARK languages, important upgrades to GNAT Pro and complementary technologies and roadmaps that will all help you get fully up-to-date with our technology.

Specific sessions include:

- Writing reliable software: formal verification and static analysis techniques and possibilities.
- Qualimetrics: a software development dashboard.
- A new generation of toolsets: GPS 6 (IDE presentation, Gtk 3), GNAT Tracker 3.
- Ada development on ARM processors.
- The latest and greatest (and future developments) in GNAT Pro: (GPRbuild, GNAT2XML, new ports, GNATcoverage/GNATemulator, Qualifying Machine, …).
- AdaCore partner presentations.

Many members of AdaCore's technical staff will be present and will be happy to discuss any questions you may have on Ada, SPARK, and AdaCore products.

For the full agenda and to register, please visit:

http://www2.adacore.com/gnatpro-day

Ada 2012 talk at DANSAS'13

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu, 15 Aug 2013 10:39:59 +0200
Subject: Ada 2012 talk at DANSAS'13
Newsgroups: comp.lang.ada

I've gotten a talk on Ada 2012 accepted for the Danish Static Analysis Symposium (DANSAS'13) Friday August 23rd in Odense. The title and abstract are:

Contract-based Programming with Ada 2012 - an experience report.

The 2012 version of the Ada programming language standard includes checked "contracts" and "aspects" for subprograms and types. Some of these are by definition checked at compile-time, while other checks can be postponed to run-time, if a static analysis is unfeasible (or just not implemented).

At AdaHeads, we are currently developing a hosted telephone reception system, where the core component is written in Ada 2012. We picked Ada 2012 specifically to be able to use the contracts and aspects to increase our confidence that the software is correct.

Our experience so far is that GNAT-GPL-2013 (the only generally available Ada 2012 compiler) only implements static (compile-time) checking of contracts and aspects where it is required by the language standard. This means that for now, the big static analysis benefits of using Ada are related to the basic type system, which also existed in earlier versions of the standard, and the major benefit of switching to Ada 2012 at the moment is in the improved run-time checks.

General information on DANSAS'13 can be found at:

http://dansas.sdu.dk/2013/

Ada and Education

On Teaching Types

From: Mike Hopkins <postmaster@ada-augusta.demon.co.uk>
Date: Sun, 11 Aug 2013 16:15:59 +0100
Subject: Re: 4 beginner's questions on the PL Ada
Newsgroups: comp.lang.ada

Once again a thread in this news group shows a polarisation of mind sets concerning right and wrong approaches to writing a program. I am reminded of my teaching days when, even before writing a single piece of Ada code on the white board, I would initiate some class discussion on the question of whether time and duration are same and, if they are not the same, does it matter. I knew I could expect general agreement that the result of adding a pair of time variables was meaningless whereas subtracting such pair of times could be valid, but only if one was aware that the result was not a time. The fun would start when the discussion moved on to questions of how one might detect or, better still prevent, a time/duration program error. If I was lucky, opinions would become quite heated concerning personal responsibility and managerial responsibility should such an error reach a production version of a product.

When the dust began to settle I would ask whether it might not be useful if such an error could be detected at compilation time. Sometimes there would be a small minority who would greet that question with incredulity. More interesting to me was identifying those who showed an interest in how this might be achieved in well written Ada and, in comparison, those who were sufficiently sure of their own capabilities that they could regard any additional lines of defensive coding in any language as an unnecessary irrelevance. Worse still were those who would be deaf to the idea that although additional lines of declaration source code can be expected to change the resulting executable code that does not necessarily mean a change of the amount of generated execution code nor necessarily a change of execution times.

Ada-related Resources

Experimental Continuous Integration System for Open Source Projects

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon, 12 Aug 2013 08:40:00 +0300
Subject: Experimental Continuous Integration system for open source Ada projects
Newsgroups: comp.lang.ada

I have setup an experimental continuous integration system for open source Ada projects at

http://build.ada-language.com/

The system builds selected set of projects in regular intervals using 3 different Ada compilers (GNAT, Janus/Ada, ICCAda) on two platforms (Windows 7, Debian/amd64 7.0).

The idea is to see how well the projects can be built with different compilers and to catch changes which break portability.

For now, I have included only projects which are known to be portable across compilers and which have public source code repository available.

The system is implemented by running Jenkins [1] on a cheap "lowend" virtual private server (from waveride.at), so there are no availability or uptime guarantees and the server might get wiped out at any moment (I do backups and the server + connections have been stable for a month or so, but still…)

If you want to get your project listed and built, please send me an email.

And to get useful results, please make sure that your project doesn't use any GNAT.* packages or GNAT-specific features (like 'Img). Keeping external dependencies in minimum also helps.

For security reasons, I don't allow any builds commands (like "make") to be run. Instead, I include all build commands directly to the Jenkins, so I know what is executed. (So, if your project source files need some specific treatment and cannot be compiled with "gnatmake mainprocedure", please mention that.)

[1] http://jenkins-ci.org/

SPARK 2014

From: AdaCore & Altran
Date: Wed Aug 14 2013
Subject: SPARK 2014
URL: http://www.spark-2014.org/

[Papers and reference information on the upcoming version of SPARK. —sparre]

[See also the discussion "The Future of SPARK (and Ada)" in the "Ada in Context" section. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon Aug 19 2013
Subject: Repositories of Open Source software
To: Ada User Journal

AdaForge: 7 repositories [1]

Bitbucket: 53+ repositories [2,3]

Codelabs: 17 repositories [4]

GitHub: 384 repositories [5]

97 developers [6]

Rosetta Code: 570 examples [7]

25 developers [8]

Sourceforge: 224 repositories [9]

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

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

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

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

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

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

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

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

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

[See also "Repositories of Open Source software", AUJ 34-2, p. 65. —sparre]

Ada-related Tools

Ada-Fuse

From: Nicolai Ruckel <nicolai.ruckel@uni-weimar.de>
Date: Fri Apr 19 2013
Subject: Ada-Fuse
URL: https://github.com/RanaExMachina/ada-fuse

Ada-Fuse provides Ada bindings for Fuse. Our goal was to make it possible to use the Fuse operations with Ada-like types and functions.

You can use most of the Fuse operations. The missing operations are `lock`, `utimens`, `bmap`, `ioctl` and `poll`. For most filesystems this should not be a problem, in fact we never saw a Fuse-Filesystem using these operations. Be aware that not all of the implemented functions are tested. Untested functions are marked in the source.

Ada-Fuse should work on 32bit and 64bit Linux and Mac OS. See "Known limitations / bugs" for more info.

[…]

Ada 2005 Math Extensions

From: Simon Wright <simon@pushface.org>
Date: Wed, 29 May 2013 19:43:28 +0100
Subject: ANN: Ada 2005 Math Extensions 20130529
Newsgroups: comp.lang.ada

Available at https://sourceforge.net/projects/gnat-math-extn/files/20130529/

Packages Ada_Numerics.Float_Arrays and .Long_Float_Arrays are provided; they are instantiations of .Generic_Arrays for the standard Float and Long_Float types.

The packages are declared Pure.

The code is compatible with GNAT GPL 2013 (a minor change was required to avoid a compilation warning).

[See also "Math Extensions", AUJ 33-3, p. 143. —sparre]

GNAT GPL

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Wed, 29 May 2013 15:36:31 +0200
Subject: GNAT 2013 is out!
Newsgroups: comp.lang.ada

Just connected to libre and surprise! The site proposed the 2013 edition!

[http://libre.adacore.com/download/configurations —sparre] From: Jamie Ayre <ayre@adacore.com>

Date: Wed, 5 Jun 2013 10:09:07 +0200
Subject: [AdaCore] Announcing the availability of GNAT GPL and SPARK Hi-Lite GPL 2013
To: libre-news@lists.adacore.com

Dear GNAT and SPARK GPL user,

We are pleased to announce the availability of GNAT GPL 2013 and SPARK Hi-Lite GPL. GNAT GPL 2013 provides new Ada 2012 language features, introduces new tools and new versions of existing tools, incorporates a range of improvements and adds several new platforms. Some of the key enhancements:

New Language Features

- Final touches on Ada 2012 support
- Automatic Endianness conversion ('Scalar_Storage_Order)
- Dimensionality checking (new aspects and packages)

New version of existing tools

- GtkAda

Gtk+ version 3 brings new widgets, a CSS-based theming framework, and an improved API that has been clarified and has a more homogeneous naming scheme.

- New versions of IDEs
- GPRbuild
- GDB debugger

Switch to GCC 4.7 back-end

New GNATcheck rules

SPARK Hi-Lite GPL 2013 is a package that can be installed after GNAT GPL 2013, to provide access to the new SPARK toolset that was developed in project Hi-Lite. This release is a major evolution of the SPARK toolset providing formal verification for a subset of Ada programs. This new toolset uses Ada 2012-style contracts (e.g. pre- and postconditions), instead of the stylized comments in previous versions, to provide specifications of programs. The benefits of this new version are:

- larger supported subset of Ada (including generics, discriminants, etc.)
- same contracts used for testing and formal verification
- applicable to units partly in SPARK
- improved automatic proof of complex contracts
- new integration in GPS

A preview of the data and information analysis is also available. For more information on SPARK 2014, please visit www.spark-2014.com.

Both toolsets can be downloaded from libre.adacore.com.

LDAP Client

From: Diogenes <phathax0r@gmail.com>
Date: Thu, 30 May 2013 17:40:02 -0700
Subject: LDAP client/server in Ada?
Newsgroups: comp.lang.ada

Has anyone done this yet?

I need to write an Ada client that speaks the LDAP protocol. The protocol is specified using ASN.1 notation.

I could do the work by hand…done that sort of thing before, but I was wondering if anyone else has done it before. Preferably without needing an ASN.1 compiler.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 31 May 2013 07:04:14 +0200
Subject: Re: LDAP client/server in Ada?
Newsgroups: comp.lang.ada

There is some LDAP support in AWS, not complete though. Check if it fits your needs.

Matreshka

From: Vadim Godunko <vgodunko@gmail.com>
Date: Mon, 3 Jun 2013 10:16:28 -0700
Subject: Announce: Matreshka 0.5.0
Newsgroups: comp.lang.ada

We are pleased to announce next major release of Matreshka framework. It includes new features:

- support to process data in JSON format
- driver for MySQL server

and enhancements of:

- SQLite3 database driver
- WSDL to Ada translator
- text codecs for IBM-437, KOI-8R and KOI-8U

See Release Notes for detailed list of changes:

http://forge.ada-ru.org/matreshka/wiki/ReleaseNotes/0.5

Matreshka 0.5.0 can be downloaded in source code and binary form from page:

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

[See also "Matreshka", AUJ 34-1, p. 8. —sparre]

Paraffin

From: Brad Moore <brad.moore@shaw.ca>
Date: Wed, 19 Jun 2013 00:41:43 -0600
Subject: ANN: Paraffin 4.3, Parallelism Generics
Newsgroups: comp.lang.ada

I am pleased to announce Paraffin 4.3.

Paraffin is a set of Ada 2012 generics that may be used to add parallelism to iterative loops and recursive code. Older releases (prior to 4.0) also support Ada 2005.

Paraffin includes generics for both Ravenscar and non-Ravenscar use. The Ravenscar version utilizes static task pools with dispatching domains intended for real-time programming.

Paraffin also includes Paraffinalia, which is a suit of useful parallel utilities that utilize the Paraffin generics. These include generics for;

1) generic to integrating a function in parallel
2) generic to apply quicksort algorithm in parallel to an array
3) generic to apply fast fourier transform to an array of data.
4) generic Red-Black tree container that performs some operations in parallel.
5) function to solve matrices using Gauss-Jordan Elimination
6) generic to perform prefix sum calculations
7) generic to perform sequence alignment using the Smith-Waterman algorithm to find similar regions between two strings for problems such as comparing genetic nucleotide or protein sequences, or checking for plagiarism between two text sources.

This release has the following notable features;

1) Most importantly, to those who want to compile Paraffin with the latest GNAT 2013 GPL release, this version contains bug fixes that allow compilation.
2) A new Paraffinalia app has been added. This implements the Smith-Waterman dynamic algorithm in parallel. This app performs sequence alignment, which means it may be used to find similar regions between two text strings. Such an algorithm is of interest to genetic comparisons of nucleotide or protein sequences. It may also be used to compare two text documents against each other for plagiarism, etc.
3) A Ravenscar compliant version of the Smith-Waterman app has also been added.
4) Several wait-free barriers have been added. These offer several advantages over the facilities of Ada.Synchronous_Barriers, in that the workers are released in parallel, as opposed to sequentially, for barriers that are implemented as protected objects. In addition, there is no blocking, no queues, and these new barriers are Ravenscar compliant, and objects of these barriers can be declared at nested levels in a Ravenscar application, unlike barriers that are implemented as protected objects. The last point to note is that using these barriers can make a significant improvement in performance. The matrix-solving paraffinalia app has been seen to complete twice as fast in certain circumstances.
5) A new facility has been added to the work sharing loop iteration packages. This is a subprogram, Get_Worker_Id, that allows the caller to statically determine which worker will be assigned to a particular loop iteration number. This is particularly useful for algorithms that use barriers, as typically one needs to know how many workers will be synchronizing on the barrier, as well as to map intermediate user-defined result arrays with worker numbers.
6) The Smith-Waterman app, the matrix solving app, and the histogram cumulative sum paraffinalia apps were modified to use this new facility.

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

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

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

```
mkdir sandbox
cd sandbox
git clone git://git.code.sf.net/p/paraffin/code paraffin-code
```

[See also "Paraffin and Paraffinalia", AUJ 34-2, p. 67. —sparre]

RTEMS and Ada on Raspberry Pi

From: Brian Catlin <brian.catlin@gmail.com>
Date: Wed, 3 Jul 2013 17:42:08 -0700
Subject: RTEMS (and thus Ada) on Raspberry Pi
Newsgroups: comp.lang.ada

It appears that RTEMS has been ported to the RasPi

http://www.raspberrypi.org/phpBB3/viewtopic.php?f=72&t=38962

From: Simon Wright <simon@pushface.org>
Date: Thu, 04 Jul 2013 17:31:28 +0100
Subject: Re: RTEMS (and thus Ada) on Raspberry Pi
Newsgroups: comp.lang.ada

[…]

What I meant was, does the port of RTEMS to the Pi include a BSP so that applications built to RTEMS can run on the bare Pi?

And on re-reading the link, I see that it does [1], though only the timer and the UART are supported so far.

[1] http://alanstechnotes.blogspot.co.uk/2013/03/rtems-on-raspberry-pi.html

AdaControl

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 11 Jul 2013 07:01:46 +0200
Subject: [Ann] New version of AdaControl released
Newsgroups: comp.lang.ada

Adalog is pleased to announce the release of version 1.15r5 of AdaControl, featuring 452 possible checks, including a number of checks for Ada 2012 constructs like expression functions, quantifiers, if/case expressions… and of course “in out” parameters in functions!

As usual, it is available on SourceForge (http://adacontrol.sourceforge.net) or from its home page (http://www.adalog.fr/adacontrol2.htm).

AdaControl is free software (GMGPL) with commercial support available, see User's Guide. Don't hesitate to write to info@adalog.fr for more information on the great benefits of commercial support.

[See also “AdaControl”, AUJ 33-3, p. 146. —sparre]

Qt5Ada

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Fri, 12 Jul 2013 03:09:13 -0700
Subject: Announce: Qt5Ada version 5.1.0 release 10/07/2013 free edition
Newsgroups: comp.lang.ada

Qt5Ada is an Ada 2012 binding to the Qt5 framework (based on Qt 5.1.0 final).

Qt5ada version 5.1.0 open source and qt5c.dll(libqt5c.so) built with Microsoft Visual Studio 2012 in Windows and gcc x86 in Linux.

Package tested with GNAT-GPL-2012 in Windows 32bit and 64bit and Linux x86 Debian 7.

It supports GUI, SQL, multimedia, web, networking and many others things.

Qt5Ada for Windows and Linux (Unix) is available from http://users1.jabry.com/adastudio/index.html

My configuration script to build Qt5 is:

configure -opensource -release -nomake tests -opengl desktop -icu -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -prefix "e:/Qt/5.1"

I have added new packages to support Touch devices, SerialPorts and Sensors.

The full list of released classes is in “Qt5 classes to Qt5Ada packages relation table.pdf”.

[See also “Qt5Ada”, AUJ 34-2, p. 68. —sparre]

Pseudo Random Number Generators

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Fri, 19 Jul 2013 21:28:32 +0200
Subject: “A Comparison of Four Pseudo Random Number Generators Implemented in Ada”
Newsgroups: comp.lang.ada

I was searching the web for simple pseudo-random number generator suitable for Monte Carlo simulation, when I found a paper comparing some PRNG implemented in Ada. Don't know if it well suited for simulation and “calculation” based on random input, but probably always worth to be mentioned here :-p

“A Comparison of Four Pseudo Random Number Generators Implemented in Ada”

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.7884&rep=rep1&type=pdf

William N. Graham

April 1999

Ada source is provided at the end of the document.

From: Shark8 <onewingedshark@gmail.com>
Date: Fri, 19 Jul 2013 14:43:56 -0700
Subject: Re: “A Comparison of Four Pseudo Random Number Generators Implemented in Ada”
Newsgroups: comp.lang.ada

I recall a RNG that was posted [or posted about] here on Comp.Lang.Ada; here's a link to a thread:

https://groups.google.com/forum/?fromgroups#!searchin/comp.lang.ada/RNG/comp.lang.ada/Iy6J3UzDwVQ/fvVkxUQoRfoJ

Its name, if you need to search, was/is: KISS4691

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Sat, 20 Jul 2013 02:41:16 +0200
Subject: Re: ³A Comparison of Four Pseudo Random Number Generators Implemented in Ada²
Newsgroups: comp.lang.ada

> Why not use Ada.Numerics.{Discrete,Float}_Random ?

Also and more basically, it's often suggested to not rely on a single generator and use at least two ones, different enough, for comparison of results, as most papers about Monte Carlo methods introduces this. That's after all just like with hash functions.

You may also favour algorithms (either in mathematical or comprehensible source form) over libraries when, as you say, you want to be able to reproduce an exact same sequence without a record of it (may weight too much, easily some hundreds of MB), or else want the same in different contexts, say Ada and SML without external binding.

And above all, this paper mentions Ada and put it at the front, so that's a lot worthy anyway :-D

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Sun, 11 Aug 2013 14:37:14 -0700
Subject: New RNGs in the PragmARCs
Newsgroups: comp.lang.ada

The beta version of the PragmAda Reusable Components for ISO/IEC 8652:2007 now contains an implementation of Marsaglia's KISS RNG, and a "combined" RNG that combines both the Universal and KISS generators to result in a generator that should be both better quality than either and with a much longer period as well. You can find the PragmARCs at

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

I hope those of you who are into RNGs will take a look and provide feedback.

[See also “PragmAda Reusable Components”, AUJ 34-2, p. 66. —sparre]

AdaControl for FreeBSD

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Tue, 23 Jul 2013 09:35:14 +0200
Subject: AdaControl now officially supported on FreeBSD
Newsgroups: comp.lang.ada

Thanks to John Marino, AdaControl is in the FreeBSD Ports Collection. It is built with lang/gcc-aux which is GNAT FSF 4.7 which is paired with ASIS 2011. So it's the "old" version for now.

http://www.freshports.org/lang/adacontrol/

AdaSockets

From: Freecode <http://www.freecode.com/>
Date: Sun, 4 Aug 2013 11:16:00 +0000
Subject: AdaSockets 1.8.11 has been released [Freecode]
To: Ada User Journal

sigra just announced version 1.8.11 of AdaSockets on Freecode.

The release notes for this version are as follows:

This release uses the right compiler to compile C constants files.

Project description:

AdaSockets is a library that lets you use sockets in Ada 95. It supports unicast and multicast sockets, and uses object oriented structures to ease sockets manipulation.

Detailed history and release notes are available here:

http://freecode.com/projects/adasockets#release_356761

Gate3-in Code Generator for Glade2 and Glade3 GtkBuilder

From: Rob Groen <robgr@xs4all.nl>
Date: Sun, 04 Aug 2013 13:33:01 +0200
Subject: gate3-in code generator (v0.4) for Glade GtkBuilder and LibGlade files
To: gtkada <gtkada@lists.adacore.com>

Last February I "released" version 0.3 of gate3-in, an Ada code sketcher for Glade, based on gate-in which is not maintained anymore. Now I am releasing version 0.4 which sees a number of fixes and enhancements (I hope). The sketcher still targets Gtk2 (Gtkada 2.24) and Glade3 version prior to 3.8.1. When time permits I will move gate3-in to GTK3 (Gtkada 3.x), but this will probably take some time.

The work is based on the sources found in the 2011 GtkAda distribution, notably Glib.Glade, Gtk.Glade and Gtk_Generates. This is work in progress, so not all Glade features are supported (yet). Also, testing is limited to Glade files that I have used in the past and new ones that were created targeting changes that I made, so they probably don't touch all the functionality. Testing has been done on WinXP and on Ubuntu 10.04, using the 2012 GNAT GPL and GtkAda 2.24 distribution. No other prerequisites are known to me.

Download the zip file from: http://robgr.home.xs4all.nl/

Both versions (0.3 and 0.4) can be found there. Zip files include a list of changes. When building gate3-in the gpr file specifies "debug" and "obj" subdirectories relative to the directory where the sources and the gpr file are found. You must create these subdirectories before using gnatmake.

If you have problems downloading, send me an e-mail so I can mail the zip file directly.

Comments are welcome!

[See also “Gate3-in code generator for Glade2 and Glade3 GtkBuilder”, AUJ 34-1, p. 12. —sparre]

Comfignat

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Wed, 07 Aug 2013 19:32:09 +0200
Subject: Introducing Comfignat
Newsgroups: comp.lang.ada

Last Friday I published the first release of Comfignat. Comfignat is common, convenient, command-line-controlled compile-time configuration of software built with the GNAT tools on Unix-like operating systems.

In my work on packaging Ada software in Fedora I have found that most Ada projects have rather inflexible build systems. Makefiles and project files usually have to be modified to meet Fedora’s policies. Files placement is often not configurable enough, and support for multiarch systems and installation to a staging directory is often missing. There is also a lack of naming conventions. In C projects Make variables such as CFLAGS and LDFLAGS are a well established de facto standard. Among Ada projects there is no consensus. The lack of conventions slows packaging down as it takes time to figure out each makefile.

It’s quite understandable that Ada programmers don’t want to write a lot of Make code for every project but rather focus on their Ada programming, but the result is inflexible makefiles that don’t meet users’ and distributions’ needs.

To make my own projects fully configurable, multiarch-capable and stageable while minimizing the amount of Make code that must be written for every new project, I have written Comfignat. It consists of a makefile foundation with generic Make code to be included by each project’s makefile, and an abstract GNAT project file to be imported by each project’s project files. Leveraging GNU Make and Gnatprep, Comfignat adds the flexibility that GNAT project files lack, so that programs and libraries can easily be configured for all sorts of use cases, such as installing locally from source, packaging in a distribution, building relocatable binary packages, or testing and debugging on a developer’s workstation.

As all the code in Comfignat is generic it should be useful in any project that targets GNAT and Unix-like systems, and will greatly reduce the amount of Make code that needs to be written for each project. It works for mixed-language projects as well as pure Ada projects, and Gnatmake and GPRbuild are both supported.

Read more:

https://www.rombobjörn.se/Comfignat/

Download the tarball:

https://www.rombobjörn.se/Comfignat/download/

Browse the code online:

https://gitorious.org/comfignat/comfignat/trees/master

See how my projects use Comfignat:

https://gitorious.org/adamilter/adamilter/trees/master

https://gitorious.org/adamilter/system_log/trees/master

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Mon, 12 Aug 2013 11:03:57 +0200
Subject: Re: Introducing Comfignat
Newsgroups: comp.lang.ada> [...] How do you control what goes to staging directory?

GPRbuild or Gnatmake copies the source files. It's supposed to be only those files that are needed for compiling code that uses the library, that is the specifications of the interface packages and those bodies that contain generics or inlined subprograms. Your build project says “for Library_Interface use ("Ahven");”, so the package Ahven is the only interface package. It contains a generic procedure. Therefore ahven.ads and ahven.adb are staged.

> Also, how do I get documentation (built by a separate Python tool) there (easily)?

Comfignat doesn't know about the Python tool so you'll need to write a rule in your makefile to invoke it. To get the documentation staged you should use the Make variables stage_mandir (for manpages), stage_infodir (for the Info format) and stage_miscdocdir (for other documentation).

Hopefully the tool allows you to specify an output directory, and then you can tell it to write directly to "${stage_miscdocdir}/ahven" for example.

In case the tool is hardcoded to write the files in the source tree, your makefile will have to copy them to the appropriate directories. In that case the tool also doesn't support out-of-tree builds, but will write the files in the source tree even when a separate build directory is used, so you'll be copying from srcdir. The commands might be:

```
mkdir -p "${stage_miscdocdir}/ahven"
cp -RPp ${srcdir}/some/where/* "${stage_miscdocdir}/ahven/"
```

A possible future extension to Comfignat might be additional makefile modules for some popular documentation generators.

Lapack

From: Leo Brewin <leo.brewin@internode.on.net>
Date: Sun, 11 Aug 2013 16:53:29 -0700
Subject: Updated ada-lapack on Sourceforge
Newsgroups: comp.lang.ada

I've updated the ada-lapack library on sourceforge. The library now provides native Ada code for

- Matrix determinant and inverse on general matrices,
- Eigenvalues and eigenvectors of general, real and hermitian symmetric matrices,
- Solutions of systems of equations for general, real and hermitian symmetric coefficient matrices,
- Singular value decomposition for general matrices

New procedures in this release are

syev, syevd, sysv, heev and heevd

(implementing dsyev, dsyevd, dsysv, zheev, zheevd and zsysv).

There are also a collection of functions (and two procedures)

MatrixDeterm,
MatrixInverse,
Eigenvalues,
EigenvaluesRealSymm,
EigenvaluesHermSymm,
Eigensystem,
EigensystemRealSymm,
EigensystemHermSymm,
SolveSystem,
SolveSystemRealSymm,
SolveSystemHermSymm

These provide a more familar Ada style interface to the Lapack routines. There are, as yet, no similar interfaces for the singular value decompostion procedures (gesv,gesdd).

You can find the code at

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

[See also "Lapack", AUJ 34-1, p. 8. —sparre]

Simple Web-based IDE

Subject: Compile and Execute Ada online
Date: Mon Aug 19 2013
From: compileonline</>com
URL: http://www.compileonline.com/compile_ada_online.php

[A web site, where you can compile and test Ada applications on-the-fly. —sparre]

Ada-related Products

Vector Software Announces Support for the AdaCore GNAT Pro Compiler for ARM Cortex

From: Vector Software Press Releases
Date: 24 June 2013
Subject: Vector Software Announces Support for the AdaCore GNAT Pro Compiler for ARM Cortex
URL: https://www.vectorcast.com/news/vector-software-press-releases/2013/vector-software-announces-support-adacore-gnat-pro-compiler

Newest VectorCAST integration strengthens Vector Software's offering for embedded Ada software development

June 24, 2013

Vector Software, the leading provider of dynamic software testing solutions for embedded systems, today announced VectorCAST support for AdaCore's GNAT Pro Safety-Critical product for ARM micro-controllers.

The AdaCore GNAT Pro Safety-Critical application provides a complete Ada development environment, oriented towards systems that have safety-critical or stringent memory constraints requirements.

ARM is a popular low-cost, low power microprocessor that is growing in popularity in industries like aerospace, defense, and transportation.

Vector Software's VectorCAST embedded software testing platform, is a family of products that automates testing activities across the software development lifecycle and supports C, C++, and Ada. VectorCAST includes a suite of Ada test tools that significantly reduces the time, effort, and cost associated with testing safety-critical software written in Ada.

Support for ARM by AdaCore and VectorCAST allow organizations developing safety-critical applications for ARM in Ada, or a combination of Ada, C, and C++, to have a complete Ada development and automated testing environment. In addition, the VectorCAST platform supports AdaCore's customized run-time profiles including: ZFP, Cert, and Ravenscar.

"Ada has long been recognized for its strong software engineering benefits including portability, reliability and maintainability," said Jamie Ayre, Marketing Director at AdaCore. "We are delighted that Vector Software has integrated its industry-leading VectorCAST suite with AdaCore's GNAT Pro Safety Critical product for ARM."

"This new integration demonstrates our commitment to providing the Ada development community a complete safety-oriented development toolset on a large range of targets," said William McCaffrey, Chief Operating Officer at Vector Software. "Customers can now benefit from the richness of the hardware platforms used by the wider market beyond safety-critical systems."

GNAT Pro for Wind River Linux

From: AdaCore Press Center
Date: Tue Jul 2 2013
Subject: AdaCore Brings Ada to Wind River Linux
URL: http://www.adacore.com/press/adacore-brings-ada-to-wind-river-linux/

GNAT Pro 7.1 now on Wind River Linux, with full Ada 2012 support

STUTTGART, NEW YORK and PARIS, July 2, 2013 – Embedded Konferenz – AdaCore today announced the availability of the GNAT Pro Ada development environment on the Wind River Linux platform. This new implementation continues a long, successful relationship between AdaCore and Wind River, marked by hundreds of joint customers worldwide, and brings the Ada language's reliability benefits to the increasingly popular Wind River Linux platform. AdaCore offers the industry's leading Ada solution for Wind River's products, including a GNAT Pro implementation for Wind River's VxWorks® real-time operating system (RTOS).

Wind River Linux is the market-leading commercial grade Linux solution for embedded device development. It features an optimized run-time; a flexible, scalable build system; pre-integrated middleware packages for specific device types; an integrated development environment; and a suite of professional open source tools, adapted and extended for embedded development.

Programmers can use AdaCore and Wind River products together to develop applications that freely combine modules in Ada, C and C++, and can manipulate and analyze Ada applications through Wind River's Linux browser and tools. Furthermore, this new implementation of GNAT Pro on Wind River Linux supports all versions of Ada (Ada 2012 / 2005 / 95 / 83) and is tightly integrated into the Wind River Workbench development environment.

"AdaCore and Wind River share many of the same goals in embedded software development: reliability, performance and portability," explains AdaCore product manager Dr. Pat Rogers. "We're very excited that this latest port of our technologies to Wind River Linux continues to build on our strong relationship and offers our joint customers a highly sophisticated and efficient software development process."

"This new integration brings the Ada development community everything they need to build and support highly differentiated solutions and deploy on an industry leading commercial grade Linux solution, based on the Yocto Project open source development infrastructure," said Davide Ricci, Product Line Manager at Wind River. "With AdaCore's long history in providing solutions for high-integrity applications such as in the aerospace and defense industry, the combination of Wind River Linux and GNAT Pro provides embedded developers with powerful capabilities during the development process of secure applications."

GNAT Pro for Wind River Linux includes support for the Wind River Linux 4.3 platform, the PowerPC and Power PC e500v2 target platforms, and the Linux host.

Ada and GNU/Linux

GtkAda in Fedora

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Mon, 22 Jul 2013 12:09:55 +0200
Subject: The status of GTKada in Fedora
To: gtkada@lists.adacore.com

> […]

Fedora 18 and later has GTKada 2.24.2. Fedora 17 has GTKada 2.18.0. I should look into packaging GTKada 3 but I haven't had time for that yet. Would you like to help?

AVR-Ada for Fedora

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Fri, 26 Jul 2013 21:56:19 +0300
Subject: AVR-Ada 1.2.2 RPMs for Fedora 19
To: AVR-Ada <avr-ada-devel@lists.sourceforge.net>

Just in case someone doesn't follow my Twitter feed, reddit/r/ada, comp.lang.ada, or planet.ada.cx, same info here:

I build AVR-Ada 1.2.2 RPM packages for Fedora 19. They are available from my personal fedora.ada-language.com RPM repository.

The package sources consist of 1.2.2 release sources + 2 patches from me. First patch reverts UART back to AVR-Ada 1.2 version, and another fixes libavrada.a linking problems so that board/mcu specific stuff is built and linked correctly.

More details at http://arduino.ada-language.com/avr-ada-122-rpms-for-fedora-19.html.

Ada and Mac OS X

GNAT

From: Simon Wright <simon@pushface.org>
Date: Sun, 07 Jul 2013 19:37:02 +0100
Subject: GCC 4.8.1 for Mac OS X
Newsgroups: comp.lang.ada

You can find this at

https://sourceforge.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/4.8.1/

The README says:

This is GCC 4.8.1 built for Mac OS X Mountain Lion (10.8.4, Darwin 12.4.0).

gcc-4.8.1-x86_64-apple-darwin12.tar.bz2
=====================================

Compilers included: Ada, C, C++, Objective C, Objective C++, Fortran.

Tools included: ASIS, AUnit, GPRbuild, GNATColl, XMLAda from GNAT GPL 2013.

Target: x86_64-apple-darwin12

Configured with:

```
../gcc-4.8.1/configure \
  --prefix=/opt/gcc-4.8.1 \
  --disable-multilib \
  --enable-languages=
        c,c++,ada,fortran,objc,obj-c++ \
  --target=x86_64-apple-darwin12 \
  --build=x86_64-apple-darwin12
```

Thread model: posix

gcc version 4.8.1 (GCC)

MD5 (gcc-4.8.1-x86_64-apple-darwin12.tar.bz2) = 549d32da94a7af15e99bb98a7d288be9

Install by
==========

```
$ cd /
$ sudo tar jxvf ~/Downloads/gcc-4.8.1-x86_64-apple-darwin12.tar.bz2
```

and put /opt/gcc-4.8.1/bin first on your PATH.

Notes
=====

The compiler is GPL version 3 with the Runtime Exception, so executables built with it can be released on proprietary terms PROVIDED THAT they make no use of the packages from GNAT GPL 2013, which are full GPL.

Changes made to GPRbuild GPL 2013 are in gprbuild-2013-src.diff. They:

- remove the '-c' flag that is wrongly passed to ranlib (and isn't by gnatmake).
- correct a problem when building static stand-alone libraries.

Changes made to GNATColl GPL 2013 are in gnatcoll-gpl-2013-src.diff. Only changes necessary for the build are included.

Changes to ASIS GPL 2013 are in asis-gpl-2013-src.diff. Only changes necessary for the build are included.

In addition to the above, a new library gnat_util is required by GNATColl. A Sourceforge project to provide this has been set up at https://sourceforge.net/projects/gnatutil/; release 4.8.1 is included here. This is the equivalent of the Debian libgnatvsn.

The GNATColl build was configured as below, which is minimal apart from GNU Readline being enabled. Users may wish to reconfigure for their own requirements.

```
Shared libraries:     yes (default: static)
Gtk+:   no (requires pkg-config and
        gtkada.gpr)
Python: yes /System/Library/
        Frameworks/Python.framework/
        Versions/2.7 (see --with-python)
PyGtk:  no (see --enable-pygtk)
PyGObject: no (see --enable-pygobject)
Syslog: yes (see --enable-syslog)
Readline (GPL license): yes (see --with-
        readline --enable-gpl)
gmp: no (see --with-gmp)
PostgreSQL: no -L/usr/lib (see --with-
        postgresql) Sqlite: embedded
        (see --with-sqlite)
Iconv:  yes (see --with-iconv)
Projects: yes
```

References to Publications

Storing Large Volumes of Data With AVR-Ada

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue 28 May 2013
Subject: Storing large data amount to flash memory
URL: http://arduino.ada-language.com/storing-large-data-amount-to-flash-memory.html

If you paid attention in my Olimex MOD-LCD3310 article [1], you noticed that I stored a large array for font/characters to RAM memory. This is highly inefficient since the array takes space both on the flash and on the memory at the same time.

To make the space usage more efficient, you can specify the array to be only on the flash. This way, you can use the precious RAM to other things. The downside is that you cannot access the array on the flash as easily as from RAM.

However, AVR-Ada provides you relatively simple means to place data on flash and to retrieve it from there. You need to do only two things, add a linker pragma like:

```
pragma Linker_Section (
            Fonts, ".progmem");
```

and use AVR.Programspace package to fetch the data:

```
Place := Fonts (0, I)'Address;
Offset := AVR.Programspace.
                    Get_Byte (Place);
```

Notice how the (flash) address of Fonts array is specified by Address attribute. You don't need to know any details how the address is calculated, the compiler handles everything for you.

The updated code is available at my arduino-mod-lcd3310 repository [2] as usual. The revision number for this change is ab6d9f7edc6f.

And here the .bss usage before and after Linker_Section pragma:

```
 $ avr-size main-ram.elf
text  data   bss   dec   hex filename
2570  582    512   3664  e50 main-
                              ram.elf
 $ avr-size main-progmem.elf
text  data   bss   dec   hex filename
3038  112    511   3661  e4d main-
                              progmem.elf
```

The data (RAM) usage is higher without .progmem Linker_Section pragma, and with the pragma the text (flash section) is higher.

[1] http://arduino.ada-language.com/displaying-characters-on-mod-lcd3310-by-using-olimexino-328-with-ada.html

[2] https://bitbucket.org/tkoskine/arduino-mod-lcd3310

Developing Secure Code Using SPARK

From: Benjamin M. Brosgol, AdaCore
Date: Sat Jun 1 2013
Subject: Developing secure code using SPARK – Part 1
URL: http://www.embedded.com/design/safety-and-security/4419246/Developing-secure-code-using-SPARK---Part-1-

The modern cyberworld is a dangerous place. Software must not only be reliable—i.e., perform its intended functions—but also robust: It needs to withstand attempts at subversion from the array of threats posed by malevolent sources. Implementing security as an add-on isn't effective. Performing static analysis retrospectively on the source code of an existing system may uncover bugs and vulnerabilities but can never demonstrate their absence. Instead, developers have to consider preventive measures from the start, and for the most critical kinds of systems, implement an approach backed by mathematical rigor.

One technique is to use an appropriate programming language that permits specifying relevant security-oriented properties ("contracts"), and then demonstrate statically that these properties are satisfied by applying automated tools as the system is being designed and implemented.

[…]

Managing Assertion Execution

From: Yannick Moy
Date: Jun 2013
Subject: Gem #149 : Asserting the truth, but (possibly) not the whole truth
URL: http://www.adacore.com/adaanswers/gems/gem-149-asserting-the-truth-but-possibly-not-the-whole-truth/

[…]

So now the Ada programmer has a rich set of assertions to state control-relevant properties (Assert, Pre, Post, Loop_Invariant, Assume, Assert_And_Cut) and data-relevant properties (Static_Predicate, Dynamic_Predicate, Type_Invariant).

How does one state which assertions get executed? And how does one differentiate between different executables, say, between one created for debugging/testing, and one created for production?

[…]

[Yannick Moy explains how you select which assertions get executed, both with "pragma Assertion_Policy" and with GNAT command line arguments. —sparre]

Ada-Python Demonstration

From: Maciej Sobczak <maciej@msobczak.com>
Date: Thu, 11 Jul 2013 14:12:24 -0700
Subject: Ada-Python demo
Newsgroups: comp.lang.ada

Some time ago I have posted an article presenting the principles of writing Ada loadable modules that can be used as extensions for Python scripts.

This time I have gathered some basics for the opposite case, which is writing the Ada program that embeds the Python interpreter and loads external Python scripts:

http://www.inspirel.com/articles/Ada_Python_Demo.html

Having covered both directions of the inter-language integration, the article also contains the complete two-part working demo that shows all this magic working.

Attiny4313 and I2C Master Using USI

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: July 13 2013
Subject: Attiny4313 and I2C master using USI
URL: http://arduino.ada-language.com/attiny4313-and-i2c-master-using-usi.html

Getting I2C working with attiny processors has been yet another long-time project of me. It has been stuck mostly because I haven't had time to create attiny4313 board with I2C chip on it.

But finally, I managed to create one:

[…]

The Attiny_TWI.Master package still needs some finishing touches (like fixing some delays), but once I am happy with it, I will commit it into AVR-Ada repository.

Ada for the C++ or Java Developer

From: Quentin Ochem, AdaCore
Date: July 23 2013
Subject: Ada for the C++ or Java Developer
URL: http://www.adacore.com/knowledge/technical-papers/ada-for-the-c-or-java-developer/

This document will present the Ada language using terminology and examples that are familiar to developers that understand the C++ or Java languages.

[67 pages PDF document presenting Ada. —sparre]

SPARK 2014: Why I am Backing a Predictable Winner

From: Stuart Matthews, Altran
Date: Sun Aug 4 2013
Subject: SPARK 2014: Why I am backing a predictable winner
URL: http://www.embedded.com/electronics-blogs/other/4419245/SPARK--Why-I-am-backing-a-predictable-winner

In a recent posting, Embedded.com technical editor Bernard Cole compared the C programming language to a prize-winning race horse: temperamental, stubborn, and unpredictable

These are not the qualities that you would choose for a programming language you intend to deploy for a high-integrity embedded system, where safety or security are key requirements. In this context, we want languages that offer predictability and an efficient and effective means of automated verification.

[…]

Embedded.com

From: KK6GM
<mjsilva@scriptoriumdesigns.com>
Date: Mon, 5 Aug 2013 19:32:56 -0700
Subject: Embedded.com goes gaga over Ada
Newsgroups: comp.lang.ada

Almost a dozen articles on Ada and/or SPARK. Must be a record!

http://e.ubmelectronics.com/audience/UBMTechNewsletters/08-05-13-EMB-Tech-Focus.html

Now, where is that ARM Cortex GNAT release…release…release… (yes, that's the sound of a broken record).

Updated Ada 2012 Rationale Available

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri Aug 16 2013
Subject: Updated Ada 2012 Rationale available
URL: http://www.adaic.org/2013/08/updated-ada-2012-rationale-available/

An updated edition of the Ada 2012 Rationale is available at:

http://www.ada-auth.org/standards/rationale12.html

This edition of the Rationale combines the first eight chapters of the Rationale into a single document, fixes a number of errors, adds an index, and adds discussion of various details of Ada 2012 that were changed since the original publication of these chapters in the Ada User Journal. We expect that additional chapters will be added to this edition roughly every three months.

The Rationale for Ada 2012 provides an overview of new Ada 2012 features, examples of their use, compatibility with Ada 95 and 2005, and more. It was written by John Barnes, and was sponsored in part by the Ada Resource Association. This is an unofficial description of the language; refer to the Ada 2012 standard for detailed language rules.

Randy Brukardt, ARG Editor

Ada Inside

Scala Musical Software

From: Manuel Op de Coul
<Manuel.op.de.Coul@eon.com>
Date: Wed Dec 12 2012
Subject: Scala Home Page
URL: http://www.huygens-fokker.org/scala/

Scala is a powerful software tool for experimentation with musical tunings, such as just intonation scales, equal and historical temperaments, microtonal and macrotonal scales, and non-Western scales. It supports scale creation, editing, comparison, analysis, storage, tuning of electronic instruments, and MIDI file generation and tuning conversion. All this is integrated into a single application with a wide variety of mathematical routines and scale creation methods. Scala is ideal for the exploration of tunings and becoming familiar with the concepts involved. In addition, a very large library of scales is freely available for Scala and can be used for analysis or music creation.

[…]

Features

- Reliable. Scala is written in the programming language Ada.
- Available on multiple platforms: Windows, GNU Linux, MacOS X (10.4 or higher) and Unix, see the Download page.
- Free. Please read the distribution section below.

[…]

Development software

Scala was developed in Ada with the following excellent free tools:

- Excel Writer
- GNAT: Gnu Ada Translator
- Glade
- GtkAda
- Gtk+
- Zip-Ada

NetWeather

From: Peter C. Chapin
<PChapin@vtc.vsc.edu>
Date: Sun Apr 21 2013
Subject: NetWeather
URL: https://github.com/pchapin/netweather

NetWeather is a network accessible weather station. It was used as a class project during the Spring 2008 semester of ELT-2720 and CIS-2720. A team of students in Williston (under the direction of Matt Gallagher) developed the weather station hardware and electronics. A team of students in Randolph (under my direction) developed the server software. This repository contains the result of those efforts.

It is my hope that NetWeather will continue to grow and develop. Many useful and interesting extensions can be imagined. The project is, in general, a good example of the nature of electrical and computer engineering technology as taught at Vermont Technical College. The existing system could serve as a good starting point for future projects or even have a role as a marketing and recruitment tool.

GNAT Pro and PolyORB to be used for the ISS Core Ground System

From: AdaCore Press Center
Date: Tue Jun 11 2013
Subject: Astrium Selects AdaCore's GNAT Pro and PolyORB for International Space Station
URL: http://www.adacore.com/press/astrium-polyorb/

Ada-based software development environment and middleware ensure high reliability of essential communications technologies for ISS' Core Ground System.

BERLIN, PARIS, and NEW YORK, June 11, 2013 - Ada-Europe 2013 Conference - AdaCore announced today that Astrium, a wholly owned subsidiary of EADS, has selected AdaCore's GNAT Pro development environment and PolyORB middleware toolset for use in the Core Ground System (CGS) - CGS forms the basis to operate the Columbus laboratory, the European contribution to the International Space Station (ISS). The CGS insures efficient communication across a network of User Support and Operation Centres distributed throughout Europe. GNAT Pro and PolyORB are being used to facilitate efficient, reliable communication across the Ada applications that Astrium developed for Columbus ground and onboard applications. PolyORB also supplies Astrium's developers with the interoperability necessary for CGS, allowing for equally seamless integration of software systems written in Java or C++.

The CGS was first built as a test and checkout system, in Ada, to test and qualify the Columbus laboratory on the ground, and it continues to be used as a monitoring and control system for the Columbus laboratory and the European payloads integrated in Columbus at the Columbus Control Centre in Oberpfaffenhofen and various payload operation centers spread across Europe. More specifically, the CGS is an integrated software toolset that is used as the common software in all operational European ground facilities. It links together the processes and phases of development, integration, test and operations, eliminating incompatibilities in the work flow.

To carry out the Ada development of CGS, Astrium selected the GNAT Pro development environment. This product includes tools that take advantage of Ada's properties to perform additional static and dynamic analysis, reaching even higher levels of reliability. For the middleware implementation of the CGS, Astrium selected the PolyORB toolset. PolyORB provides distribution services through standard programming interfaces and communication protocols. It addresses distribution model interoperability issues by allowing a single middleware instance to efficiently support multiple personalities executing simultaneously. Its modular architecture, emphasizing code reuse, allows the definition and deployment of middleware configurations that are specially adapted for real-time, high integrity applications.

"Heterogeneity in modern systems is an increasing reality, driving strong requirements in terms of interoperability," explains Quentin Ochem, Technical Account Manager at AdaCore. "PolyORB provides an integrated solution, allowing the interconnection of components developed using the Ada programming language with software developed by other providers on different technologies, guaranteeing the longevity of the development investments."

"The usage of CORBA allows us to use our well-proven Ada applications in a complex operation scenario in connection with other ground products. With PolyORB, we now get all major Ada development tools and components from one source. The competent support of the AdaCore team and the high quality of the AdaCore products helped us in selecting the PolyORB middleware," states Stephan Marz, software engineer at Astrium.

By selecting AdaCore's products, Astrium has found a valuable `one-shop' solution for both the Ada development and the middleware implementation of the CGS. The expert advice and unmatched product support that AdaCore offers can take into account the needs of Astrium's Ada developers, and also answers the challenges Astrium faces creating robust software that functions efficiently across the many different systems created by the international community at work on the ISS.

Authoritative DNS Server

From: Barry Fagin and Martin Carlisle
Date: Wed Jun 12 10:45:00 2013
Subject: Provably Secure DNS: A Case Study in Reliable Software
URL: http://ironsides.martincarlisle.com/

IRONSIDES is an authoritative DNS server that is provably invulnerable to many of the problems that plague other servers. It achieves this property through the use of formal methods in its design, in particular the language Ada and the SPARK formal methods tool set. Code validated in this way is provably exception-free, contains no data flow errors, and terminates only in the ways that its programmers explicitly say that it can. These are very desirable properties from a computer security perspective.

IRONSIDES is not a complete implementation of DNS. In particular, it does not support zone transfers or recursive queries. It does, however, support a sufficient number of DNS records to be useful as an authoritative DNS server for an enterprise.

[...]

Wasabee

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sun Aug 11 2013
Subject: Gautier's blog: Wasabee's first steps…
URL: http://gautiersblog.blogspot.dk/2013/08/wasabees-first-steps.html

Wasabee's first steps…

http://sf.net/projects/wasabee

[Wasabee is intended to become a Web browser with a focus on user safety. —sparre]

Elliot 900 Emulator

From: Erik Baigar <erik@baigar.de>
Date: Wed, 14 Aug 2013 19:55:15 +0200
Subject: Re: Low-level programming in Ada?
Newsgroups: comp.lang.ada

> […]

Hey very nice that someone mentions the Elliott machines here. The awareness of this architecture is near to zero.

2003 I got hands on a small mil spec computer which I reverse engineered and reanimated. Only years later I learned, that it is an embedded 102 (essentially 12 bit variant of the 900 series machines which have been the successor of the 803).

I also ported an emulator for the 920 which was written by two good old hands in Ada many years ago to more modern platforms - if interested you may have a look to my page regarding the project:

http://www.programmer-electronic-control.de

Ada in Context

Dummy "out" Parameters

From: Adam Beneschan <adam@irvine.com>
Date: Wed, 29 May 2013 19:01:50 -0700
Subject: Re: GNAT 2013 is out!
Newsgroups: comp.lang.ada

[…]

I've always wanted some kind of feature in Ada that would allow a caller to provide a "dummy" for OUT parameters, without having to declare a new variable. The compiler would allocate a temporary object (and a separate one for each use of a "dummy") and then discard it after the call. It wouldn't work well when parameter types are unconstrained array or discriminant records, though.

[…]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 30 May 2013 14:53:43 -0500
Subject: Re: GNAT 2013 is out!
Newsgroups: comp.lang.ada

> […]

Hmm, that seems like a good idea to me. But what would the syntax be? <> maybe?

```
My_Proc (Obj1, Obj2, Result => <>);
```

Someone should seriously propose something on this line on Ada-Comment. I can see objections about making it too easy to ignore errors -- but errors shouldn't be returned in parameters in the first place, so I don't find that terribly compelling.

Anyway, this problem was a significant annoyance in the design of Claw. We have routines that return rarely used values that would normally just be discarded. That's especially an issue for call-back routines, where we have to provide all of the parameters that you could possibly use, even if you have no need for half of them. (And the typical solutions using overloading and/or default parameters is impractical.)

I tried to work out a solution based on default parameters for all modes (which would provide your dummy result along with other uses), but it didn't work out very well. The main problem was that the default objects usually had to be globals, and that could cause an unsafe use of shared variables in a tasking environment.

The <> solution doesn't suffer from this, and it also would make the dummy--ness of the parameter visible to the reader, which would reduce two of the major objections. (The latter means that style checkers could prevent the use of such dummies if it was considered a problem -- and it would make the fact that they're dummies much more visible than a regular declaration does.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 31 May 2013 17:07:37 -0500
Subject: Re: GNAT 2013 is out!
Newsgroups: comp.lang.ada

> […] won't work if the type of Result is unconstrained, […]

That's not an issue, that's a feature. I would not expect this to work, because the "dummy" needs bounds/discriminants. <> means "default initialized", and if that is illegal, it should be illegal here, too.

[…]

Proper Flags to 'gnatmake'

*From: Peter Brooks
<peter.h.m.brooks@gmail.com>
Date: Mon, 17 Jun 2013 03:57:26 -0700
Subject: Range check for type 'Integer'
Newsgroups: comp.lang.ada*

[…]

```
procedure Compute_Loop is

  procedure Double(Item : in out Integer) is
  begin
   Item := Item * 2;
  end Double;

  X : Integer := 1;
begin
  loop
    Put ("This is ");
    Put (X);
    New_Line;
    Double (X);
  end loop;
end Compute_Loop;
```

Output:

[…]

```
This is   268435456
This is   536870912
This is  1073741824
This is -2147483648
This is           0
```

… [forever]

So the 'Integer' has rolled over to negative and then rolled back to 0 - but with no run-time error.

Why is there no range check error on type Integer?

*From: Simon Wright
<simon@pushface.org>
Date: Mon, 17 Jun 2013 12:54:02 +0100
Subject: Re: Range check for type 'Integer'
Newsgroups: comp.lang.ada*

[…], compile with -gnato to enable integer overflow detection.

*From: Jacob Sparre Andersen
<jacob@jacob-sparre.dk>
Date: Mon, 17 Jun 2013 15:23:11 +0200
Subject: Bug in 'gnatmake' (Was: Range check for type 'Integer')
Newsgroups: comp.lang.ada*

> […]

Because there is a major error in the design of GNAT. You have to give 'gnatmake' a very specific set of command line flags to turn it into an Ada compiler:

```
-fstack-check -- Generate stack checking
                 code
-gnata        -- Enable assertions
-gnatE        -- Dynamic elaboration
                 checking
-gnato        -- Overflow checking
```

If you use the GNAT Project Manager (and '*.gpr' files), I suggest that you copy this file project file and use it in all your GNAT based Ada projects:

```
abstract project Ada_2012 is
  for Source_Dirs use ();
  package Compiler is
    for Default_Switches ("Ada")
      use ("-fstack-check", -- Generate stack
                  -- checking code (part of Ada)
           "-gnata", -- Enable assertions
                     -- (part of Ada)
           "-gnatE", -- Dynamic elaboration
                     -- checking (part of Ada)
           "-gnato", -- Overflow checking
                     -- (part of Ada)
           -- Project preferences below:
           "-gnatf", -- Full, verbose error
                    --messages
           "-gnatwa", -- All optional warnings
           "-gnatVa", -- All validity checks
           "-gnaty3abcdefhiklmnoOprstux",
                     -- Style checks
           "-gnatwe", -- Treat warnings as
                     -- errors
           "-gnat2012", -- Use Ada 2012
           "-Wall", -- Enable all GCC warnings
           "-O2"); -- Optimise (level 2/3)
  end Compiler;
end Ada_2012;
```

Here is a simple example of how to use the file:

```
with "ada_2012";
project AUJ_Tools is
  for Main use ([…]);
  package Compiler renames
        Ada_2012.Compiler;
end AUJ_Tools;
```

The more complicated version reads something like this:

```
project AUJ_Tools is
  for Main use ([…]);
  package Compiler is
    for Default_Switches ("Ada")
      use Ada_2012.Compiler'
          Default_Switches ("Ada") &
              ("-some-extra-switches");
  end Compiler;
end AUJ_Tools;
```

*From: Robert A Duff
<bobduff@shell01.TheWorld.com>
Date: Mon, 17 Jun 2013 12:50:06 -0400
Subject: Re: Bug in 'gnatmake' (Was: Range check for type 'Integer')
Newsgroups: comp.lang.ada*

> -fstack-check
>   -- *Generate stack checking code*
> -gnata   -- *Enable assertions*
> -gnatE
>   -- *Dynamic elaboration checking*
> -gnato   -- *Overflow checking*

You should never use -gnatE, unless you are dealing with legacy code that that won't work without it, and you can't afford to fix the code.

-fstack-check is the default, at least on the most popular targets.

You usually want -g (debugging info). And the debugger works better if you turn off optimizations (-O0).

Elaboration Order Handling

*From: Jeffrey R. Carter
<jrcarter@acm.org>
Date: Tue, 18 Jun 2013 18:21:02 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada*

> […]

Elaboration order isn't entirely implementation defined, is it? There's a partial order defined by calls made during elaboration from one package to another. Within that ordering there may be groups of packages that may be elaborated in any order after the packages that must be elaborated before them and before the packages they must be elaborated before. Within those groups, why not use lexical order of package names?

*From: Robert A Duff
<bobduff@shell01.TheWorld.com>
Date: Wed, 19 Jun 2013 08:38:21 -0400
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada*

> […]

In standard Ada, there's a partial order defined by 'with' clauses, parent/child relationships, and various pragmas.

GNAT takes into account calls made during elaboration, but standard Ada does not. And GNAT's rules are necessarily conservative (see Halting Problem).

> […] Within those > groups, why not use lexical order of package names?

Good idea. ;-)

It doesn't solve all the problems with Ada's elaboration model, but it solves the most expensive one (portability). We'd still have the problem that the chosen order can be wrong. And the fact that programmers have to deal with a bunch of kludgy pragmas. And the fact that what should be a compile-time error is a run-time exception. And the fact that the order is global, rather than localized to the children of a single package.

Oh, and the fact that the whole model is overly restrictive.

For example, it makes perfect sense to say:

```
package Symbols is
  type Symbol is private;
  function Intern(S: String) return
       Symbol;
  Empty_Symbol: constant Symbol :=
       Intern(""); -- Wrong!
```

But that doesn't work in Ada. […]

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Wed, 19 Jun 2013 08:22:30 -0400
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

[…]

Or you could base it on the order in which 'with' clauses happen to appear.

Programmers shouldn't be depending on lexicographic (or whatever) order, but if they do so by accident, it's best that their program still works 10 years later when it is ported (probably by a different set of programmers).

> […]

There's no reason the order has to be “natural”. I'm just saying it should be the same on all implementations.

[…] with elab order, there is exactly ZERO benefit to allowing arbitrary orders. And the disadvantage is huge: I've seen people wasting boatloads of money on this!

[…]

From: Adam Beneschan <adam@irvine.com>
Date: Wed, 19 Jun 2013 08:46:10 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

[…]

> Or you could base it on the order in which 'with' clauses happen to appear.

No, I don't think that would work. Say there are two packages, Pack1 and Pack2, that get included in the program, and there is no rule (in the current language) that specifies which one gets elaborated first. Adding a rule based on the “with” clause order would work if Pack1 and Pack2 are both with'ed by the same package, Pack3, and are not with'ed anywhere else. But other than that one narrow case, I don't see how you could write a rule to base elaboration order on “with” clause order.

From: Georg Bauhaus <bauhaus@maps.arcor.de>
Date: Wed, 19 Jun 2013 22:47:29 +0200
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

[…]

Having had to live with products of programmers favoring symbolic cleverness, I naturally think of what happens when some project depend on lexicographical order and then someone wishes to give packages different names. ARGH!

From: Adam Beneschan <adam@irvine.com>
Date: Wed, 19 Jun 2013 14:36:42 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

Yeah, I thought about that. It would seem strange to have a program that had been tested suddenly quit working when a programmer decides to change the name of a package, and makes no other change. That would certainly be a frustrating occurrence.

On the other hand, the current situation isn't any better. If you have two packages whose elaboration order isn't defined by the language, the compiler could elaborate them in one order, and then in a future build, could choose the reverse order for whatever reason it chooses. Depending on the compiler, changing the name of a package could cause that to happen, if (say) that causes an old package name to be removed from the middle of some internal list and then inserted at the end of the list with the new name, which could cause a difference in how the compiler determines the elaboration order.

The more I think about it, the more I think the answer is that the elaboration of library package P should just be prohibited from calling subprograms in another package Q, or accessing variables declared in Q's specification, unless there is an Elaborate(_All) pragma, or unless there's some other reason Q must be elaborated before P (e.g. P's *specification* says “with Q”), or Q is Pure, or perhaps some other things. Writing the rules to make sure this happens wouldn't be easy. It probably means that P's elaboration code also can't call a subprogram in P unless that subprogram is also declared as “promising not to call anything outside this package”. There would have to be restrictions on dispatching calls and calls through access-to-subprograms. I'm not sure this would be a feasible solution. But to me, having the language defined so that “if the elaboration order is undefined, we'll put restrictions on things so that the order can't possibly matter” seems better, theoretically, than coming up with some unnatural order just so that we can say “something is defined”. That's just my gut feeling. Since I doubt anyone is really going to think about adding this to Ada, all this is hypothetical until Bob decides to finally define and implement his hobby language. :-)

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Wed, 19 Jun 2013 20:57:35 -0400
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

Of course programmers shouldn't do that (depend on lexicographical order). But as I said before, that's already possible with GNAT -- it uses lexicographical order. I doubt if anyone depends on that on purpose, but it's easy to do so by accident.

And with standard Ada, it's even worse: the order can change for any reason, or for no reason. The implementation is allowed to roll dice to determine the order! Even running the exact same program without recompiling could change order!

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 19 Jun 2013 18:09:24 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

Since we're discussing elaboration order, there's a case that I wonder about:

```
with B;
package A is

 function F (I : Integer) return B.Thing;
 function R return Integer;

end A;

package body A is

  function F (I : Integer) return B.Thing is
  begin
    return B.To_Thing (I);
  end F;

  function R return Integer is
  begin
    return 7;
  end R;

end A;

package B is
  type Thing is private;

  function To_Thing (I : Integer) return
            Thing;
private
  type Thing is new Integer;

end B;

with A;
package body B is
  function To_Thing (I : Integer) return
            Thing is
  begin
    return Thing (I);
  end To_Thing;

  C : constant Integer := A.R;
end B;
```

It seems to me there's a valid elaboration order for these:

* spec of B

* spec of A

* body of A

* body of B

I've never been able to get such code to bind, though. Is there some way to get this accepted, or is it illegal Ada?

From: Adam Beneschan <adam@irvine.com>
Date: Wed, 19 Jun 2013 19:29:48 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

It's legal Ada (and should run without raising Program_Error). It looks like http://docs.adacore.com/gnat-unw-docs/html/gnat_ugn_36.html, especially C.10, discusses this sort of situation, but I don't know if you looked there already.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 19 Jun 2013 23:08:46 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

No, I hadn't. Thanks for pointing it out. I'd always tried to fix this using "pragma Elaborate[_All]" in the context clauses, which always failed. I hadn't considered the effect of putting "pragma Elaborate_Body" in the spec of A. With that, I can get it to build.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Thu, 20 Jun 2013 11:11:11 -0400
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

A case of mutually-recursive packages. That's unusual, because it's usually better to design software in layers, where higher layers use the lower layers, but not vice-versa.

But sometimes such mutually-dependent packages are exactly the right thing.

> […]

In standard Ada, the above is perfectly legal, and does not raise Program_Error.

> It seems to me there's a valid elaboration order for these:
>
> * spec of B
>
> * spec of A
>
> * body of A
>
> * body of B

Yes, and that is the only order allowed by the RM.

> […]

Without -gnatE, GNAT will complain about an elaboration cycle, because it is inserting an implicit "pragma Elaborate_All(A);" on the body of B. It does that because it's trying to preserve abstraction -- it wants the code to work no matter what A.R does (it's not looking at the body of A.R while compiling body of B). In fact, A.R could have called A.F, in which case the program is broken -- the only allowed order causes Program_Error in standard Ada, and the whole point of the not-gnatE mode is to do all such run-time checks statically. Imagine adding that call to A.F during maintenance.

And the reason the implicit pragma causes a cycle is that Elaborate_All is transitive -- it requires the body of B to be elaborated before the body of B, which is impossible.

But you've got mutual recursion here, so A and B are necessarily tightly coupled, and you, the programmer, can know that A.R does NOT call anything in B. In that case, you can solve the problem by putting "pragma Elaborate(A);" on the body of B. GNAT uses that Elaborate *instead* of the implicit Elaborate_All, and Elaborate is nontransitive, so it all works, using the order you showed above.

But be careful: If you later change A.R to call A.F, you'll be in trouble.

The other way to get this example to work is to use -gnatE. I don't recommend that.

From: Adam Beneschan <adam@irvine.com>
Date: Wed, 19 Jun 2013 19:11:14 -0700
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […] the order can change for any reason, or for no reason. The implementation is allowed to roll dice to determine the order! Even running the exact same program without recompiling could change order!

You say that like it's a bad thing. Actually, I can see a use for a compiler option to use a random number generator to determine the elaboration order (or other things that are implementation-dependent), because if the program relies on the order when it shouldn't, a random order will increase the chance that repeated testing will expose the problem.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Thu, 20 Jun 2013 10:44:53 -0400
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

Yes, I agree that would be useful. But only as an option, and only for testing.

It would be a useful option even if Ada were designed as I've been saying it should (such that elaboration order is deterministic/portable). If the rule was "lexicographic order", as I have suggested, I still don't think programmers should depend on lexicographic order, and this option could help prevent that.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Wed, 19 Jun 2013 12:41:46 -0400
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […] semantics could depend on the lexicographical order of the identifiers.

Well, the GNAT dialect of Ada does just that! Is it weird to have it in the Ada RM, but not so weird to have it in the GNAT RM?

Yes, it's kind of weird to use that order, but there's no other order that's any less weird. What we want is an arbitrary order that is the same for all compilers. "Arbitrary" in the sense that we don't care what the order is, and we don't deliberately write code that depends on it.

> […]

The with clauses (and parent/child relationships and so forth) form a directed graph. You can define an order in which to walk it however you like, so long as it's deterministic. You could say, start at the main procedure spec. Walk all of the with'ed specs, in the order mentioned. Walk the corresponding body. ("Walk" is recursive, of course. And as usual for graph walks, you keep track of which nodes have been visited, and if you run across an already-visited node, do nothing.)

This visits all the library items in a well-defined order, and if all compilers were required to use that algorithm, it would be a portable order. Then you say that whenever the current Ada rules allow multiple orders, you must choose the order from the above graph-walking algorithm.

> […]

If Pack1 and Pack2 are with-ed elsewhere (than Pack3), then either that "elsewhere" comes before or after Pack3 in the walk (or some of each). Whichever 'with Pack1' you run across first in the walk is the one that determines where Pack1 occurs in the order.

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Wed, 19 Jun 2013 14:07:59 +0100
Subject: Re: Elaboration order handling (Was: Bug in 'gnatmake')
Newsgroups: comp.lang.ada

> […]

I did not know about -gnatwl, and had dozens of Elaborate_All pragmas, which I now know to have been mostly unnecessary.

C Bindings and Memory Management

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 1 Jul 2013 14:32:27 +0200
Subject: Re: Thick bindings to a C library and gnattest: suggestions?
Newsgroups: comp.lang.ada

> [passing huge arrays to and from C functions]

The first point is that it is not the objective of bindings to manage memory. Of course, there could be bindings which do that, in which case you would allocate objects transparently to the caller and have some garbage collection schema behind opaque handles to the objects. This is a possible design but it is not what you probably wanted. So let us take for granted that it is the client's responsibility to allocate objects. In this case the bindings shall work for any kind of objects allocated in any possible memory pool, stack included.

Now, how would you do that? There are many ways.

1. Prior to Ada 2005, the usual method was one you find in Ada.Text_IO.Get_Line. You use a procedure and a parameter telling how much elements were written:

```
procedure
      Read_Double_Array_From_FITS
         (A : in out Double_Array;
          Last : out Positive);
```

   Here Last indicates the last element of A containing data. The implementation would raise End_Error when there are more than A'Length elements in A. Get_Line, for example, returns Last = A'Last meaning that there is more to read. The caller uses A (A'First..Last) in further calls.

   In my libraries I am using a slightly more universal approach:

```
procedure
      Read_Double_Array_From_FITS
         (A : in out Double_Array;
          Pointer : in out Positive);
```

   Here A (Pointer..A'Last) is where the result is stored and then Pointer is advanced to the first element following the input. So the result is between old pointer and new pointer - 1.

2. With Ada 2005 you can use return statement

```
procedure
         Read_Double_Array_From_FITS
         return Double_Array is
begin
   return Result : Double_Array
         (1..Get_Number_Of_Elements) do
      -- Fill Result here
   end return;
end Read_Double_Array_From_FITS;
```

   The caller is free to use this function with the allocator new:

```
A : access Double_Array :=
         new Double_Array'
         (Read_Double_Array_From_FITS);
```

   Theoretically the compiler could optimize temp object away. (You should check if GNAT really does this)

Dealing with huge arrays I would prefer the approach #1.

I would probably allocate some scratch buffer and reuse it all over again.

Another approach is using #1 or #2 with some custom storage pool organized as a stack or arena in order to minimize memory management overhead.

In any case, it is not the bindings' business.

> The fact that Ada arrays can have arbitrary bounds whom they carry is one of the things that made me interested towards Ada at the beginning. Why did you say this might be “troublesome”?

Because C arrays have none. When you want to pass an Ada array to C you must flatten it. One way is to declare a subtype:

```
procedure Bar (A : Double_Array) is
   subtype Flat is A (A'Range);
   B : Flat := ...;
begin
   -- B does not have bounds and can be
   -- passed around as-is
```

Some pass pointer to the first element. After all, C's arrays are a fiction.

Some use addresses. E.g. GtkAda bindings pass System.Address for any C objects sparing headache of proper types. Purists would consider this approach rather being sloppy.

[…]

From: Georg Bauhaus <bauhaus@maps.arcor.de>
Date: Tue, 02 Jul 2013 10:55:39 +0200
Subject: Re: Thick bindings to a C library and gnattest: suggestions?
Newsgroups: comp.lang.ada

> […]

Avoid pointers on the Ada side and you need not worry about the stack. This is true insofar as Ada language definition says that Ada arrays will be passed as pointers to the C world, automatically. (See Interfacing to C)

Indeed, just do what is natural on both sides:

You can pass C-array variables (pointers) on the C side and expect Ada to handle plain Ada-array variables. No pointers needed. In fact, they complicate things due to doubled indirections.

The following seems to work on my system.

The Ada side “imports” data and exports its subprograms.

```
#include <stdlib.h>
#include <stdio.h>
double call_ada (size_t n)
{
  extern void ada_side_takes_vector
             (double*, size_t);

  double *thing = malloc(n * sizeof(double));
  for (int k = 0; k < (int)n; ++k) {
    thing[k] = k;
  }
  ada_side_takes_vector(thing, n);
  return thing[n/2];
}

int main()
{
  extern void adainit(void);
  extern void adafinal(void);
  double result;

#define M ((2<<20)/sizeof(double))

  adainit();
  result = call_ada(500 * M);
  adafinal();
  printf("result is %f\n", result);
  return 0;
}

with Interfaces.C; use Interfaces;

package Bigimport is

   pragma Pure (Bigimport);
   subtype Dbl is C.Double;
   subtype Zint is C.ptrdiff_t range
            0 .. C.ptrdiff_t'Last;
   type Lots_Of_Numbers is array (Zint) of
            Dbl;
   pragma Convention (C,
            Lots_Of_Numbers);
   procedure Takes_Vector (V : in out
            Lots_Of_Numbers;
             N : in C.size_t);
   pragma Export (C, Takes_Vector,
            "ada_side_takes_vector");

end Bigimport;

package body Bigimport is

   procedure Takes_Vector (V : in out
            Lots_Of_Numbers;
            N : in C.size_t) is
     use type Dbl, C.size_t;

   begin
     for K in Zint range 0 .. Zint (N - 1) loop
       V (K) := V (K) / 2.0;
     end loop;
   end Takes_Vector;

end Bigimport;
```

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 02 Jul 2013 10:00:29 -0700
Subject: Re: Thick bindings to a C library and gnattest: suggestions?
Newsgroups: comp.lang.ada

> […]

There is no need for access types in Ada to do this. See ARM B.3 (70):

“An Ada parameter of an access-to-subprogram type is passed as a pointer to a C function whose prototype corresponds to the designated subprogram's specification.”

In other words, you can write

```
type Vector is array (Positive range <>) of
        Natural;

procedure Read_Vector_From_File
        (A : out Vector) is
   procedure Internal (A : out Vector;
          N : in Interfaces.C.Unsigned);
   pragma Import (C, Internal,
        "read_vector");
begin
   Internal (A => A,
        N => A'Length);
end Read_Vector_From_File;
```

and the compiler takes care of passing a C pointer to the 1st component of A. No need for the components of Vector to be aliased, and no problems with accessibility checks.

Switching from GtkAda 2 to GtkAda 3

From: Chris Sparks <mr_ada@cox.net>
Date: Mon, 01 Jul 2013 06:10:58 -0700
Subject: Surprised by the changes to GTK 3.0
Newsgroups: gmane.comp.gnome.gtk+.ada

I have been using GTK 2.x for a while and going to the 3.0 version really messed up years of personal software development. I was wondering if anyone has a reference to how one converts from the 2.0 series to the 3.0 series?

From: Nicolas Setton <setton@adacore.com>
Date: Mon, 1 Jul 2013 15:21:21 +0200
Subject: Re: Surprised by the changes to GTK 3.0
Newsgroups: gmane.comp.gnome.gtk+.ada

Yes, the GtkAda manual has a chapter about this, "Transitioning from GtkAda 2 to GtkAda 3". This contains an overview, and a package-by-package transition guide.

You can also refer to the Gtk+ documentation at:

<https://developer.gnome.org/gtk3/3.3/gtk-migrating-2-to-3.html>

[…]

For having gone through this in GPS, the most impacting change from the application developer's perspective was switching all the low-level drawing from Gdk to Cairo - the rest is fairly mechanical.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 1 Jul 2013 15:55:00 +0200
Subject: Re: Surprised by the changes to GTK 3.0
Newsgroups: gmane.comp.gnome.gtk+.ada

> […]

Well, other nasty stuff is:

1. RC files are gone. CSS is the replacement which lacks a good deal of features RC files had. E.g. stock images. BTW, Gtk promptly crashes on an attempt to load a non-existent CSS file.

2. "expose-event" is now "draw". That is not just already mentioned switching from GC to cairo. "draw" is propagated differently than "expose-event" did.

3. "size-request" signal is gone. If your custom widget needs resizing that has to be reworked completely. There is new "configure-event" for that. Unfortunately it is impossible to catch in some cases, even with Gtk_Event_Box added and various event masks set.

4. Gtk_Object is gone. All custom widgets derived from it must be redesigned.

5. There is no more messages loop quit handler. You should move cleanup to the application window or a widget.

6. Regarding GtkAda.

6.a It made many things tagged, but for reasons I don't understand it does not use Ada 2005 interfaces. As the result in old code you will have to add a lot of explicit conversions which weren't needed before. E.g.

```
Add_Model (View, To_Interface (Store));
```

Since it is Ada 2005 anyway, why Gtk_Tree_Model cannot be an interface Gtk_List_Store_Record implements?

6.b. Initialization of widgets, their classes are slightly different, yet enough different to break all custom widgets which register their own classes and properties.

6.c. Signal handlers must be library level. Of course, this is not a problem for a real-life project. But if you are accustomed writing small single-file unit tests, you should either split them into multiple files or else do ugly Unchecked_Conversions to work around accessibility checks.

Benefits of Ada on Small Embedded Systems

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 3 Jul 2013 14:19:02 -0500
Subject: Re: Help with embedded hardware/software platform selection for ADA
Newsgroups: comp.lang.ada

[…]

Ada was designed as a programming language for "programming in the large", and that means that its strengths don't really show up on tiny programs (which is what you can fit on tiny boards). That's a problem for Ada if you consider the tiny boards as an entry to working on larger systems down the road; so I'm not against efforts to use Ada on those sorts of systems -- I'm just dubious that they really can be successful (if they make Ada into "just another programming language", it's unclear that anyone will understand why Ada is so great).

From: Georg Bauhaus <bauhaus@maps.arcor.de>
Date: 03 Jul 2013 20:50:15 GMT
Subject: Re: Help with embedded hardware/software platform selection for ADA
Newsgroups: comp.lang.ada

> […]

This still seems pessimistic, in particular since Ada 2012.

Ada has a few features that single it out. It could not become just another programming language, even when tasking we dropped and exception handling limited.

1) The type system uses name equivalence for every kind of type.

2) If you define a scalar type, you know it, unlike int, or CARDINAL.

3) Structures are built from types, not from pointers and preprocessing definitions.

3) static binding of operations is the default.

4) Geared to explicit formality, not to artful exegesis of things implied.

5) change a type and have the compiler remind you of the other necessary changes.

When writing low level software in Ada, you can set the third bit of something by assigning True to a component of a packed array. In C, the main contender, you can pride yourself on having mastered C's shifting and masking. (Problem solving provides for combinatorial exercise already. Why more?)

McCormick's long term study shows how this has made a real difference, more than tasking has.

I think that Ada 2012 adds to that set. It is a good language for expressing exactly what you want to happen in the small, and a good language for describing interfaces of objects explicitly, in packaged types, including all scalar types, and bridled named pointer types.

A reasonably small Ada, therefore, could not be just like some other language, I think.

From: Eryndlia Mavourneen <eryndlia@gmail.com>
Date: Mon, 8 Jul 2013 05:53:24 -0700
Subject: Re: Help with embedded hardware/software platform selection for ADA
Newsgroups: comp.lang.ada

> […]

And don't forget the ability to specify the address of a variable -- Sooo much easier and clearer than using offsets and address arithmetic!

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Thu, 04 Jul 2013 01:02:48 +0300
Subject: Re: Help with embedded hardware/software platform selection for ADA
Newsgroups: comp.lang.ada

> […]

In addition to the syntax, you are gaining much of the conceptual and compile-time support of Ada, which is a vast improvement over C, IMO.

>> (Ada without exceptions and most tasking isn't Ada at all, IMHO).

Still much better than C.

>> RRS tried to serve that market back in the early days and got nowhere.

That was a shame, but I don't think it proves your point.

> […]

Present-day "tiny boards" or microcontrollers can have up to a few megabytes of code; more if off-chip memory is added. By 1983 standards, that qualifies as "large".

> […]

I concur with other replies that even an Ada with limited or no tasking and run-time support still has much of the goodness of Ada.

The Future of SPARK (and Ada)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 10 Jul 2013 18:10:15 -0500
Subject: Re: The future of SPARK . SPARK 2014 : a wreckage
Newsgroups: comp.lang.ada

> […]

I'm always presuming a proper globals annotation. You really can't do anything without it (Janus/Ada does almost no optimizations across subprogram calls precisely because Ada doesn't have this information in its contracts).

> Secondly, Ada's "in", "out", and "in out" information give only a cursory view of what is going on.

Of course. But that and the postcondition is all you ought to be depending upon. If you're depending on more, your program is more fragile than it ought to be.

… (detailed and possibly interesting examples of Depends removed)

…

> Without "Depends", your cool analysis tool verifying all pre- and postconditions will not catch this bug.

Sure, but so what? It's a fools game to try to catch all bugs, it not even a worthwhile goal. (That's essentially where SPARK goes wrong, in my view.)

If you had some magic annotations that could specify a totally bug-free program, then there no longer is any need for the program at all. Just execute the annotations, and forget the error-prone Ada code!

What I care about is extending what Ada already does well: catch the low-hanging fruit of bugs. There are always going to be a few really tough bugs that aren't going to be detectable automatically -- the goal should be to reduce that number in practical programs (no matter how large), not to eliminate all bugs in a barely usable subset of tiny programs.

Think about all of the "bugs" that we Ada programmers don't really have to deal with, because the compiler or runtime has already automatically detected it. That's not just type errors and array indexes out of range, but also dereferencing null pointers, accessing the wrong variant in a record, and many more. (I don't think Janus/Ada would have ever worked reliably without the variant check -- the early versions always had weird stability problems that disappeared as soon as we implemented the variant checks [and spent months eliminating all of the errors that turned up]).

What I think is important is bringing that level of ability to user-defined properties (think Is_Open and Mode for Text_IO files), and detecting more of these problems at compile-time (which is always better then runtime).

It's not being able to detect every possible bug.

My main point is that I think people are trying to solve the wrong problem, and that leads to having over-elaborate contracts.

> I agree that no one should depend on what the body of some Subprogram does. But then, information flow analysis is actually useful!

…in very marginal cases. I don't think it's worth trying to cover every possible base, certainly not before compilers even do a plausible job on the easy cases. Once every Ada compiler does basic proof using pre/post/globals/exception contracts, we can revisit.

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Thu, 11 Jul 2013 21:23:29 +0200
Subject: Re: The future of SPARK . SPARK 2014 : a wreckage
Newsgroups: comp.lang.ada

> […]

Your program can depend on whatever has been specified (and hopefully proven) in the specification. Which is one reason why "Depends" is actually useful -- you are allowed to make more specific contracts.

> […]

By my own experience, having worked occasionally with SPARK, information flow analysis actually catches a lot of errors -- even without having specified any pre- or postconditions. In other words, SPARK's information flow analysis actually gives you the low-hanging fruits you mention.

The fruits (well, bugs) you catch by employing pre- and postconditions are a bit higher, actually. At least, that is what I gather from my own experience YMMV.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 11 Jul 2013 19:21:16 -0500
Subject: Re: The future of SPARK . SPARK 2014 : a wreckage
Newsgroups: comp.lang.ada

> […]

You want to catch this very unlikely bug, and add a giant pile of extra junk annotations for this purpose. My contention is that it won't catch enough bugs by itself to be worth the complication.

> […] information flow analysis actually catches a lot of errors -- even without having specified any pre- or postconditions. […]

Preconditions (and constraints) come first. You write them before you write any bodies, before you write any comments even - they're an integral part of the specification of a subprogram. So the situation you speak of isn't even possible.

Moreover, the vast majority of the information in "Depends" is already in the Ada subprogram specification (once you include its pre and postconditions). The question is how much it adds to already properly annotated subprograms, not what happens when the tools are misused.

I don't care at all about "after-the-fact" addition of these things. Hardly anyone is going to be adding annotations to existing packages (it's not clear whether we're ever going to do that for the language-defined packages).

> The fruits (well, bugs) you catch by employing pre- and postconditions are a bit higher […]

Your experience seems to have been on annotating existing code, and doing it backwards ("depends" first). I can see why you might have annotated existing code that way, but that's not a goal or concern of mine. I'm only interested in new code, and code that's written properly (that is, the parameter modes tell one the data flow).

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Fri, 12 Jul 2013 11:12:26 +0200
Subject: Re: The future of SPARK . SPARK 2014 : a wreckage
Newsgroups: comp.lang.ada

> […]

Firstly, whatever data flow annotations are, they are not "junk".

Secondly, there is no "giant pile" of data flow annotations. Actually, they usually take a lot less lines of "anno-code" than writing pre- and postconditions. So even if you consider the data flow annotations as redundant, their overhead is small.

Thirdly, maybe my example has been too artificial. Below, I'll briefly describe one real-world example.

> Your experience seems to have been on annotating existing code,

Not at all! (Well, I tried once to SPARKify existing Ada code -- but I got rid of that disease very very quickly. Turning naturally-written Ada into proper SPARK is a pain in the you-know-where!)

One real life example (simplified for the purpose of posting this) is the implementation an authenticated encryption scheme. Consider two byte strings X and Y of the same length, X being the message and being Y the "key stream". There is additional authentication key K. The output of the authenticated encryption is the ciphertext (X xor Y), followed by a cryptographic checksum[*] of X under the key K.

Specifying and proving the first part of the output (X xor Y) was easy. But specifying and proving the checksum part turned out to be tough. So I stopped trying it -- concentrating on the low-hanging fruits, as you put it.

However, I still had the flow annotations in my spec. (I use to write the flow annotations first, then the pre- and postconditions, and then the implementation.) The flow annotations specified the flow from X and K to Z. And that actually caught my error of using (X xor Y) instead of X in the implementation.

> […] And it tries to do too much as well.

Agreed! Well, certainly there are people who require as much assurance as SPARK 05 provides, but I did find the work with old SPARK rather tedious, and I don't need that amount of assurance.

[*] The correct terminology would be "message authentication code" or "MAC", rather than "checksum".

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 12 Jul 2013 15:47:40 -0500
Subject: Re: The future of SPARK . SPARK 2014 : a wreckage
Newsgroups: comp.lang.ada

> […]

I called them "junk" because they're redundant (certainly in well-designed code). The OP complained that the proposed annotations for SPARK 2014, and I agree with him on that. But I find it irrelevant because they're redundant.

And I'm strongly opposed to putting redundant information in specifications or anywhere else. I've learned by painful experience over the years that redundant information -- in type declarations, a compiler symbol table, or a programming language standard -- always ends up out of sync with the original information. That's especially true if it cannot be automatically checked (I'm dubious that "Depends" could be checked that way given the information available to an Ada compiler). So, I want to eliminate as much as possible of that information.

Clearly, we're not getting rid of parameter modes. Clearly, we need preconditions and postconditions, they can't be done any other way. That makes "Depends" the redundant information that we should not have in a specification.

Moreover, I really don't see what value they could possibly have. A subprogram has a number of "in" parameters and a number of "out" parameters (and possibly, but hopefully not some input globals and output globals, and treating a function result as a "out" parameter for this discussion). All of the "in" parameters should somehow effect the "out" parameters (and it is best if there is only one output).

Routines that don't have this structure are already dubious. It sometimes can't be avoided, but it should be extremely rare. So, already, the extra information gained by this "flow" information is minimal. On top of that, "out" parameters that don't depend on some parameters are likely to be obvious in the postcondition for that parameter.

The point is that there isn't much information to be gained from such an annotation; the vast majority of it is repeating the obvious (inputs affect outputs). I realize that there are a lot of badly designed subprograms out there, but I wouldn't want a significant language feature just for badly designed subprograms -- especially when we still need lots of support for *well* designed subprograms!

> […]

The reason I said that if you design new code, you write the preconditions and postconditions along with the subprogram specification (many in the form of predicates, I would hope, as those are a lot easier than repeating large pre- and post-). (And you surely don't write subprograms where only some of the inputs affect some of the outputs.) Before you write anything else (like bodies or even descriptions of the purpose of the routine). I've never heard of any experienced Ada programmer writing subprogram specifications without considering the proper subtypes of the parameters immediately! So I don't see how you could get just "Depends" annotations.

I grant that you might "beef up" the preconditions and postconditions later, to provide a more complete description, but you're almost always going to start with some.

> [implementation an authenticated encryption scheme]

This sounds like precisely the bad sort of subprogram that typically should be avoided. Multiple outputs, and a strange non-dependence of some of the outputs on some of the inputs. I grant this can be unavoidable in some cases, but it should be rare. I don't see any point of an annotation that only provides extra information for such rare cases.

Probably a more sensible annotation here would be the negative: that is, declare which inputs a particular output does *not* depend upon. That normally should be a null list, in the rare case where it is non-null the information could be given.

But, as I said, this is not information useful to an Ada compiler (while the other annotations will improve code quality).

On Contracts and Implementations

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 12 Jul 2013 16:30:23 -0500
Subject: Re: On contracts and implementations
Newsgroups: comp.lang.ada

> […]

I understand your point, but I think it's short-sighted to think about removing the contracts. If you do so, then the contracts can't be used to "hoist" what otherwise have to be checks in the code (as in Text_IO, the containers, and many other packages). Turning off these checks in language-defined code is a complete non-starter, the result would not meet the language specifications (and would introduce erroneousness where none currently exists).

Secondly, as compiler's proof abilities increase, most of these contract assertions will disappear. The compiler will be able to prove that they always will succeed, and thus no check will be made.

After all, this is the case with existing constraints and exclusions. Compilers work overtime to eliminate unnecessary checks, and tend to remove 60-80% of the checks. It's rare that I see an explicit range check when I examine generated code these days. The same should happen to assertion checks (but only if they are well-written, using globals annotations and/or expression functions).

To look further at your example, I agree that evaluating Is_Sorted could be prohibitively expensive. But how did that object *get* sorted? One presumes that it was explicitly sorted somewhere, in which case that routine has an Is_Sorted postcondition. If the compiler can tie those together, there certainly is no need to reevaluate the Is_Sorted on a following precondition. Similarly, the compiler may have been able to prove that the Is_Sorted postcondition is always true. In that case, that won't be executed either. So you still will have the safety of checks on, and *still* have no actual overhead.

Now, obviously, these things won't always be next to each other; the array might have been a parameter. But in that case you can use a predicate to ensure that that parameter Is_Sorted, pushing the optimization to another level.

To summarize, I expect these to become much like range checks. You'll sometimes have to suppress them, but it will be pretty rare, and often you can add some subtypes in appropriate places to remove the checks rather than actually suppressing them (the latter being more dangerous).

Speed Test

From: Bill Findlay
<yaldnif.w@blueyonder.co.uk>
Date: Fri, 12 Jul 2013 02:01:29 +0100
Subject: Slow? Ada??
Newsgroups: comp.lang.ada

[…]

When debugging Whetstone Algol under my KDF9 emulator I looked at the KDF9 assembly code programming of the arctan function and was completely baffled by it, so I asked my former colleague, Michael Jamieson to investigate. He successfully reconstructed the mathematics behind it, which are very non-obvious (to put it mildly).

A couple of days ago I idly wondered how well this 50 years old algorithm would compare with modern implementations, so I wrote a test program to race it against GNAT GPL 2013's arctan. I expected to find that 2013 would spank 1963's botty. To my astonishment, the old algorithm was faster.

Digging down, I found that Ada.Numerics.Long_Elementary_Functions.arctan does quite a bit of argument range reduction and result error checking, but finally invokes the "fpatan" opcode of the x86_64 CPU.

The 1963 algorithm, expressed in Ada 2012 but compiled with aggressive optimization, is only 17% slower than that single CPU opcode!

Note also that, although it was designed for a machine with 48-bit floating point, it gets the same accuracy as the hardware method on a machine with 64-bit floating point.

Here is the code, with the observed results included as commentary:

```
with
Ada.Numerics.Long_Elementary_Functions;
with Ada.Text_IO;
with CPU_Timing;
with System.Machine_Code;

use
Ada.Numerics.Long_Elementary_Functions;
use  Ada.Text_IO;
use  CPU_Timing;
use  System.Machine_Code;

procedure arctan_test64 is

  -- typical output:
  -- R = 100000000 evaluations
  -- checksum = 5.00000005000000E+07,
  -- loop   time per repetition = 6 ns
  -- checksum = 4.38824577044418E+07,
  -- P51V15 time per evaluation = 49 ns
  -- checksum = 4.38824577044418E+07,
  -- arctan time per evaluation = 53 ns
  -- checksum = 4.38824577044418E+07,
  -- fpatan time per evaluation = 41 ns

  -- P51V15 is the KDF9 algorithm used in
  -- Whetstone Algol, ca. 1963
  -- See http://www.findlayw.plus.com/
  -- KDF9/Arctan%20Paper.pdf

  type vector is array (0 .. 5) of Long_Float;

  V : constant vector :=
           (28165298.0 / 1479104550.0,
            28165300.0 / 1479104550.0,
            56327872.0 / 1479104550.0,
            113397760.0 / 1479104550.0,
            179306496.0 / 1479104550.0,
           1073741824.0 / 1479104550.0);

  function P51V15 (x : Long_Float) return
           Long_Float with Inline;

  function P51V15 (x : Long_Float) return
           Long_Float is
    A : Long_Float := 1.0;
    S : Long_Float := V(0);
    B : vector;

  begin
    B(0) := sqrt (x * x + 1.0);
    -- 4 AGM (Arithmetic-Geometric Mean)
    -- cycles give a set of values …
    for i in 0 .. 3 loop
      A := (A + B (i)) * 0.5;
      B (i + 1) := sqrt (A * B (i));
    end loop;
    -- … that is subjected to a convergence
    -- acceleration process:
    for i in 1 .. 5 loop
      S := S + V (i) * B (i - 1);
    end loop;
    return x / S;
  end P51V15;

  -- this is the hardware arctan function of the
  -- x86_64 Core i7 CPU
  function fpatan (X : Long_Float) return
           Long_Float with Inline;

  function fpatan (X : Long_Float) return
           Long_Float is
    Result : Long_Float;

  begin
    Asm (Template => "fld1"
               & Character'Val (10) -- LF
               & Character'Val (9)  -- HT
               & "fpatan",
         Outputs  => Long_Float'Asm_Output
           ("=t", Result),
         Inputs   => Long_Float'Asm_Input
           ("0", X));
    return Result;
  end fpatan;

  R : constant := 1e8;
           -- number of loop repetitions

  function ns_per_rep
           (c : CPU_Usage_in_Microseconds)
           return Natural is

  begin
    return Natural (c * 1e3 / R);
  end ns_per_rep;

  x : Long_Float;
  c : CPU_Timer;
  l : CPU_Usage_in_Microseconds;
  t : CPU_Usage_in_Microseconds;

begin
  Put_Line ("R =" & Integer'Image (R) & "
           evaluations");

  -- determine the fixed overhead time
  x := 0.0;
  Reset_Timer (c);

  for i in 1 .. R loop
    x := x + Long_Float (i) / Long_Float (R);
  end loop;
  l := User_CPU_Time_Since (c);
  Put_Line ("checksum =" & x'Img & ", "
      & "loop   time per repetition ="
      & Natural'Image
           (ns_per_rep (l)) & "  ns");

  x := 0.0;
  Reset_Timer (c);

  for i in 1 .. R loop
    x := x + P51V15 (Long_Float (i) /
           Long_Float (R));
  end loop;
  t := User_CPU_Time_Since (c) - l;
  Put_Line ("checksum =" & x'Img & ", "
      & "P51V15 time per evaluation ="
      & Natural'Image
           (ns_per_rep (t)) & " ns");

  x := 0.0;
  Reset_Timer (c);

  for i in 1 .. R loop
    x := x + arctan (Long_Float (i) /
           Long_Float (R));
  end loop;
  t := User_CPU_Time_Since (c) - l;
  Put_Line ("checksum =" & x'Img & ", "
      & "arctan time per evaluation ="
      & Natural'Image
            (ns_per_rep (t)) & " ns");
  x := 0.0;
  Reset_Timer (c);

  for i in 1 .. R loop
     x := x + fpatan (Long_Float (i) /
            Long_Float (R));
  end loop;
  t := User_CPU_Time_Since (c) - l;
  Put_Line ("checksum =" & x'Img & ", "
       & "fpatan time per evaluation ="
       & Natural'Image
            (ns_per_rep (t)) & " ns");

end arctan_test64;
```

The difference in time between Ada.Numerics.Long_Elementary_Functions.arctan and the fpatan function above is due to the afore-mentioned range reduction and checking. The KDF9 programmers in 1963 were less punctilious about such matters than we rightly expect Ada to be nowadays.

Separate Private Part of Packages

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Thu, 25 Jul 2013 07:41:58 +0200
Subject: Re: GNAT GPL is proving...educational
Newsgroups: comp.lang.ada

> […]

Would be nice to even go further, with separate package's private part.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 25 Jul 2013 22:25:35 +0200
Subject: Re: GNAT GPL is proving...educational
Newsgroups: comp.lang.ada

> […]

This was considered for Ada 2012 - another of these nice little features that end up opening truck-loads of worms…

Low-level Programming

From: Paul Rubin
Date: Wed, 07 Aug 2013 21:39:58 -0700
Subject: Low-level programming in Ada?
Newsgroups: comp.lang.ada

I'm wondering if anyone can suggest a reference (preferably online) about low-level programming (e.g. for operating system implementation) in Ada. Not about the language itself, but examples of dealing with machine addresses, device registers, page tables, memory management, etc., preferably without dropping to assembler more than a tiny bit.

This isn't for a specific project or anything like that. It's just general interest in how to do this stuff that's traditionally the domain of C.

From: Michael Erdmann <michael.erdmann@snafu.de>
Date: 08 Aug 2013 13:47:47 GMT
Subject: Re: Low-level programming in Ada?
Newsgroups: comp.lang.ada

> […]

Maybe you should have a look at florist:

http://en.wikibooks.org/wiki/Ada_Programming/Platform/POSIX

http://www.cs.fsu.edu/~baker/florist.html

For the assembler stuff; if you really need it refer to the GNAT manual and what is written in the gnuasm manual; e.g.

http://tigcc.ticalc.org/doc/gnuasm.html

From: Eryndlia Mavourneen <eryndlia@gmail.com>
Date: Thu, 8 Aug 2013 07:59:04 -0700
Subject: Re: Low-level programming in Ada?
Newsgroups: comp.lang.ada

> […]

```
with System;

package Addresses is

  Data_Address : constant
  System.Address_Type := 16#22000#;
  Data : Integer;
  for Data'Address use Data_Address;

begin
  Data := 555;

end Addresses;
```

Of course, for device registers and the like, you will want to use record representation clauses to specify the bit fields, etc.

Using these techniques will allow you to delve into the dark world of virtual-to-physical address translation, paging, or what-have-you. :-)

All of this is sooooo much easier than messing around with offsets, shifts, and such in C and its offspring.

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: 8 Aug 2013 21:17:45 GMT
Subject: Re: Low-level programming in Ada?
Newsgroups: comp.lang.ada

> […]

You might like to have a look at the code for my KDF9 emulator, which deals with similar issues around the representation of hardware structures:

http://www.findlayw.plus.com/KDF9/emulation/emulator.html#About

Have a look in particular at the kdf9*.ad? files.

From: Mike Hopkins <postmaster@ada-augusta.demon.co.uk>
Date: Sat, 10 Aug 2013 12:25:03 +0100
Subject: Re: Low-level programming in Ada?
Newsgroups: comp.lang.ada

> […]

I have only done it once and it is nearly 30 years since I did it, but I remember starting by writing a mirror of the memory map of the target device as an Ada package. Perhaps my memories are rose-tinted but, from there, it all seemed to grow naturally.

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 13 Aug 2013 08:14:54 +0000
Subject: Re: Low-level programming in Ada?
Newsgroups: comp.lang.ada

> […]

Have a look at the Ada bare bones I wrote on osdev.org wiki.

When Is Formal Verification Appropriate

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu, 15 Aug 2013 15:02:17 +0200
Subject: SPARK vs. Ada 2012 for static analysis (Was: Ada 2012 talk at DANSAS'13)
Newsgroups: comp.lang.ada

Paul Rubin wrote:

> Does SPARK-2014 help? I'm not sure if it exists yet, but I gather GNATProve (which is out there) is some kind of precursor to it.

SPARK would of course give compile/analysis-time checking, but we don't consider it appropriate for the project. The cost of implementing sockets and containers in SPARK alone would probably kill that idea.

SPARK-2014 wasn't announced when we started the project, and I don't consider it ready for real-life use, as tasking isn't covered yet.

We want as much static analysis as we can get, but we are at the same time working in a context where the value of making an "absolutely perfect" application isn't that much bigger than writing a "definitely above average reliability" application. The whole system also depends on other parts, which may fail too. As long as we have a few orders of magnitude fewer failures than the other parts, we are quite happy.

As I see things, the important place for complete static analysis (i.e. SPARK) is in components which have a unique possibility of breaking your system. One obvious example is a PRNG used for cryptography; if it is broken, your whole system is broken, and nothing else can break the system in quite the same way.

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 15 Aug 2013 07:01:21 -0700
Subject: Re: SPARK vs. Ada 2012 for static analysis (Was: Ada 2012 talk at DANSAS'13)
Newsgroups: comp.lang.ada

> […]

I see what you're saying, but I [somewhat] disagree: the scope you're using is too small. it's the small "everybody uses it and assumes it's correct" things that need SPARK-verification.

Take DNS for example: there's a *lot* of bugs that have been found in any main DNS0server over the past decade [or two]. Everybody using the internet is interacting with a DNS, even if indirectly. And so it behooves us to eliminate everything [bug-wise] that we can -- which is what formal verification does, and it's what the twp guys who developed Ironsides did. (http://ironsides.martincarlisle.com/)

The two papers linked just above the "Download" heading are quite informative and say things much better than I can.

I would love to see a formally-verified OS, and to be honest MS would have a much better product to sell if they did so to their OS instead of worrying about "looking stylish". (See the Windows 8 disaster).

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.

2013

☺ October 03	ICPP2013 - **International Workshop on Embedded Multicore Systems** (EMS'2013), Lyon, France. Topics include: programming models for embedded multicore systems; software for Multicore, GPU, and embedded architectures; real-time system designs for embedded multicore environments; applications for automobile electronics of multicore designs; compiler for worst-case execution time analysis; formal method for embedded systems; etc.
October 03-04	5th **International Workshop on Software Engineering for Resilient Systems** (SERENE'2013), Kiev, Ukraine. Topics include: relations between resilience, dependability and quality attributes; requirements engineering & re-engineering for resilience; error, fault and exception handling in the software life-cycle; verification and validation of resilient systems; empirical studies in the domain of resilient systems; global aspects of resilience engineering: education, training and cooperation; frameworks, patterns and software architectures for resilience; etc.
October 10-11	7th **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2013), Baltimore, Maryland, USA. Topics include: qualitative methods; replication of empirical studies; empirical studies of software processes and products; industrial experience and case studies; evaluation and comparison of techniques and models; reports on the benefits / costs associated with using certain technologies; empirically-based decision making; quality measurement and assurance; software project experience and knowledge management; etc.
October 14-17	20th **Working Conference on Reverse Engineering** (WCRE'2013), Koblenz, Germany. Topics include: program comprehension, reengineering to distributed systems, mining software repositories, software architecture recovery, empirical studies in reverse engineering, program analysis and slicing, re-documenting legacy systems, reengineering patterns, program transformation and refactoring, reverse engineering tool support, etc.
October 20-23	13th **International Conference on Formal Methods in Computer-Aided Design** (FMCAD'2013), Portland, Oregon, USA. Co-located with MEMOCODE'2013 and DIFTS'2013. Topics include: theory and application of formal methods in hardware and system design and verification; modeling and specification languages, model-based design, correct-by-construction methods, experience with the application of formal and semi-formal methods to industrial-scale designs, application of formal methods in new areas, etc.
October 26-28	6th **International Conference on Software Language Engineering** (SLE'2013), Indianapolis, Indiana, USA. Topics include: formalisms used in designing and specifying languages and tools that analyze such language descriptions; language implementation techniques; program and model transformation tools; language evolution; approaches to elicitation, specification, or verification of requirements for software languages; language development frameworks, methodologies, techniques, best practices, and tools for the broader language lifecycle; design challenges in SLE; applications of languages including innovative domain-specific languages or "little" languages; etc.
☺ October 26-31	ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2013), Indianapolis, Indiana, USA.

☺ Oct 26-31 28th **Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2013). Topics include: any aspect of programming, systems, languages, and applications; any aspect of software development, including requirements, modeling, prototyping, design, implementation, generation, analysis, verification, testing, evaluation, maintenance, reuse, replacement, and retirement of software systems; large-scale software repositories; tools (such as new languages, program analyses, or runtime systems) or techniques (such as new methodologies, design processes, code organization approaches, and management techniques) that go beyond objects in interesting ways; etc.

Oct 27 - Nov 01 8th **International Conference on Software Engineering Advances** (ICSEA'2013), Venice, Italy. Topics include: advances in fundamentals for software development; advanced mechanisms for software development; advanced design tools for developing software; software security, privacy, safeness; specialized software advanced applications; open source software; agile software techniques; software deployment and maintenance; software engineering techniques, metrics, and formalisms; software economics, adoption, and education; improving productivity in research on software engineering; etc.

Oct 29 - Nov 11 15th **International Conference on Formal Engineering Methods** (ICFEM'2013), Queenstown, New Zealand. Topics include: abstraction and refinement; program analysis; software verification; formal methods for software safety, security, reliability and dependability; tool development, integration and experiments involving verified systems; formal methods used in certifying products under international standards; formal model-based development and code generation; etc.

November 04-07 24th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2013), Pasadena, CA, USA.

♦ Nov 10-14 ACM SIGAda **Annual International Conference on High Integrity Language Technology** (HILT'2013), Pittsburgh, Pennsylvania, USA.

November 13-16 15th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2013), Osaka, Japan. Topics include: fault-tolerance and dependability, formal methods and distributed systems, etc.

November 17-22 26th **International Conference for High Performance Computing, Networking, Storage and Analysis** (SC'2013), Denver, Colorado, USA. Topics include: applications, programming systems (technologies that support parallel programming, such as compiler analysis and optimization, parallel programming languages and notations, programming models, runtime systems, tools, software engineering for parallel programming, solutions for parallel programming challenges, ...), state of the practice, etc.

December 02-05 20th **Asia-Pacific Software Engineering Conference** (APSEC'2013), Bangkok, Thailand. Topics include: software engineering methodologies; software analysis and understanding; software testing, verification and validation; software maintenance and evolution; software quality and measurement; software process and standards; software security, reliability and privacy; software engineering environments and tools; software engineering education; distributed and parallel software systems; embedded and real-time software systems; formal methods in software engineering; etc.

December 09-11 11th **Asian Symposium on Programming Languages and Systems** (APLAS'2013), Melbourne, Australia. Topics include: foundational and practical issues in programming languages and systems, such as semantics, design of languages and type systems, domain-specific languages, compilers, interpreters, abstract machines, program analysis, verification, model-checking, software security, concurrency and parallelism, tools and environments for programming and implementation, etc.

December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

☺ December 15-18 19th IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2013), Seoul, Korea. Topics include: parallel and distributed applications and algorithms, multi-core and multithreaded architectures, security and privacy, dependable and trustworthy computing and systems, real-time systems, cyber-physical systems, embedded systems, etc.

December 18-21 20th IEEE **International Conference on High Performance Computing** (HiPC'2013), Hyderabad, India. Topics include: parallel and distributed algorithms / applications, parallel languages and programming environments, hybrid parallel programming with GPUs, scheduling, resilient/fault-tolerant algorithms and systems, scientific/engineering/commercial applications, compiler technologies for high-performance computing, software support, etc.

2014

January 09-11 15th IEEE **International Symposium on High Assurance Systems Engineering** (HASE'2014), Miami, Florida, USA. Topics include: tools and techniques used to design and construct systems that, in addition to meeting their functional objectives, are safe, secure, and reliable.

January 20-22 9th **International Conference on High-Performance and Embedded Architectures and Compilers** (HiPEAC 2014), Vienna, Austria. Topics include: processor, memory, and storage systems architecture; parallel, multi-core and heterogeneous systems; architectural support for programming productivity; architectural and run-time support for programming languages; programming models, frameworks and environments for exploiting parallelism; compiler techniques, etc.

January 20 2nd **Workshop on High-performance and Real-time Embedded Systems** (HiRES 2014). Topics include: runtimes and operating systems combining high-performance and predictability requirements; programming models and compiler support for providing real-time capabilities to multi- and many-core architectures, models and tools for code generation, system verification and validation, etc.

☺ January 22-24 41st ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2014), San Diego, USA. Topics include: all aspects of programming languages and systems, with emphasis on how principles underpin practice.

Jan 20-21 ACM SIGPLAN **Workshop on Partial Evaluation and Program Manipulation** (PEPM'2014). Topics include: program and model manipulation techniques (such as: partial evaluation, slicing, symbolic execution, refactoring, ...); program analysis techniques that are used to drive program/model manipulation (such as: abstract interpretation, termination checking, type systems, , ...); techniques that treat programs/models as data objects (including: metaprogramming, generative programming, embedded domain-specific languages, model-driven program generation and transformation, ...); etc. Application of the above techniques including case studies of program manipulation in real-world (industrial, open-source) projects and software development processes, descriptions of robust tools capable of effectively handling realistic applications, benchmarking. Deadline for submissions: October 5, 2013 (papers).

February 12-14 22nd Euromicro **International Conference on Parallel, Distributed and Network-Based Computing** (PDP'2014), Turin, Italy. Topics include: embedded parallel and distributed systems, multi- and many-core systems, programming languages and environments, runtime support systems, simulation of parallel and distributed systems, dependability and survivability, real-time distributed applications, etc.

February 19-21 7th **India Software Engineering Conference** (ISEC'2014), Chennai, India. Topics include: static analysis, specification and verification, model-driven software engineering, component-based software engineering, embedded and real-time systems, software security, software architecture and design, development paradigms, tools and environments, maintenance and evolution, software engineering education, multicore software engineering, etc. Deadline for submissions: October 1, 2013 (workshops, tutorials).

February 26-28 6th **International Symposium on Engineering Secure Software and Systems** (ESSoS'2014), Munich, Germany. Topics include: security architecture and design for software and systems; specification formalisms for security artifacts; verification techniques for security properties; systematic support for security best practices; programming paradigms, models and DSL's for security; processes for the development of secure software and systems; support for assurance, certification and accreditation; security by design; etc.

March 24-28 29th ACM **Symposium on Applied Computing** (SAC'2014), Gyeongju, Korea.

☺ Mar 24-28 **Track on Programming Languages** (PL'2014). Topics include: compiling techniques, domain-specific languages, formal semantics and syntax, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, programming languages from all paradigms, etc.

☺ Mar 24-28 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2014). Topics include: aspects and components, distribution and concurrency, formal verification, integration with other paradigms, software evolution, language design and implementation, modular and generic programming, secure and dependable software, static analysis, type systems, etc.

Mar 24-28 **Track on Software Verification and Testing** (SVT'2014). 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.

March 24-28 **Design, Automation & Test in Europe** (DATE 2014), Dresden, Germany.

March 24-28 **Track on Embedded Systems Software** (Track E). Topics include: real-time, networked, and dependable systems, compilation and code generation for embedded software, model-based design and verification for embedded systems, embedded software architectures, cyber-physical systems.

Mar 31- Apr 04 7th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2014), Cleveland, Ohio, USA. Topics include: embedded software testing, testing concurrent software, testing large-scale distributed systems, testing in multi-core environments, security testing, quality assurance, inspections, testing of open source and third-party software, software reliability, formal verification, empirical studies of testing techniques, experience reports, etc.

April 05-13 **European Joint Conferences on Theory and Practice of Software** (ETAPS'2014), Grenoble, France. Events include: CC, International Conference on Compiler Construction; ESOP, European Symposium on Programming; FASE, Fundamental Approaches to Software Engineering; FOSSACS, Foundations of Software Science and Computation Structures; POST, Principles of Security and Trust; TACAS, Tools and Algorithms for the Construction and Analysis of Systems. Deadline for submissions: October 4, 2013 (abstracts), October 11, 2013 (papers).

April 22-26 13th **International Conference on Modularity** (Modularity'2013), Lugano, Switzerland. Topics include: varieties of modularity (generative programming, aspect orientation, software product lines, components; ...); programming languages (support for modularity related abstraction in: language design; verification, contracts, and static program analysis; compilation, interpretation, and runtime support; formal languages; ...); software design and engineering (evolution, empirical studies of existing software, economics, testing and verification, composition, methodologies, ...); tools (refactoring, evolution and reverse engineering, support for new language constructs, ...); applications (distributed and concurrent systems, middleware, cyber-physical systems, ...); complex systems; etc. Deadline for submissions: October 13, 2013 (round 2).

Apr 29 - May 05 6th **NASA Formal Methods Symposium** (NFM'2014), NASA Johnson Space Center, Houston, Texas, USA. Topics include: identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems; static analysis; model-based development; applications of formal methods to aerospace systems; correct-by-design and design for verification techniques; techniques and algorithms for scaling formal methods, e.g. abstraction and symbolic methods, compositional techniques, parallel and distributed techniques; application of formal methods to emerging technologies; etc. Deadline for submissions: November 14, 2013 (abstracts), November 21, 2013 (papers).

May 13-16 10th **European Dependable Computing Conference** (EDCC'2014), Newcastle upon Tyne, UK. Topics include: hardware and software architecture of dependable systems, safety critical systems, embedded and real-time systems, impact of manufacturing technology on dependability, testing and validation methods, privacy and security of systems and networks, etc. Deadline for submissions: October 13, 2013 (papers).

☺ June 01-07 36th **International Conference on Software Engineering** (ICSE'2014), Hyderabad, India.

June 16-20 26th **International Conference on Advanced Information Systems Engineering** (CAiSE'2014), Thessaloniki, Greece. Topics include: methods, techniques and tools for IS engineering (models and software reuse; adaptation, evolution and flexibility issues; languages and models; variability and configuration; security; ...); innovative platforms, architectures and technologies for IS (model-driven architecture; component based development; distributed and open architecture; ...); etc. Deadline for submissions: October 18, 2013 (workshops), November 29, 2013 (papers), December 13, 2013 (tutorials).

♦ June 23-27	19th **International Conference on Reliable Software Technologies - Ada-Europe'2014**, Paris, France. Sponsored by Ada-Europe, in cooperation requested with ACM SIGAda, SIGBED, SIGPLAN. Deadline for submissions: December 8, 2013 (papers, tutorials, workshops), January 19, 2014 (industrial presentations).
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

Association for Computing Machinery
2 Penn Plaza, Suite 701
New York, NY 10201-0701
USA

ACM's High Integrity Language Technology Conference
HILT 2013 Advance Program

Pittsburgh, Pennsylvania, USA / November 10–14, 2013
www.sigada.org/conf/hilt2013 *Sponsored by ACM SIGAda*

Come to HILT 2013 and discover the latest developments in language technology for safe, secure, and reliable software.

Listen to and meet world-renowned experts in the field, see how industry is converting research into practical experience, and learn both the challenges confronting high-integrity software and the solutions available to address them.

REGISTER ONLINE BY OCTOBER 21 FOR THE LOWEST REGISTRATION RATES

Association for Computing Machinery

Advancing Computing as a Science & Profession

Visit www.sigada.org/conf/hilt2013

Association for Computing Machinery

Advancing Computing as a Science & Profession

HILT 2013: HIGH INTEGRITY LANGUAGE TECHNOLOGY

ACM SIGAda's Annual International Conference

November 10–14, 2013 / Pittsburgh, Pennsylvania / Advance Program

High integrity software must not only meet correctness and performance criteria but also satisfy stringent safety and/or security demands, typically entailing certification against a relevant standard.

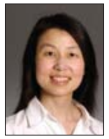
A significant factor affecting whether and how such requirements are met is the chosen language technology and its supporting tools: not just the programming language(s) but also languages for expressing specifications, program properties, domain models, and other attributes of the software or overall system.

HILT 2013 will provide a forum for experts from academia/research, industry, and government to present their latest findings in designing, implementing, and using language technology for high integrity software.

Sponsored by SIGAda, ACM's Special Interest Group on the Ada Programming Language, in cooperation with SIGAPP, SIGBED, SIGCAS, SIGCSE, SIGPLAN, SIGSOFT, Ada-Europe, and the Ada Resource Association.

FEATURED SPEAKERS

Model Checking: Past, Present, and Future
EDMUND M. CLARKE
Carnegie Mellon University
Electrical and Computer Engineering (ECE)



Formal Methods: An Industrial Perspective
JEANNETTE WING
Microsoft Research

Building Confidence in System Behavior
JOHN GOODENOUGH
Carnegie Mellon University
Software Engineering Institute (SEI)

Model-Based Engineering
MICHAEL WHALEN
University of Minnesota

CORPORATE SPONSORS

PLATINUM LEVEL

AdaCore
The GNAT Pro Company

GOLD LEVEL

Microsoft Research

SILVER LEVEL

Ellidiss Software

VEROCEL
The Software Verification Company

BASIC LEVEL

LDRA

MathWorks®

PRE-CONFERENCE TUTORIALS / November 10–11

SUNDAY

SA1—Morning / 9:00 AM–12:30 PM
Ed Colbert / Absolute Software
Ada 2012 Part 1

SA2—Morning / 9:00 AM–12:30 PM
Tucker Taft / AdaCore
Proving Safety of Parallel/Multi-Threaded Programs

SP1—Afternoon / 2:00 PM–5:30 PM
Ed Colbert / Absolute Software
Ada 2012 Part 2

SP2—Afternoon / 2:00 PM–5:30 PM
Ethan K. Jackson / Microsoft Research
Formula 2.0: A Language for Formal Specifications; A Tool for Automated Analysis

MONDAY

MA1—Morning / 9:00 AM–12:30 PM
Nikolaj Bjorner / Microsoft Research
Satisfiability Modulo Theories for High Integrity Development

MA2—Morning / 9:00 AM–12:30 PM
Francesco Logozzo / Microsoft Research
Practical Specification and Verification with CodeContracts

MP1—Afternoon / 2:00 PM–5:30 PM
Sagar Chaki / SEI
Bounded Model Checking for High-Integrity Software

MP2—Afternoon / 2:00 PM–5:30 PM
Ricky Sward / Mitre Corporation
Jeff Boleng / SEI
Service-Oriented Architecture (SOA) Concepts and Implements

TECHNICAL PROGRAM / November 12–14

TUESDAY

9:00 AM–10:30 AM
Greetings
SIGAda and Conference Officers

Keynote Address
Edmund Clarke, CMU/ECE
Model Checking: Past, Present, and Future

10:30 AM–11:00 AM Break / Exhibits

11:00 AM–12:30 PM
Panel on Underlying Formal Verification Technolgies
Topics to be covered: Model Checking, SAT Solvers and SMT Solvers, Static Analysis and Abstract Interpretation, Coq-Based Proofs

Sponsor Presentation

12:30 PM–2:00 PM Break / Exhibits

2:00 PM–3:30 PM
Formal Verification Toolsets

J. Hendrix
SAW: The Software Analysis Workbench

A. Hawthorn
Optimizing Development and Verification Effort with SPARK 2014

Z. Zhang
Towards the Formalization of SPARK 2014 Semantics with Explicit Run-time Checks Using Coq

3:30 PM–4:00 PM Break / Exhibits

4:00 PM–5:30 PM
High-Integrity Parallel Programming

Panel on Safe, Efficient Parallel Programming
Topics to be covered: Real-Time Programming on Accelerator Many-Core Processors, Bringing Parallel Programming to the SPARK Verifiable Subset of Ada, Deadlock Detection for Ada 2012

Sponsor Presentation

5:30 PM–6:00 PM Break

6:00 PM–10:00 PM
Social Event / Dinner

WEDNESDAY

9:00 AM–10:30 AM
Announcements
SIGAda Awards
Ricky E. Sward, Past SIGAda Chair

Invited Talk
Michael Whalen, University of Minnesota
Model-Based Engineering

10:30 AM–11:00 AM Break / Exhibits

11:00 AM–12:30 PM
Model-Based Integration and Code Generation

D. Ward
An Approach to Integration of Complex Systems: The SAVI Virtual Integration Process (industrial presentation)

M. Beeby
Using Autocode Generators for Avionics Systems and Maintaining Compliance to DO-178 and DO-331 (industrial presentation)

Industrial Presentation

12:30 PM–2:00 PM Break / Exhibits

2:00 PM–3:30 PM
Keynote Address
John Goodenough, CMU/SEI
Building Confidence in System Behavior

3:30 PM–4:00 PM Break

4:00 PM–5:30 PM
Architecture-Level Design Languages and Compositional Verification

A. Murugesan
Compositional Verification of a Medical Device System

B. Larson
Illustrating the AADL Error Modeling Annex (v. 2) Using a Simple Safety-Critical Medical Device

Sponsor Presentation

5:30 PM–7:00 PM Break

7:00 PM–10:00 PM
Workshops / Birds-of-a-Feather Sessions

THURSDAY

9:00 AM–10:30 AM
Announcements
Best Paper and Student Paper Awards
Tucker Taft, HILT 2013 Program Chair

Keynote Address
Jeannette Wing, Microsoft Research
Formal Methods: An Industrial Perspective

10:30 AM–11:00 AM Break

11:00 AM–12:00 PM
Panel on Approaches to Software Safety and Security
Topics to be covered: Secure Coding, Static Analysis, Formal Verification, Automatic vs. Interactive Program Verification

12:00 PM–12:30 PM
Announcements
(Ada-Europe 2014, SIGAda 2014)

Closing Remarks and Conference Adjournment

To register online, and for more information and updates, visit
www.sigada.org/conf/hilt2013

Photo of Pittsburgh by User:Derek.cashman courtesy of Wikimedia Commons

VENUE / HOTEL

HILT 2013 will be held at the Wyndham Pittsburgh University Center, *www.tinyurl.com/HILT2013-hotel*.

The Wyndham Pittsburgh University Center has reserved a block of rooms for the HILT 2013 conference. The conference rate is $119 for single, double, triple, or quadruple occupancy rooms. This includes complimentary wireless Internet in all the guest rooms and free parking for guests. State and local taxes will be added per night. All reservations must be guaranteed by credit card. Please also visit *www.sigada.org/conf/hilt2013/hotel-rates.html* and *www.acm.org/sig_volunteer_info/whyhotel.htm* for additional details. For directions from Pittsburgh International Airport (PIT) as well as information about transportation to and from the airport via Super Shuttle, Taxi Services, and Bus Service, please visit *www.pitairport.com*, or *www.pitairport.com/public_transportation* for public transport options.

SPONSORS / EXHIBITORS

HILT 2013 will include vendor participation, featuring presentations on their products and services during main sessions. For specific information, please contact the Exhibits Chair, Greg Gicca, *gicca@verocel.com*.

GRANTS TO EDUCATORS

As in past years, SIGAda is offering grants to educators to attend the conference. Grants cover the registration and tutorial fees; members of the GNAT Academic Program may be eligible for travel funds from AdaCore. Apply by e-mail, no later than October 14, 2013. Grant program details are available from the conference website or Professor Michael B. Feldman, *mfeldman@gwu.edu*.

WORKSHOPS / BIRDS-OF-A-FEATHER

To propose a focused workshop or informal Birds-of-a-Feather session related to the conference theme, please contact the Workshops Chair, John W. McCormick, *mccormick@cs.uni.edu*.

REGISTRATION FEES

CONFERENCE (FULL)	CONFERENCE (ONE DAY)	TUTORIAL (FULL DAY)	TUTORIAL (HALF DAY)
Member of ACM, SIGAda, or cooperating organization:	Member of ACM, SIGAda, or cooperating organization:	Member of ACM, SIGAda, or cooperating organization:	Member of ACM, SIGAda, or cooperating organization:
$575 early / $725 after Oct. 21	**$325 early** / $325 after Oct. 21	**$310 early** / $370 after Oct. 21	**$155 early** / $185 after Oct. 21
Non-members:	Non-members:	Non-members:	Non-members:
$875 early / $975 after Oct. 21	**$325 early** / $325 after Oct. 21	**$420 early** / $470 after Oct. 21	**$210 early** / $235 after Oct. 21
Full-time Student: $50	Full-time Student: $25	Full-time Student: $30	Full-time Student: $15

For early registration rates, register online by October 21 at **http://sigada.org/conf/hilt2013/register/index.html**

CONFERENCE TEAM

Conference Chair / Local Arrangements Chair
Jeff Boleng, Software Engineering Institute / *JLBoleng@SEI.CMU.edu*

Program Chair / Proceedings Chair
Tucker Taft, AdaCore / *taft@adacore.com*

Treasurer
Ricky E. Sward, The MITRE Corporation / *rsward@mitre.org*

Workshops Chair / Tutorials Chair
John W. McCormick, University of Northern Iowa / *mccormick@cs.uni.edu*

Webmaster
Clyde Roby, Institute for Defense Analyses / *clyderoby@acm.org*

Exhibits and Sponsorships Chair
Greg Gicca, Verocel / *gicca@verocel.com*

Registration Chair / Academic Community Liaison
Michael B. Feldman, George Washington University (Ret.) / *mfeldman@gwu.edu*

Publicity Chair
Alok Srivastava, TASC Inc. / *alok.srivastava@tasc.com*

Logo Designer
Weston Pan, Raytheon Space and Airborne Systems

SIGAda Officers

Chair
David Cook, Stephen F. Austin State University / *cookda@sfasu.edu*

Vice Chair
Tucker Taft, AdaCore / *taft@adacore.com*

Secretary / Treasurer
Clyde Roby, Institute for Defense Analyses / *clyderoby@acm.org*

International Representative
Dirk Craeynest, KU Leuven, Department of Computer Science / *dirk.craeynest@cs.kuleuven.be*

Past Chair
Ricky E. Sward, The MITRE Corporation / *rsward@mitre.org*

ACM Ada Letters Editor
Alok Srivastava, TASC Inc. / *alok.srivastava@tasc.com*

Ada-Europe 2014

Call for Papers

19th International Conference on Reliable Software Technologies – Ada-Europe 2014

23-27 June 2014, Paris, France

http://www.ada-europe.org/conference2014

General Information

The 19th International Conference on Reliable Software Technologies – Ada-Europe 2014 will take place in Paris, France. As per its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical and industrial programs, along with tutorials and workshops on Monday and Friday.

Schedule

8 December 2013	Submission of regular papers, tutorial and workshop proposals
19 January 2014	Submission of industrial presentation proposals
16 February 2014	Notification of acceptance to all authors
16 March 2014	Camera-ready version of regular papers required
18 May 2014	Industrial presentations, tutorial and workshop material required

Topics

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

Topics of interest to this edition of the conference include but are not limited to:

- **Multicore and Manycore Programming**: Predictable Programming Approaches for Multicore and Manycore Systems, Parallel Programming Models, Scheduling Analysis Techniques.
- **Real-Time and Embedded Systems**: Real-Time Scheduling, Design Methods and Techniques, Architecture Modelling, HW/SW Co-Design, Reliability and Performance Analysis.
- **Theory and Practice of High-Integrity Systems**: Challenges from Mixed-Criticality Systems; Medium to Large-Scale Distribution, Fault Tolerance, Security, Reliability, Trust and Safety, Languages Vulnerabilities.
- **Software Architectures**: Design Patterns, Frameworks, Architecture-Centred Development, Component-based Design and Development.
- **Methods and Techniques for Software Development and Maintenance**: Requirements Engineering, Model-driven Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- **Enabling Technologies**: Compilers, Support Tools (Analysis, Code/Document Generation, Profiling), Run-time Systems and Libraries.
- **Software Quality:** Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Cloud Environments, Smart Energy systems, Serious Games, etc.
- **Experience Reports in Reliable System Development**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **Experiences with Ada and its Future**: Reviews of the Ada 2012 new language features; implementation and use issues; positioning in the market and in the software engineering curriculum; lessons learned on Ada Education and Training Activities with bearing on any of the conference topics.

General Chair
Jean-Pierre Rosen
Adalog
rosen@adalog.fr

Program co-Chairs
Laurent George
LIGM/UPEMLV - ECE Paris
lgeorge@ieee.org

Tullio Vardanega
University of Padova
tullio.vardanega@unipd.it

Industrial Chair
Jørgen Bundgaard
Rambøll Denmark A/S
jogb@ramboll.dk

Tutorial co-Chairs
Liliana Cucu
INRIA
Liliana.Cucu@inria.fr

Albert Llemosí
Universitat de les Illes Balears
albert.llemosi@uib.cat

Exhibition Chair
To be appointed

Publicity co-Chairs
Jamie Ayre
AdaCore
ayre@adacore.com

Dirk Craeynest
Ada-Belgium & KU Leuven
Dirk.Craeynest@cs.kuleuven.be

Local Chair
Magali Munos
ECE
munos@ece.fr

In cooperation requested with ACM SIGAda, SIGBED, SIGPLAN

acm In-Cooperation

Call for Regular Papers

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall exceed 14 LNCS-style pages in length. Authors shall submit their work via EasyChair following the relevant link on the conference web site. The format for submission is solely PDF.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the start of the conference. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by March 16, 2014. For format and style guidelines authors should refer to *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The CORE ranking (dated 2008) has the conference in class A. The CiteSeerX Venue Impact Factor had it in the top quarter. Microsoft Academic Search has it in the top third for conferences on programming languages by number of citations in the last 10 years. The conference is listed in DBLP, SCOPUS and Web of Science Conference Proceedings Citation index, among others.

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Industrial Presentations

The conference seeks industrial presentations which deliver value and insight but may not fit the selection process for regular papers. Authors are invited to submit a presentation outline of exactly 1 page in length by January 19, 2014. Submissions shall be made via EasyChair following the relevant link on the conference web site. The *Industrial Committee* will review the submissions and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by May 18, 2014, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the *Industrial Chair* directly.

Call for Tutorials

Tutorials should address subjects that fall within the scope of the conference and may be proposed as either half- or full-day events. Proposals should include a title, an abstract, a description of the topic, a detailed outline of the presentation, a description of the presenter's lecturing expertise in general and with the proposed topic in particular, the proposed duration (half day or full day), the intended level of the tutorial (introductory, intermediate, or advanced), the recommended audience experience and background, and a statement of the reasons for attending. Proposals should be submitted by e-mail to the *Tutorial Chair*. The authors of accepted full-day tutorials will receive a complimentary conference registration as well as a fee for every paying participant in excess of 5; for half-day tutorials, these benefits will be accordingly halved. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference week. Workshop proposals should be submitted to the *General Chair*. The workshop organizer shall also commit to preparing proceedings for timely publication in the Ada User Journal.

Call for Exhibitors

The commercial exhibition will span the three days of the main conference. Vendors and providers of software products and services should contact a *Conference Co-Chair* for information and for allowing suitable planning of the exhibition space and time.

Grant for Reduced Student Fees

A limited number of sponsored grants for reduced fees is expected to be available for students who would like to attend the conference or tutorials. Contact the *General Chair* for details.

Rationale for Ada 2012: Epilogue

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 is the last of a number of papers describing the rationale for Ada 2012. In due course it is anticipated that the papers will be combined (after appropriate reformatting and editing) into a single volume for formal publication.

This last paper summarizes a small number of general issues of importance to the user such as compatibility between Ada 2012 and Ada 2005. It also briefly revisits a number of problems that were considered for Ada 2005 but rejected for various reasons; the important ones have been solved in Ada 2012.

Finally, it discusses a small number of corrections that have been found necessary since the standard was approved.

Keywords: rationale, Ada 2012.

1 Compatibility

There are two main sorts of problems regarding compatibility. These are termed Incompatibilities and Inconsistencies.

An incompatibility is a situation where a legal Ada 2005 program is illegal in Ada 2012. These can be annoying but not a disaster since the compiler automatically detects such situations.

An inconsistency is where a legal Ada 2005 program is also a legal Ada 2012 program but might have a different effect at execution time. These can in principle be really nasty but typically the program is actually wrong anyway (in the sense that it does not do what the programmer intended) or its behaviour depends upon the raising of a predefined exception (which is generally considered poor style) or the situation is extremely unlikely to occur.

As mentioned below in Section 2, during the development of Ada 2012 a number of corrections were made to Ada 2005 and these resulted in some incompatibilities and inconsistencies with the original Ada 2005 standard. These are not considered to be incompatibilities or inconsistencies between Ada 2005 and Ada 2012 and so are not covered in this section.

1.1 Incompatibilities with Ada 2005

Each incompatibility listed below gives the AI concerned and the paragraph in the ARM which in some cases will give more information. Where relevant, the section in this rationale where the topic is discussed is also given. Where appropriate the incompatibilities are grouped together.

Note that this list only covers those incompatibilities that might reasonably occur. There are a number of others which are so unlikely that they do not seem worth mentioning.

1 – The word **some** is now reserved. Programs using it as an identifier will need to be changed. (AI-176, 2.9)

Adding new reserved words is a very visible incompatibility. Six were added in Ada 95, three in Ada 2005, and now just one in Ada 2012. Perhaps this is the end of the matter. The word **some** is used in quantified expressions; it already was reserved in SPARK [1] where it is used in quantified expressions in proof contexts.

2 – If a predefined package has additional entities then incompatibilities can arise. Thus suppose the predefined package Ada.Stuff has an additional entity More added to it. Then if an Ada 2005 program has a package P containing an entity More then a program with a use clause for both Ada.Stuff and P will become illegal in Ada 2012 because the reference to More will become ambiguous. This also applies if further overloadings of an existing entity are added.

This can be overcome by adding child packages of course. However, adding lots of child packages can be an inconvenience for the user and so in many cases extending a package seemed more appropriate especially if the identifiers concerned are unlikely to have been used by programmers.

The following packages have been extended with additional entities as listed.

Ada.Characters.Handling – Is_Line_Terminator, Is_Mark, Is_Other_Format, Is_Punctuation_Connector, Is_Space. (AI-185, A.3.2)

Ada.Containers – Capacity_Error. (AI-1, A.18.1)

Ada.Containers.Vectors – Assign, Copy, Constant_Reference, Constant_Reference_Type, Iterate, Reference, Reference_Type, Vector_Iterator_Interfaces. (AI-1, AI-212, A.18.2)

There are similar additions to the other containers Ada.Containers.Doubly_Linked_Lists etc.

Ada.Directories – Name_Case_Kind, Name_Case_Equivalence. (AI-49, A.16)

Ada.Dispatching – Yield. (AI-166, D.2.1)

Ada.Environment_Variables – Value. (AI-285, A.17)

Ada.Execution_Time – Interrupt_Clocks_Supported, Separate_Interrupt_Clocks_Supported, Clocks_For_Interrupts. (AI-170, D.14)

Ada.Task_Identification – Environment_Task, Activation_Is_Complete. (AI-189, C.7.1)

Ada.Strings.Fixed – Find_Token. (AI-31, A.4.3)

Ada.Strings.Bounded – Find_Token. (AI-31, A.4.4)

Ada.Strings.Unbounded – Find_Token. (AI-31, A.4.5)

There are similar additions to Ada.Strings.Wide_Fixed, Ada.Strings.Wide_Bounded and Ada.Strings.Wide_Unbounded. (AI-31, A.4.7)

Ada.Tags – Is_Abstract. (AI-173, 3.9)

It seems unlikely that existing programs will be affected by these potential incompatibilities.

3 – Membership tests are no longer allowed as a discrete choice. This is explained in detail in Section 6 of the paper on Expressions. (AI-158, 3.8.1)

4 – Allowing functions to have parameters of all modes led to the introduction of stricter rules on aliasing. It is possible that a program that seemed to work in Ada 2005 is illegal in Ada 2012. See Section 2 of the paper on Structure and Visibility. (AI-144, 6.4.1)

5 – Implicit conversion is now allowed from anonymous access types to general access types. Such conversions can make calls ambiguous in the presence of overloading where only one call was permitted in Ada 2005. Consider

```
type RT is access all T;
function F return RT;
function F return access T;

procedure B(R: RT);
```

and then the call

```
B(F);            -- ambiguous in Ada 2012
```

The call of B is ambiguous in Ada 2012 because the call could be to either function F. But in Ada 2005, the implicit conversion is not possible and so the call has to be to the first function F. (AI-149, 8.6)

6 – It is now illegal to declare a formal abstract subprogram whose controlling type is incomplete. This is related to various improvements to incomplete types described in Section 3 of the paper on Structure and Visibility. (AI-296, 12.6)

7 – The pragma Controlled has been removed from the language. It was never implemented anyway. (AI-229, 13.11.3)

8 – The package Ada.Dispatching was Pure in Ada 2005 but has been downgraded to Preelaborable because of the addition of Yield. This is unlikely to be a problem. (AI-166, D.2.1)

1.2 Inconsistencies with Ada 2005

Note that this list only covers those inconsistencies that might reasonably occur. There are a number of others which are so unlikely that they do not seem worth mentioning.

1 – The definition of character sets can change with time. It is thus possible that the result of character classification functions for obscure characters might be or become inconsistent. (AI-91, AI-227, AI-266, 2.1, 2.3)

2 – User defined untagged record equality is now defined to compose and be used in generics. Code which assumes that predefined equality reemerges in generics and in predefined equals for composite types could fail. However, it is more likely that this change will fix bugs. (AI-123, 4.5.2)

3 – A stand alone object of an anonymous access type now has dynamic accessibility. This is most likely to make illegal programs now legal. However, it is possible that a program that raised Program_Error in Ada 2005 will not do so in Ada 2012. It seems very unlikely that a program would rely on the raising of this exception. (AI-148, 4.6)

4 – There is an obscure interaction between the change to the composability of equality and renaming. Renaming of user-defined untagged record equality is now defined to call the overridden body so long as the overriding occurred before the renames. Consider

```
package P is
  type T is
    record
      ...
    end record;
                  -- (1) consider renaming here
private
  function "=" (L, R: T) return Boolean;
end P;

with P;
package Q is
  function Equals renames P."=";
end Q;
```

In Ada 2005, Equals refers to the predefined equality, whereas in Ada 2012 it refers to the overridden user-defined equality in the private part. This is so that composed equality and explicit calls on "=" give the same answer. However, if the renaming had been at the point (1) then calling Equal would call the predefined equality. Remember that renaming squirrels away the operation so that it can be retrieved. (AI-123, 8.5.4)

5 – A group budget is now defined to work on a single processor. However, it is unlikely that any implementation of Ada 2005 managed to implement this on multiprocessors anyway. (AI-169, D.14.2)

2 Retrospective changes to Ada 2005

In the course of the development of Ada 2012, a number of small changes were deemed to apply also to Ada 2005 and thus were classified as binding interpretations rather than amendments. Some were mentioned in previous papers (including that which ensured that package Ada is legal); see Sections 2 and 6 of the paper on Iterators, Pools etc. Most of these do not introduce incompatibilities or inconsistencies so will not be discussed further.

A few binding interpretations do introduce minor incompatibilities or inconsistencies and will now be briefly discussed.

2.1 Incompatibilities with original Ada 2005

There are a small number of incompatibilities between the original Ada 2005 and that resulting from various corrections.

1 – The rules for full conformance have been strengthened; for example, null exclusions must now match. (AI-46, AI-134, AI-207, 6.3.1)

2 – When an inherited subprogram is implemented by a protected function, the first parameter has to be an **in** parameter, but not an access to variable type. Ada 2005 allowed access to variable parameters in this case; the parameter will need to be changed to access to constant by the addition of the **constant** keyword. (AI-291, 9.4)

3 – A missing rule is added that a limited with clause cannot name an ancestor unit. (AI-40, 10.1.2)

4 – Matching of formal access to subprogram types uses subtype conformance in Ada 2012 whereas it only used mode conformance in original Ada 2005. This change was necessary to avoid undefined behaviour in some situations. (AI-288, 12.5.4)

5 – An address attribute with a prefix of a subprogram with convention Intrinsic is now illegal. This is discussed in Section 6 of the paper on Iterators, Pools etc. (AI-95, 13.3)

6 – Stream attributes must be specified by a static subprogram name rather than by a dynamic expression. (AI-39, 13.13.2)

7 – The use of discriminants on Unchecked_Union types is now illegal in record representation clauses. It makes no sense to specify the position of something that is not supposed to exist. (AI-26, B.3.3)

8 – A nonvolatile generic formal derived type precludes a volatile actual type. (AI-218, C.6)

9 – The restriction No_Relative_Delay has been extended to also prohibit a call of Timing_Events.Set_Handler with a Time_Span parameter. (AI-211, D.7)

10 – Various restrictions have been reworded to prevent the bypassing of the restriction by calling the forbidden subprogram via renames. (AI-211, D.7)

2.2 Inconsistencies with original Ada 2005

There are a small number of inconsistencies between the original Ada 2005 and that resulting from various corrections.

1 – The description of Dependent_Tag has been changed to say that it must raise Tag_Error if there is more than one type that matches the requirements. (AI-113, 3.9)

2 – A curious omission regarding checking arrays allows a component in an aggregate whose value is given as <> even if the component is outside the bounds. It is now clarified that Constraint_Error is raised. (AI-37, 4.3.3)

3 – The first procedure Split in Ada.Calendar.Formatting raises Time_Error for a value of exactly 86400.0. This was unspecified in Ada 2005. (AI-238, 9.6.1)

4 – An address attribute with a prefix of a generic formal subprogram whose actual parameter has convention Intrinsic now raises Program_Error. (AI-95, 13.3)

5 – User specified external tags that conflict with other external tags now raise Program_Error or are illegal. (AI-113, 13.3)

6 – The definition of Set_Line is corrected. As originally defined in Ada 95 and Ada 2005, Set_Line(1) could call New_Line(0) which would raise Constraint_Error which is unhelpful. This was mentioned right at the end of the Postscript in the Rationale for Ada 2005 [2]. (AI-38, A.10.5)

7 – The definitions of Start_Search, Search, Delete_Directory, and Rename are clarified so that they raise the correct exception if misused. (AI-231, A.16)

8 – If Count = 0 for a container Insert subprogram that has a Position parameter, the Position parameter is set to the value of the Before parameter by the call. The original wording remained silent on this. (AI-257, A.18.3)

3 Unfinished topics from Ada 2005

A number of topics which seemed to be good ideas initially were abandoned during the development of Ada 2005 for various reasons. Usually the reason was simply that a good solution could not be produced in the time available and the trouble with a bad solution is that it is hard to put it right later. This section briefly reconsiders these topics which were discussed in the Rationale for Ada 2005 [2]; some have now been solved in Ada 2012; the others were considered unimportant.

3.1 Aggregates for private types

The <> notation was introduced in Ada 2005 for aggregates to mean the default value if any. A curiosity is that we can write

```
type Secret is private;

type Visible is
  record
    A: Integer;
    S: Secret;
  end record;

X: Visible := (A => 77; S => <>);
```

but we cannot write

```
S: Secret := <>;                    -- illegal
```

The argument is that this would be of little use since the components take their default values anyway.

For uniformity it was proposed that we might allow

```
S: Secret := (others => <>);
```

for private types and also for task and protected types. One advantage would be that we could then write

```
S: constant Secret := (others => <>);
```

whereas it is not possible to declare a constant of a private type because we are unable to give an initial value.

However, discussion of this issue led into a quagmire in Ada 2005 and so was abandoned. It remains abandoned in Ada 2012!

3.2 Partial generic instantiation

Certain attempts to use signature packages led to circularities in Ada 95. Consider

```
generic
  type Element is private;
  type Set is private;
  with function Union(L, R: Set) return Set  is <>;
  with function Intersection(L, R: Set) return Set is <>;
  ... -- and so on
package Set_Signature is end;
```

Remember that a signature is a generic package consisting only of a specification. When we instantiate it, the effect is to assert that the actual parameters are consistent and the instantiation provides a name to refer to them as a group.

If we now attempt to write

```
generic
  type Elem is private;
  with function Hash(E: Elem) return Integer;
package Hashed_Sets is
  type Set is private;
  function Union(L, R: Set) return Set;
  function Intersection(L, R: Set) return Set;
  ...
  package Signature is new Set_Signature(Elem, Set);
private
  type Set is
    record
      ...
    end record;
end Hashed_Sets;
```

then we are in trouble. The problem is that the instantiation of Set_Signature tries to freeze the type Set prematurely.

After a number of false starts this problem is partially overcome in Ada 2012 by the introduction of incomplete formal generic parameters. This is discussed in Section 3 of the paper on Structure and Visibility. See also Section 4.1 of this paper.

3.3 Support for IEEE 559: 1989

The proposal was to provide full support for all aspects of IEEE 559 arithmetic such as NaNs (a NaN is Not A Number). This would have necessitated adding attributes such as S'Infinity, S'Is_NaN, S'Finite and so on plus a package Ada.Numerics.IEC_559.

The proposal was abandoned because it would have had a big impact on implementers and it was not clear that there was sufficient demand. It was not reconsidered for Ada 2012.

3.4 Defaults for generic parameters

Generic subprogram parameters and object parameters of mode in can have defaults. But other parameters such as packages and types cannot. This was considered irksome and untidy and efforts were made to define a suitable notation for all possible generic parameters.

However, it was abandoned partly because an appropriate syntax seemed hard to find and more importantly, it was not felt to be that important. Again, it was not deemed important enough to be reconsidered for Ada 2012.

3.5 Pre/post-conditions for subprograms

The original proposal was to add pragmas such as Pre_Assert and Post_Assert. Thus in the case of a subprogram Push on a type Stack we might write

```
procedure Push(S: in out Stack; X: in Item);
pragma Pre_Assert(Push, not Is_Full(S));
pragma Post_Assert(Push, not Is_Empty(S));
```

This was all abandoned in Ada 2005 for various reasons; one being that pragmas are ugly for such an important matter.

However, this is neatly solved in Ada 2012 by the introduction of aspect specifications so we can now write

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

which is really excellent; this is discussed in detail in the paper on Contracts and Aspects.

3.6 Type and package invariants

This defined further pragmas similar to those in the previous proposal but concerned with packages and types. Thus the pragma Package_Invariant proposed for Ada 2005 identified a function returning a Boolean result. This function would be implicitly called after the call of each subprogram in the package and if the result were false the behaviour would be as for an Assert pragma that failed.

This proposal was also abandoned for Ada 2005. However, Ada 2012 has introduced type invariants thus

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

as discussed in the paper on Contracts and Aspects. On the other hand, package invariants remain abandoned.

3.7 Exceptions as types

This proposal originally arose out of a workshop organized by Ada-Europe. It was quite complex and considered far too radical a change and probably expensive to implement. As a consequence it was slimmed down considerably. But having been slimmed down it seemed pointless and was then abandoned. The only part to survive was the idea of raise with message which became a separate AI and was incorporated into Ada 2005.

This was not pursued in Ada 2012.

3.8 Sockets operations

This seemed a very good idea at the time but no detailed proposal was forthcoming and so it died. It has been left dead.

3.9 In out parameters for functions

The proposal was to allow functions to have parameters of all modes. The rationale for the proposal was well summarized thus "Ada functions can have arbitrary side effects, but are not allowed to announce that in their specifications".

But strangely, this AI was abandoned quite early in the Ada 2005 revision process on the grounds that it was "too late". (Perhaps too late in this context meant 25 years too late.)

However, in Ada 2012, the bullet has been bitten and functions can indeed now have parameters of all modes. See the discussion in Section 2 of the paper on Structure and Visibility.

3.10 Application defined scheduling

The International Real-Time Ada Workshops have been a source of suggestions for improvements to Ada. The Workshop at Oporto suggested a number of further scheduling algorithms [3]. Most of these such as Round Robin and EDF were included in Ada 2005. But that for application defined scheduling was not.

No further action on this topic was taken in Ada 2012.

4 Unfinished topics for Ada 2012

A number of topics which seemed to be good ideas initially were abandoned during the development of Ada 2012 for various reasons. It is interesting to note that there are far fewer of these loose ends than there were in Ada 2005. The following deserve mention.

4.1 Integrated packages (AI-135)

Difficulties sometimes arise with nested packages. Consider for example a package that needs to export a private type T and a container instantiated for that type. We cannot write

```
package P is
  type T is private;
  package T_Set is new Ordered_Sets(T);
private
  ...
end P;
```

because the type T is not frozen. We have to write something like

```
package P is
  package Inner is
    type T is private;
  private
    ...
  end Inner;
  package T_Set is new Ordered_Sets(Inner.T);
end P;
```

What we now want is some way to say that the declarations in Inner are really at the level of P itself after all. In other words we want to integrate the package Inner with the outer package P.

Various attempts were made to solve this by another kind of use clause or perhaps by putting Inner in a <> box. But all attempts led to difficulties so this remains unresolved.

4.2 Cyclic fixed point (AI-175)

Measurements in the physical world of Euclid and Newton are either lengths or angles. Angles are cyclic in nature and so can be mapped with a modular type. However, this leaves scaling in the hands of the user and is machine dependent. Consideration was given to the possibility of a cyclic form of fixed point. Sadly, there was much hidden complexity and so no solution was agreed.

One might have thought that it would be easy to use the natural wrap-around hardware. However, with a binary machine, if 180 degrees is held exactly then 60 degrees is not which excludes an exact representation of an equilateral triangle. The whole point about using fixed point is that it is precise but it just doesn't work unless the hardware uses a base with divisibility by 60. The Babylonians would have understood. The text of AI-175 includes a generic which might be useful for many applications.

4.3 Global annotations (AI-186)

The idea here was that the specification of a subprogram should have annotations indicating the global objects that it might manipulate. For example a function can have side effects on global variables but this important matter is not mentioned in the specification. This topic has strong synergy with the information given in contracts such as pre- and postconditions. However, it was abandoned perhaps because of the complexity arising from the richness of the full Ada language. It should be noted that such annotations have always featured in SPARK as comments and moreover, at the time of writing, are being considered using the aspect notation in a new version of SPARK.

4.4 Shorthand for assignments (AI-187)

Consideration was given to having some short of shorthand for assignments where source and target have commonality as in statements such as

```
A(I) := A(I) + 1;
```

But maybe the thought of C++ was too much. In any event no agreement that it was worthwhile was reached and there was certainly no agreement on what syntax might be acceptable.

5 Postscript

It should also be noticed that a few corrections and improvements have been made since Ada 2012 was approved as a standard. The more important of these will now be discussed.

A new form of expression, the raise expression, is added (AI12-22). This means that by analogy with

```
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);
```

A raise expression is a new form of relation so the syntax for relation (see Section 6 of the paper on Expressions) is extended as follows

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

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

Since a raise expression is a relation it 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";
```

On a closely related topic the new syntax for membership tests (also see Section 6 of the paper on Expressions) has been found to cause ambiguities (AI12-39).

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 yet again to

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

and changing

membership_choice ::=
  choice_expression | range | subtype_mark

to

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.

A curious difficulty has been found in attempting to use the seemingly innocuous package Ada.Locales described in Section 4 of the paper on the Predefined Library.

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 (AI12-37).

Readers might like 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.

AI12-45 notes that pre- and postconditions are allowed on generic units but they are not allowed on instances. See Section 3 of the paper on Contracts and Aspects where this topic should have been mentioned.

Another modification in this area is addressed by AI12-44 which states that type invariants are not checked on **in** parameters of functions but are checked on **in** parameters of procedures. See Section 4 of the paper on Contracts and

Aspects. This change was necessary to avoid infinite recursion which would arise if an invariant itself called a function with a parameter of the type. Note also that a class wide invariant could not be used at all without this modification.

A further aspect, Predicate_Failure, is defined by AI12-54-2. The expected type of the expression defined by this aspect 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". See Section 5 of the paper on Contracts and Aspects.

We can also use a raise expression 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.

This can be illustrated by an extension of the above example. Suppose we have

```
subtype Open_File_Type is File_Type
  with
     Dynamic_Predicate => Is_Open(Open_File_Type),
     Predicate_Failure => raise Status_Error;

subtype Read_File_Type is Open_File_Type
  with
    Dynamic_Predicate =>
                        Mode(Real_File_Type) = In_File,
    Predicate_Failure => raise Mode_Error with
          "Can't read file: " & Name(Read_File_Type);
```

The subtype Read_File_Type refines Open_File_Type. If the predicate for it were evaluated first and the file was not open then the call of Mode would raise Status_Error which we would not want to happen if we wrote

```
if F in Read_File_Type then ...
```

Care is needed with membership tests. The whole purpose of a membership test (and similarly the Valid attribute) is to find out whether a condition is satisfied. So if we write

```
if X in S then
   ...              -- do this
else
   ...              -- do that
end if;
```

we expect the membership test to be true or false. However, if the evaluation of S itself raises some exception then the purpose of the test is violated.

It is important to understand these related topics. Another example might clarify. Suppose we have a very simple predicate as in Section 5 of the paper on Contracts and Aspects such as

```
subtype Winter is Month
  with Static_Predicate => Winter in Dec | Jan | Feb;
```

where

```
type Month is (Jan, Feb, Mar, Apr, ..., Nov, Dec);
```

and we declare a variable W thus

```
W: Winter := Jan;
```

If we now do

```
W := Mar;
```

then Assertion_Error will be raised because the value Mar is not within the subtype Winter (we assume that the assertion policy is Check). If, however, we would rather have Constraint_Error raised then we can modify the declaration of Winter to

```
subtype Winter is Month
  with Static_Predicate => Winter in Dec | Jan | Feb,
       Predicate_Failure => raise Constraint_Error;
```

and then obeying

```
W := Mar;
```

will raise Constraint_Error.

On the other hand suppose we declare a variable M thus

```
M: Month := Mar;
```

and then do a membership test

```
if M in Winter then
   ...              -- do this if M is a winter month
else
   ...              -- do this if M is not a winter month
end if;
```

then of course no exception is raised since this is a membership test and not a predicate check.

Note however, that we could write something odd such as

```
subtype Winter2 is Month
  with Dynamic_Predicate =>
    (if Winter2 in Dec | Jan | Feb then true else raise E);
```

then the very evaluation of the predicate might raise the exception E so that

```
M in Winter2
```

will either be true or raise the exception E but will never be false. Note that in this silly example the predicate has to be a dynamic one because a static predicate cannot include a raise expression.

So this should clarify the reasons for introducing Predicate_Failure. It enables us to give a different behaviour for when the predicate is used in a membership test as opposed to when it is used in a check and it also allows us to add a message.

Finally, it should be noted that the predicate expression might involve the evaluation of some subexpression perhaps through the call of some function. We might have a predicate describing those months that have 30 days thus

```
subtype Month30 is Month
  with Static_Predicate =>
                  Month30 in Sep | Apr | Jun | Nov;
```

which mimics the order in the nursery rhyme. However, suppose we decide to declare a function Days30 to do the check so that the subtype becomes

```
subtype Month30 is Month
  with Dynamic_Predicate => Days30(Month30);
```

and for some silly reason we code the function incorrectly so that it raises an exception (perhaps it accidentally runs into its **end** and always raises Program_Error). In this situation if we write

```
M in Month30
```

then we will indeed get Program_Error and not false.

Perhaps this whole topic can be summarized by simply saying that a membership test is not a check. Indeed a membership test is often useful in ensuring that a subsequent check will not fail as was discussed in Section 4 of the paper on Iterators, Pools etc.

On a rather different topic, AI12-28 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.

Finally, an important modification is made to the topic of dispatching domains by AI12-33. See Section 3 of the paper on Tasking and Real-Time.

As defined originally, 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, the package System.Multiprocessors.Dispatching_Domains is extended by the addition of a type CPU_Set and two further functions 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;
```

So if we want to create a domain consisting of processors 0, 4, and 8 we can write

```
My_Set: CPU_Set(0 .. 8) :=
                  (0 | 4 | 8 => true, others => false);
```

and then

```
My_Domain: Dispatching_Domain := Create(My_Set);
```

and so on. The function Get_CPU_Set can be applied to any domain and returns the appropriate array representing the set of CPUs. Note that this function can be applied to any domain and not just to one created from a CPU_Set.

6 Acknowledgements

This is the last of the papers in this series and so this seems a good moment to once more thank Randy Brukardt for his diligence and patience in reviewing various drafts and putting me back on track when I got lost.

I must also thank AdaCore and the British Standards Institute for financial support for attending various meetings.

As usual, writing this rationale has been a learning experience for me and I trust that readers will also have found the material useful in learning about Ada 2012. An integrated description of Ada 2012 as a whole will be found in a forthcoming version of a familiar textbook.

References

John Barnes (2012) *SPARK – The proven approach to High Integrity Software,* Altran Praxis.

John Barnes (2008) *Ada 2005 Rationale*, LNCS 5020, Springer-Verlag.

ACM (2003) *Proceedings of the 12th International Real-Time Ada Workshop*, Ada Letters, Vol 32, No 4.

Using the GNAT environment to maintain a large codebase inherited from another compilation system

Daniel Bigelow

Bigelow Informatics, Bern, Switzerland; email: daniel.bigelow@bigelow.ch

Abstract

This paper describes a two-phase strategy using GNAT technologies to gain control over a large codebase inherited from another compilation system. In phase-one we create an environment to contain the state of an incremental release for the purpose of extension in a developer workspace. In phase-two we systematically incorporate project files representing key components into a domain-model to reflect the top-level architecture. Phase-one quickly provides a code-view of the system so that development work can begin. The transition to phase-two requires more time, but eventually brings the architecture of the system to the surface where it is visible to all stakeholders.

Keywords: Ada, porting, maintenance, architecture

1 Introduction

When using the GNAT environment to develop a large application from scratch, we reference architecture diagrams and other supporting artefacts to logically organize the software into a system of interconnected components, which are encapsulated in loosely-coupled modules that comprise a subsystem. Each software component is represented by one or more GNAT project files (GPFs) and the modules and subsystems are organized as directory structures. By applying good design principles, even huge systems can be constructed and efficiently maintained using this methodology.

However, after porting a large application to GNAT from another compilation system we don't have the benefit of an existing set of projects files to reflect the architecture. Instead, what we have are the following raw materials: the code-base, a long list of main programs, several directories containing related units, a top-level domain model (if we are lucky), and tons of documentation which is usually out of date. The sections that follow describe a solution for taking this scenario and moving forward with GNAT.

2 GNAT project file

Programming is a creative activity and therefore we tend to jump into it as soon as possible. However, without a well-designed process to guide development activities and an efficient development environment for making changes and managing complexity, the joy of programming comes to an end somewhere around 100-thousand lines of code or even less if your application is safety-critical and therefore subject to a formal certification process.

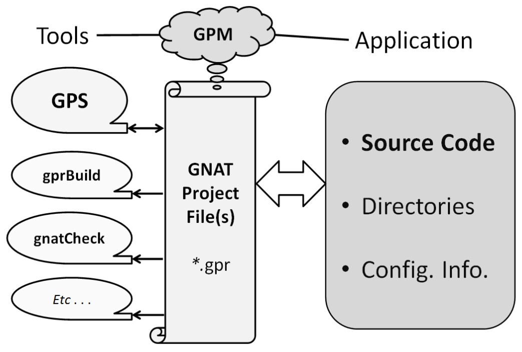


Figure 1. GPFs connect the application to the tool-chain

The GNAT Project Manager (GPM) facility uses the GNAT Project File (GPF) as the keystone for connecting the application to the tool-chain (Figure 1). For small to medium-sized projects the GNAT Programming Studio (GPS) is all you should need to create and configure GPFs to your requirements; there should be no need to manually edit the GPF. For large systems however, the GPF might require use of advanced features not supported by the GUI in GPS. In that case, it is necessary for at least one person on the development team to have a good understanding of the syntax and semantics of GPF code in order to setup the best possible environment for development and production purposes.

3 Compiling inherited code with GNAT

Large applications originally developed with Rational-Apex, ObjectAda, DEC-Ada, etc, often contain compiler-specific dependencies that prevent GNAT from compiling the entire codebase on the first attempt. For example, if the code makes direct use of declarations in package *System* and/or package *Standard* then portability issues will arise, especially if predefined numeric types are used in conjunction with representation, size and alignment clauses.

Fortunately, the GNAT environment provides facilities for extending the set of definitions in package *System* with those from another compiler. For example, when porting from DEC-Ada, the GPF can specify the attribute *Global_Configuration_Pragmas* to point to a file containing *pragma Extend_System (Aux_DEC)* which has the effect of creating the child package *System.Aux_Dec*. If necessary, it is possible to override one or more Ada runtime library packages with non-GNAT declarations by passing the '*-gnatg*' switch to *gprbuild.*

Each compiler has its own personality traits which make it unique, but also complicate the porting activity. For example, when porting from Rational Apex one needs to deal with the notion of an Apex subsystem which is top-level enclosure containing related Ada packages. Each Apex subsystem is decomposed into so-called "views" (subdirectories) to support configuration management, where each view represents a specific version of the subsystem. Also, each view specifies an interface that defines the set of Ada units visible to another (importing) view. To facilitate the task of porting from Rational Apex, GNAT provides *pragma Profile (Rational)* and other supporting mechanisms.

4 Loading the ported codebase into GPS

Once the codebase is able to compile with GNAT, the next step is to setup the development environment to take full advantage of the GNAT Programming Studio (GPS) which integrates all required tools into one place.

4.1 Medium-sized system

With the computing power available in today's PCs, GPS able to load several thousand lines of code at once and still provide good performance for all software development activities including configuration as illustrated in Figure 2.

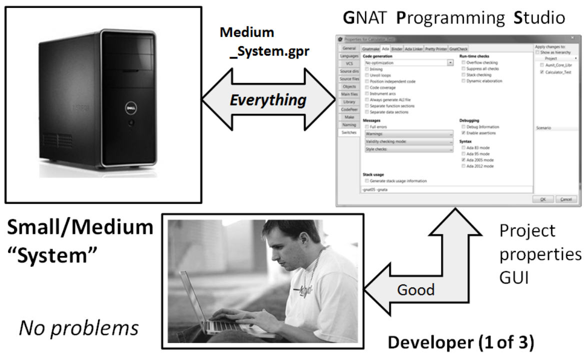


Figure 2. Medium-sized system loaded into GPS

4.2 Large-sized system

For large systems the scenario shown in Figure 2 does not work due to information overload. It is simply not practical from the perspective of workstation-performance or the ability of a developer to confront millions of lines of code all at once. Even if one has the patience to wait 10 or 15 minutes for several thousand packages to load into the GPS entity database, any attempt to perform development activities will be frustrated by additional processing delays. Therefore, the scenario in Figure 3 is not used.

5 Divide & Conquer

So what is the solution if you want to use the GNAT development environment with a large application that has just been ported from another compilation system? The answer of course is to break the problem down into manageable parts.

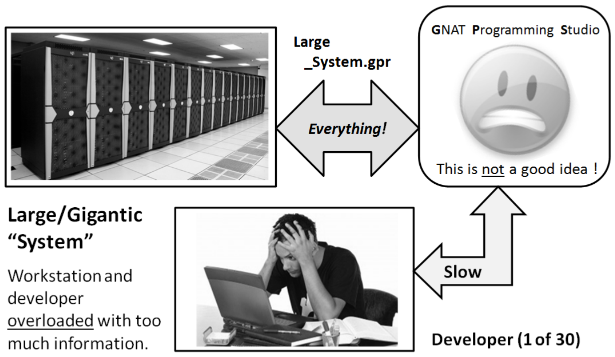


Figure 3. Large-sized system loaded into GPS

5.1 Subsystem view

Fortunately, most large systems in production or in the planning stages employ "loosely-coupled" subsystems that communicate with each other via middleware. Because middleware prevents direct compilation dependencies between subsystems, we can build, and to a large extent also develop & test each subsystem as a stand-alone entity (Figure 4).

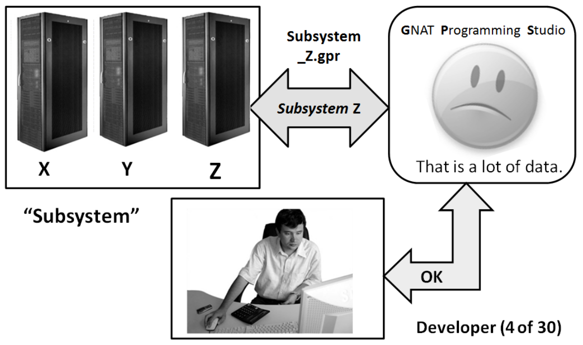


Figure 4. Subsystem loaded into GPS

A given subsystem normally has characteristics that are very different from the others and therefore, a separate GPF will exist to represent each subsystem. Examples of real-world project file names are subsystem_Search_Radar.gpr and subsystem_Engine_Control.gpr.

By selecting the subsystem view, we pass all sources to the workspace that is visible to the subsystem-project. For a large project a subsystem normally contains a lot of code and therefore, the subsystem-view represents the maximum amount of code you would ever want to load into GPS at any one time.

This view is not normally used for development purposes but rather for examining the "big picture" by browsing disparate regions of the code-base, checking conformance to programming standards, running suites of regression tests, measuring code-coverage, and building library and executable components for both development and production environments.

If your system is large and does not support loosely-coupled subsystems or even subsystems for that matter, then your application is probably monolithic and difficult to maintain. Nevertheless, assuming the code-base contains several main programs, the techniques described in this paper may still be applied to gain control and extend the service life of your application using the GNAT development environment.

6 Canonical subsystem architecture

For the purpose of this discussion, each subsystem uses the layered architecture shown in Figure 5. The Framework layer consists of self-contained modules that provide library services. Note that some framework modules, such as “Domain-Specific Types” would be deployed on each subsystem.

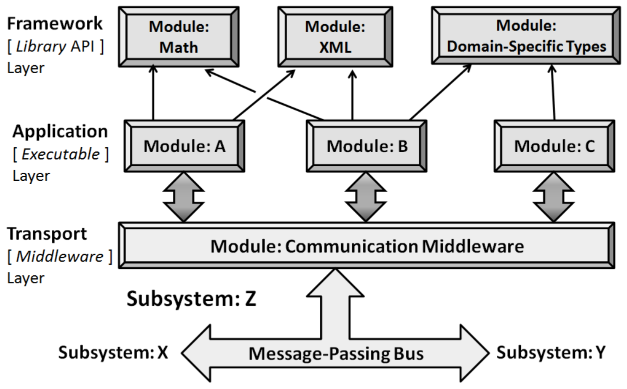


Figure 5. Subsystem architecture

The Application layer consists of modules that contain executable programs. These programs use library services and therefore the Application layer has compilation dependencies on code within the framework. Fortunately, mature framework libraries are very stable and not subject to change during the course of a major release. If however a library change is required which affects an interface, then each module in the application layer must be rebuilt and retested. But, that is why we have lots of regression tests.

The Transport layer is encapsulated in a module that provides interface protocols and mechanisms for inter-module and inter-subsystem communication. Each subsystem will contain an instance of this module. Infrastructure-type modules should, whenever possible, be purchased as a commercial off the shelf product (with support and source code) instead of developed in-house.

It is important to note that there are no direct compilation dependencies between subsystems or between modules in the same layer. The development environment should be setup to provide a mechanism to prevent direct imports between modules in the same layer because without this mechanism in place, it is only a matter of time before someone under pressure breaks this rule in order make a quick-fix. Program hacks that violate architecture rules cause an application to accumulate technical debt, which is seldom repaid by refactoring the code at a later time.

7 Modules

Continuing the analogy with hardware, subsystems contain modules. Like subsystems, modules also communicate via middleware and thereby avoid direct compilation dependencies with each other.

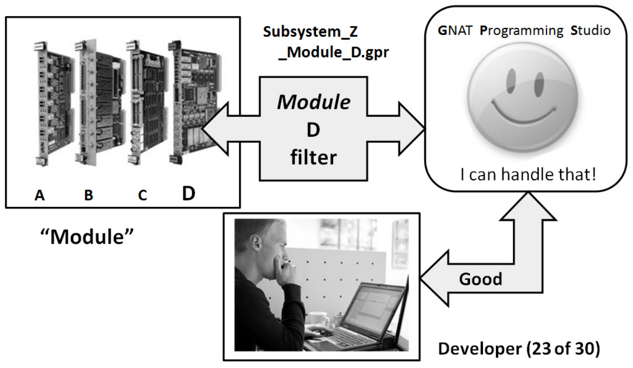


Figure 6. A module contains related components

Figure 6 shows the sources associated with the components in Module_D passed to a developer in a Workspace - the other modules in the subsystem (A, B, C) are filtered out.

The module view is useful for interface design, change-impact analysis, stress-testing, code-browsing, and build-all operations. By filtering-out all sources not related to the selected module, the demands on the workstation are greatly reduced and the developer only sees the code that matters. Source code filtering is accomplished using features of the GNAT tool chain and a scripting language to execute a simple algorithm.

In some runtime environments, application modules can be updated on-the-fly if the operating system supports the notion of a computer cluster. As for the library modules in the framework layer, these provide stable and widely-used services which under normal circumstances should not be changed by application developers.

8 Components

Each component in a module encapsulates strongly-related Ada packages that collaborate to produce the behaviour specified at the component interface.

Projects using Ada 2012 can take advantage of the contract-model to specify the semantics of an interface with such a high-degree of accuracy that static analysis may be used to validate the correctness of the associated body using mathematical logic.

Components are represented by main-programs that become executable images or class-categories that become static or dynamic library resources. In the component-view (Figure 7) the amount of code to be processed by the development environment is small enough to allow the workstation to react quickly to commands that invoke CodePeer to analyze subprograms, or AUnit to determine if a regression has occurred, GNATcheck to verify conformance to specific coding rules, GNATcoverage to expose any untested code segments, and GDB to visually trace the flow of processing to the cause of a problem. When programming, debugging, or using the static analyzer, the less the code, the better.

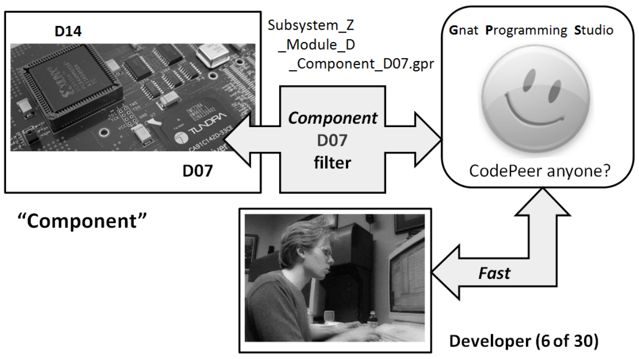


Figure 7. Programming in a component view

9 Project extension with source filters

Figure 8 illustrates a read-only, pre-built, baseline project representing the state of the last incremental release and a workspace project that extends that release, including the object files. Hence, if a developer transfers a unit from the release to the workspace, checks it out of version control, and makes a change to the body, then only that unit must be re-compiled - all other objects are found on the release drive. As a result, the project-extension strategy is very efficient in terms of computing resources because the application as seen from the workspace is already compiled; a process that can take hours on a build-server.

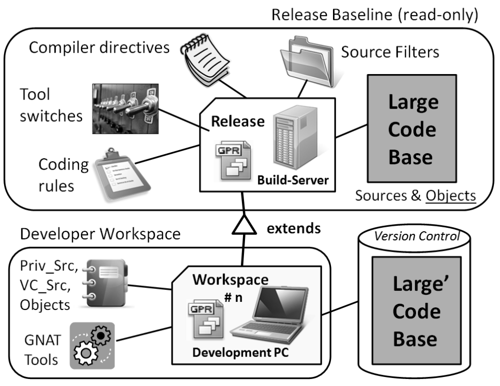


Figure 8. Project extension strategy

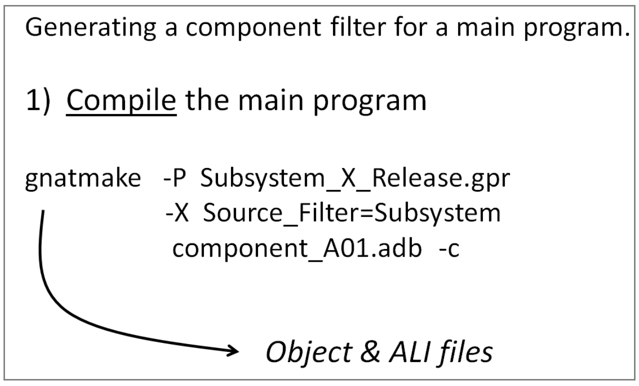
This environment is also safe and easy to manage because everyone is using the same release-project and configuration-files, all of which are read-only. That means developers can focus on design and implementation activities and need not concern themselves with complex information settings embedded in the baseline release on a network drive.

The only project file under the control of the developer is that in the workspace and this project file is very simple. Developers edit a template project file to specify only three basic attributes: Object_Dir, Exec_Dir and Source_Dirs.

10 Source filters

10.1 Executable component filter

After porting a large codebase to GNAT the only concrete things we have to represent executable components are the main programs. That being the case, how do we generate a source-filter for a main program?


Generating a component filter for a main program.

1) Compile the main program

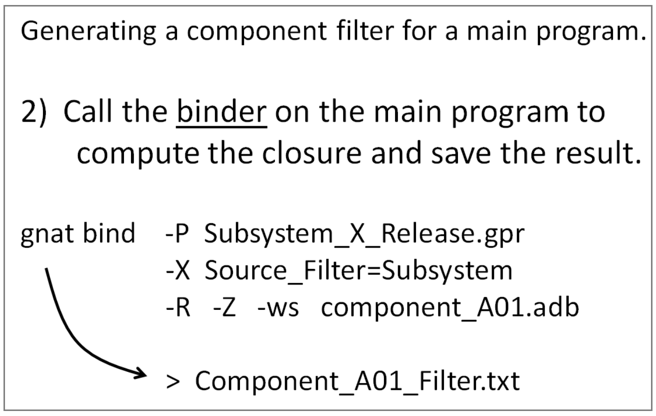
```
gnatmake  -P  Subsystem_X_Release.gpr
          -X  Source_Filter=Subsystem
           component_A01.adb  -c
```

Object & ALI files

Figure 9. Project extension strategy

The first step shown in Figure 9 is to compile the main program. We do that by passing the release-project as a subsystem and the name of the main-program to the builder (gnatmake in this case) so as to generate the object and Ada Library Information (ALI) files needed by the binder.

The 'P' switch identifies the Project file, 'X' allows us assign a value to an eXternal scenario-variable via the command line and 'c' limits processing to the compilation phase.


Generating a component filter for a main program.

2) Call the binder on the main program to compute the closure and save the result.

```
gnat bind   -P  Subsystem_X_Release.gpr
            -X  Source_Filter=Subsystem
            -R  -Z  -ws   component_A01.adb

            >  Component_A01_Filter.txt
```

Figure 10. Generating source filter for executable

The form of step 2 shown in Figure 10 is very similar to the previous step. This time however we call the binder on the main program to compute the closure and save the result in a text file that represents the filter, which is simply a list of units - one on each line. Using a consistent naming convention for files and directories greatly simplifies the implementation of the script that generates the source filters. The meaning of the switches above is as follows: R = list sources Referenced in closure, Z = Zero formatting (i.e. do not include the path to the file name), and ws = warnings are suppressed.

10.2 Module filter for executable components

To generate a module filter we apply the steps in the pseudo code of Figure 11. In this example we generate a filter for Module_A that contains four executable components: A01 through A04. The resulting file (Mod_A_Exec_Filter.txt) contains a sorted list (with no duplicates) of all units needed to build each component.

```
Generate a module filter for a set of components.

run   Generate_Component _Filter (A01 .. A04);

copy  Component_Filter (A01 .. A04).txt
        into  Mod_A_Exec_Filter.txt

organize  Mod_A_Exec_Filter.txt
            | sort-object  | get-unique  | out-file
            Mod_A_Exec_Filter.txt
            -encoding  ASCII
```

Figure 11. Producing source filter for a module

10.3 Executable component filter in project file

The location of a component filter in the Release project appears within a select-branch of the "case Source_Filter" statement as shown in Figure 12. The project file attribute "Main" identifies the main program and the attribute "Source_List_File" points to the component filter.

```
Location of a component filter in release project

 case Source_Filter is
   . . .
   when "Mod_A.Comp_A01" =>
     for  Main  use ("component_A01");

     for  Source_List_File  use  Module_A_Dir
        &  "\filter\A01_Exec\Component_A01_Filter.txt";
   . . .
end case;
```

Figure 12. Location of exec source filter in a project file

10.4 Module filter in project file

A module-filter is located in the release project (see Figure 13) in a manner similar to a component filter. However, this time the Main attribute identifies the set of main programs contained in the module.

10.5 Library component filter

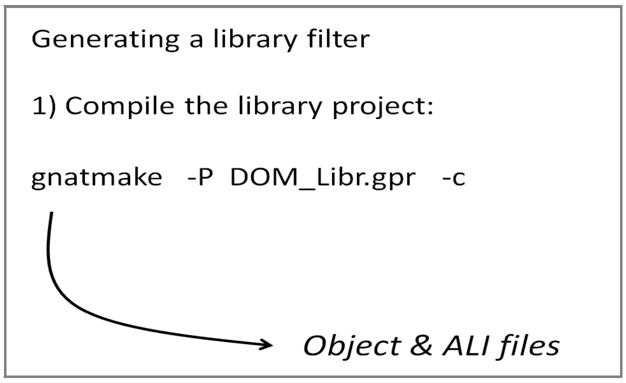
We have seen how to create source filters for executable components in the application layer. We will now discuss the creation of source filters for library components in the framework layer.

```
Location of a module filter in release project

 case Source_Filter is
   . . .
   when "Mod_A" =>
     for  Main  use ("component_A01",
                              . . .
                      "component_A04");

     for  Source_List_File  use  Module_A_Dir
        &  "\filter\A01_Exec\Mod_A_Exec_Filter.txt";
end case;
```

Figure 13. Location of exec module filter in a project file

Referring to Figure 5, we see that Module_A and Module_B invoke services in the XML library module. The list of sources in the XML module that are required by the application layer will be a small subset of the available sources that comprise the module. In other words, the XML library module filter is based on the needs of the application as determined by that portion of the API invoked by modules A and B.

Note that a framework developer responsible for maintaining a library component would load the corresponding library project and not the release project, unless the objective was to see how the library is being used from the client's perspective.


Generating a library filter

1) Compile the library project:

```
gnatmake  -P  DOM_Libr.gpr  -c
```

Figure 14. Generating required library object and ali files

As shown in Figure 14, the first step is to compile all sources in the module needed by the application. In this example the application needs services provided by the DOM component. This will generate the object and Ada Library Information (ALI) files needed by the binder.

In step 2 (Figure 15) we call the binder on the interface (the list of imported Ada specs) and send the output to a text file which represents the filter. The meaning of the (not yet encountered) GNAT switches are: n = no Ada main-program and z = zero foreign main-programs.

Generating a library filter

2) Call the binder on the interface:

```
gnat  bind  -n  -z  -P    DOM_Libr.gpr
            -R  -Z  -ws   dom-core.ali,
                          dom-core-attrs.ali,
                           . . .
                          dom-readers.ali

                    >  DOM_Libr_Filter.txt
```

Figure 15. Generate the library source filter required by modules A and B in the application layer

10.6 Library module filter in project file

A library module-filter is located in the release project as shown in Figure 16. There is no main program in the case of a library and therefore only the "Source_List_File" attribute is needed to point to the filter.

Location of a <u>library filter</u> in release project

```
 case Source_Filter is
   . . .
   when "Mod_XML.Comp_DOM_Libr" =>
     for  Source_List_File  use  Module_XML_Dir
          &  "\filter\ DOM_Libr\DOM_Libr_Filter.txt ";

end case;
```

Figure 16. Location of libr module filter in a project file

10.7 Impact analysis using source filters

The information in source filters can be used to determine the impact of a change to a unit. As shown in Figure 17, if we make a change to the body of Unit_X from Main_1 and the same unit also appears in the closure of Main_2, then Main_2 must be retested. As the number of main programs impacted by a change to a given unit increases then obviously so does the time, cost, and risk.

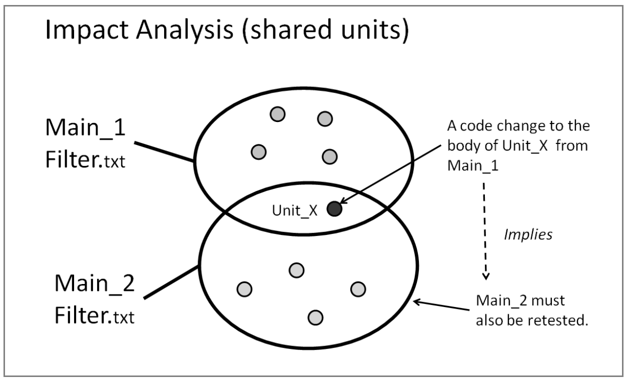

Impact Analysis (shared units)
Main_1 Filter.txt
Unit_X
A code change to the body of Unit_X from Main_1
Implies
Main_2 Filter.txt
Main_2 must also be retested.

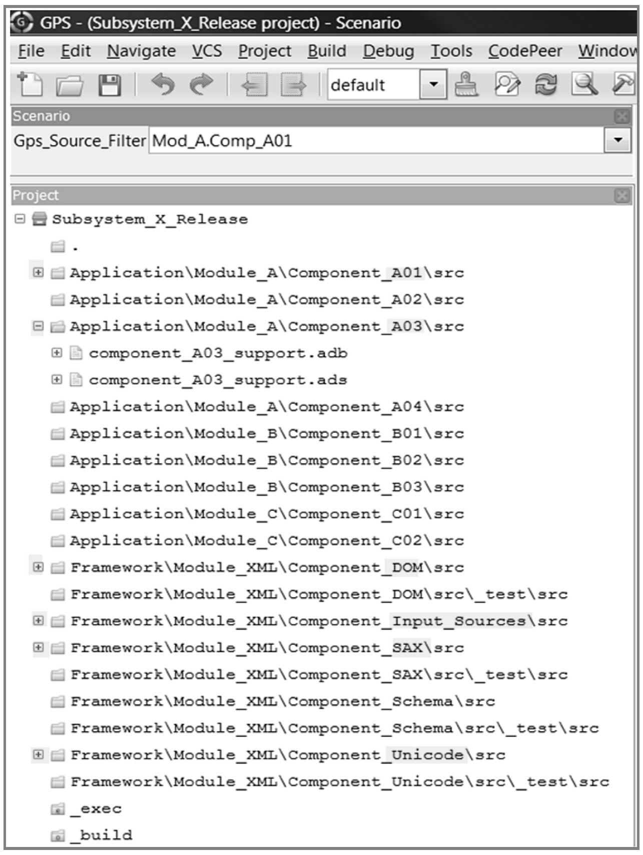
Figure 17. Unit(s) shared between main programs

If several units are shared between two or more processes (i.e. main programs) then it might make sense to factor these units out into a library located in the framework. A decision such as this must take into account various considerations such as the potential for reuse, robustness, and the unlikelihood of future code changes.

Since source filters are simply text files containing unit names, it is straightforward to data-mine these files with a script to extract change-impact and other useful results.

10.8 The effect of source filters in GPS

Once GPS is up and running, the developer selects from a dropdown list, the subsystem, a module, or a component in a module. Figure 18 shows the effect of loading Component_A01 in Module_A.


GPS - (Subsystem_X_Release project) - Scenario
File Edit Navigate VCS Project Build Debug Tools CodePeer Window
default
Scenario
Project

Gps_Source_Filter Mod_A.Comp_A01

```
⊟ Subsystem_X_Release
    .
  ⊞ Application\Module_A\Component_A01\src
    Application\Module_A\Component_A02\src
  ⊟ Application\Module_A\Component_A03\src
    ⊞ component_A03_support.adb
    ⊞ component_A03_support.ads
    Application\Module_A\Component_A04\src
    Application\Module_B\Component_B01\src
    Application\Module_B\Component_B02\src
    Application\Module_B\Component_B03\src
    Application\Module_C\Component_C01\src
    Application\Module_C\Component_C02\src
  ⊞ Framework\Module_XML\Component_DOM\src
    Framework\Module_XML\Component_DOM\src\_test\src
  ⊞ Framework\Module_XML\Component_Input_Sources\src
  ⊞ Framework\Module_XML\Component_SAX\src
    Framework\Module_XML\Component_SAX\src\_test\src
    Framework\Module_XML\Component_Schema\src
    Framework\Module_XML\Component_Schema\src\_test\src
  ⊞ Framework\Module_XML\Component_Unicode\src
    Framework\Module_XML\Component_Unicode\src\_test\src
    _exec
    _build
```

Figure 18. A source filter loaded into GPS

The project panel provides some useful information. For example, directories that contain source code are indicated by the presence of a plus or minus symbol. Component_A01 has no dependency on A02 and A04. Module_A is isolated from Modules B and C, and the Schema component in the XML module is not touched. Hence, both the presence and the absence of code serve to validate visibility rules dictated by the architecture.

11 Production build

Up to now we have described the development environment and in particular the form of a release project used by software engineers. However, at some point we have to produce another kind of release that can be deployed in a pre-production environment for use by professional testers and domain experts to verify and validate the state of the application. For that purpose we need an optimized set of project files designed to satisfy space and performance requirements of the application under real-world conditions.

The production build for a subsystem is a two-stage process. In stage 1 we produce the static and dynamic libraries for the framework layer. In stage 2 we produce the executable images for the application layer.

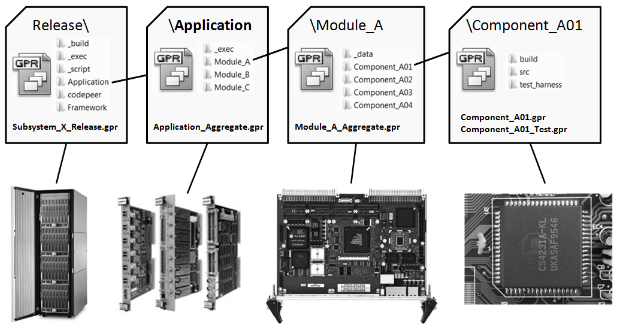


Figure 19. Project file hierarchy for a production build of the application layer.

In the example illustrated in Figure 19, the application layer of the subsystem consists of three modules, where module_A contains four components. For visualization purposes, it is helpful to use the hardware analogy where a subsystem is an electronic cabinet or a rack within a cabinet. A module is a VME-type circuit board that plugs into a slot in the cabinet. And a component is an electronic chip mounted on a circuit board.

The purpose of each component-level GPF in the application layer is to create an executable image for use in production. For example, since the visual debugger is never used on the production system, the executable images will not contain debug information. Also, most assertions are turned off and the code is optimized for performance.

A package in a GPF can be defined by a renaming-declaration to obtain attributes specified in an external GPF (e.g. Shared_Production_Attributes.gpr). This is standard practice for the Compiler and Builder packages in each component-level project file so as to maintain switch and configuration-pragma information in one place, thus ensuring a consistent build environment.

Modules are a construct for encapsulating related components and therefore, a module project file is simply an aggregate of its components. Similarly, the application layer is an aggregate of module aggregates.

Aggregate projects, which are an extension of the standard project paradigm, simplify the building of architectures that use modules and components as building blocks as illustrated in Figure 20. The simplification occurs in that we only need one command to build an entire application or framework layer. In addition, aggregate projects allow for a very-efficient build due to the fact that the builder process (e.g. gprbuild) has visibility to the sources of each component. Having this overview means duplicate work is avoided and parallel processing techniques can be used to exploit all available CPU cores.

```
aggregate project Module_A_Aggregate is
  for Project_Files use ("Component_A01 \ Component_A01.gpr",
                         "Component_A02 \ Component_A02.gpr",
                         "Component_A03 \ Component_A03.gpr",
                         "Component_A04 \ Component_A04.gpr");
end Module_A_Aggregate;
```

```
aggregate project Application_Aggregate is
  for Project_Files use ("Module_A \ Module_A_Aggregate.gpr",
                         "Module_B \ Module_B_Aggregate.gpr",
                         "Module_C \ Module_C_Aggregate.gpr");
end Application_Aggregate;
```

Figure 20. Aggregate projects used in the production build environment

The issue of build efficiency is particularly important in large projects involving millions of lines of code. For example, if the nightly-build process fails, then after someone has fixed the problem, the job must be restarted to run during working hours, and depending on circumstances, this delay can be very expensive. Therefore, it pays to have the most efficient build environment possible, and for that purpose the use of aggregate projects is recommend.

12 Conclusion

The most pragmatic and expedient way to apply GNAT technologies to a new codebase is by extending a pre-built release in a workspace and using source filters to manage complexity. This is called the code-view because initially that is all we have - the source code.

The code-view allows you to use a single GPS instance to quickly move between different modules and components within a given subsystem. Note however that the code-view configuration requires some programming effort to create the scripts for generating and applying source filters.

If the ported application is the result of conscious design then it will have an identifiable architecture, which should be brought to the surface where it is visible to all stakeholders. To accomplish this objective, project files representing key components are systematically add to each incremental release until the domain model for each subsystem is complete. Depending on the size of the application and available resources, it might require several months to establish this view.

The architecture-view is useful when development is, for example, focused on the development and testing of an algorithm in a specific component over the course of a few days. In that case, you would load the root-level project representing the component into GPS and get to work. Unlike the code-view, we are not obligated to load all the sources associated with the enclosing main program.

Electronic engineers have been applying the concepts of subsystems, modules, components, and interfaces for decades to build highly-reliable systems with great success. Hence, it only makes sense for software engineers to apply the same principals to the construction of software systems. This is easier said than done, but the design of Ada and the GNAT development environment contribute greatly to this objective.

References

[1] GNAT Pro User's Guide (2011/11/03), *The GNAT Pro Ada Compiler*, Version 7.0.1, Document revision level 180256 AdaCore.

[2] GNAT Pro Reference Manual (2012/01/03), *The GNAT Pro Ada Compiler*, Version 7.0.1, Document revision level 181986 AdaCore.

[3] GPS Documentation (2013/01/02), *Release 5.2.1* AdaCore.

[4] GPRbuild User's Guide (2012/03/28), Document revision level 187710 AdaCore.

The 16th International Real-Time Ada Workshop

Alan Burns

Department of Computer Science, University of York, UK

Abstract

The 16th occurrence of this successful workshop series took place in York, UK from 17th to 19th of April in 2013. The venue for the workshop was the medieval King's Manor situated in the centre of historical York. The workshop was sponsored by Ada-Europe, AdaCore and the University of York, and was organised by the programme committee consisting of Mario Aldea Rivas, Alan Burns, Michael González Harbour, José Javier Gutiérrez, Stephen Michell, Brad Moore, Luís Miguel Pinho, Juan Antonio de la Puente, Jorge Real, Jose F. Ruiz, Joyce Tokar, Tullio Vardanega, Andy Wellings and Rod White. In all twenty people attended the event as listed below.

Workshop Participants

- Mario Aldea Rivas, University of Cantabria, Spain.
- Geert Bosch, AdaCore, USA.
- Alan Burns, University of York, UK.
- Robert Dewar, AdaCore, USA.
- Michael González Harbour, University of Cantabria, Spain.
- Kristoffer Nyborg Gregersten, Norwegian Institute of Science and Technology (NIST), Norway.
- Stephen Michell, Maurya Software Inc., Canada.
- Brad Moore, General Dynamics, Canada.
- Luís Miguel Pinho, Polytechnic Institute of Porto, Portugal.
- Juan A. de la Puente, Technical University of Madrid, Spain.
- Jorge Real, Universitat Politècnica de València, Spain.
- José Ruiz, AdaCore, France.
- Sergio Sáez, Universitat Politècnica de València, Spain.
- Amund Skavhuag, NIST, Norway.
- Joyce Tokar, Pyrrhus Software, USA.
- Tullio Vardanega, University of Padua, Italy.
- Simon Vincent, MBDA UK Ltd, UK.
- Andy Wellings, University of York, UK.
- Rod White, MBDA UK Ltd, UK.
- Juan Zamorano, Technical University of Madrid, Spain.

1 Introduction

A call for papers and subsequent review process lead to 11 papers being accepted for the workshop. Final versions of these papers will appear in Ada Letters in due course. The structure for the workshop followed the usual pattern for this series of meetings. Participants received and read the pre-workshop versions of the papers before the workshop began. For the workshop itself, a set of topics (derived from the accepted papers, and to some extent topics addressed at previous IRTAWs) were identified and formed the basis for detailed round-the-table discussions facilitated by a session chair and summarised by a session rapporteur.

The main topics for the workshop were:

- Language features that will allow parallel hardware to be exploited;
- Protocols to support shared resources in both single processor EDF scheduling systems and fixed priority multi-processor systems;
- Language improvements and potential future for Ada; and
- Profiles that go beyond Ravenscar.

Each session gave rise to useful discussions and the identification of potential future directions for research and the development of the Ada programming language. Summaries of each of the sessions are now provided below.

2 Parallel Ada

This was the longest session at the workshop. It addressed the important issue of how to support and exploit highly parallel hardware. The majority of the session's discussions were based on the collaborative work of Michell, Moore and Pinho. The motivation for a parallel solution in Ada is in response to changes in computer chip architectures currently available, as well as future directions. The first important change noted is that Moore's law no longer applies. We can no longer rely on faster CPU clock speeds to absorb increasing complexity and demands of computer applications. Another related factor has to do with how chip manufacturers are responding to practical limits in CPU clock speed, by increasing the number of cores on a single computer chip.

The term *Parallelism OPportunity* (*POP*) was introduced to represent the locations in the program code that are suitable for parallel execution. The goal for the general model is to allow for POPs to be explicitly identified in the programmer's code. To illustrate the use and need for POPs, the example of a parallel loop was used, as loops are very prevalent in application code, and are amenable to a divide and conquer approach.

Discussion focused on the wisdom of giving any directive further than *with parallel* for the program to control the details of how parallelism is configured, executed and potentially mapped to cores at runtime. Programmers may not provide the correct specification of detailed controls, and as hardware changes over time, some argued that it is better to let the compiler have the control on these inputs. The counter argument was raised that in real-time systems there is a need for the programmer to specify such control to directly specify the behaviour, which is required for behaviour analysis and timing analysis. In other cases, the default performance parameters may be suboptimal for a particular problem, and the programmer may need to squeeze out extra performance by tweaking the controls. This could be the case in particular when code is being written for a very specific target hardware platform.

Questions were raised about the memory model of the proposal under discussion. The general model is that it supports a shared memory system, with cache coherency, with uniform access to memory, within a single partition. At the same time the desire was not to restrict the model if at all possible. Underlying memory buses and memory organization, however, mean that there can be orders of magnitude difference in accessing any particular memory location from various CPUs, and issues such as cached memory and cache flushes can cause wildly varying access times, and possibly inconsistent views of shared data. It was emphasized that the view of a partition as a shared memory model is ingrained in Ada.

There was significant discussion about needing a definition for the unit of parallelism, and to define the semantics of a *Tasklette,* and indeed whether *Tasklette* is even an appropriate name for the concept. Alternate names suggested were *Strand*, *Fibre* and even *Lemming.* The difficulty that participants had with *Tasklette* was that name is very close to *Task,* which seems to imply that one should be able to have attributes, execution time accounting, and blocking on such creations, which was antithetic to what participants wanted.

A subtopic of the discussion of Tasklettes, was what happens to exceptions that are raised inside of Tasklettes. Since Tasklettes simply represent a parallel execution within a parent task, the exception must be delivered back to parent at the point of synchronization. If multiple exceptions are raised by Tasklettes, all but one exception are discarded. Following Ada's exception semantics, it is irrelevant what Tasklette instance captured the exception, because you cannot rely upon any state that was being changed when an exception occurred.

A discussion was held that there is a model of Ada partitions as units of concurrency, which could possibly be extended to units of parallelism, however, the current restrictions on partitions make using partitions in this way less efficient. It was agreed that the remote procedure call mechanisms are heavy-weight for communicating between Tasklettes, and the shared passive partition model prevents the usual communication models between partners. A number of solutions were proposed and discussed, but no consensus was reached in this session.

At the end of the discussions the workshop concluded that efforts should continue to try and define means by which a future version of Ada could effectively exploit parallel hardware.

3 Resource Locking Protocols

This session considered two main issues: the introduction of the deadline floor locking protocol and multiprocessor locking policies. Non-locking protocols were also discussed.

Ada 2005 introduced EDF scheduling across priority bands with a version of Baker's Stack Resource Control Protocol so that ceiling priorities for protected objects could be used within an EDF context. However, this protocol is complex and the position paper by Aldea, Burns, Gutierrez and Gonzalez Harbour entitled *Incorporating the Deadline Floor Protocol in Ada* has proposed an alternative protocol that is conceptually much simpler and easier to implement. The protocol is targeted at single processor system and the discussion was held within this context. The protocol requires each protected object to have a relative deadline associated with it. This deadline is the minimum (floor) relative deadline of all the tasks that use that protected object. Proper setting of the floors ensures that each task gets only a single block and mutual exclusion is guaranteed by the protocol itself. This is achieved by reducing the absolute deadline of a task (d) when it enters a protected object (PO) at time t to the value t+F, where F is the deadline floor of the PO. This temporary change only happens if d is initially greater then t+F. After questions of clarification, a number of further issues were identified and dealt with; there included coping with release jitter, POs shared between EDF and FIFO_Within_Priority scheduling, other inheritance points in Ada. Following these discussions, the workshop agreed that the deadline floor protocol would be a useful addition to Ada and that the current protocol should be made obsolete. This could be achieved with a new dispatching policy and/or a new locking policy.

The issue of how to integrate appropriate policies for accessing protected objects in multiprocessor system (into the Ada language) is still largely unresolved. The Ada reference manual suggests that tasks busy-wait for a lock but does not specify any priority or queuing policy associated with this. There were two papers submitted to the workshop on this topic. One considered a new lock-based approach (*Locking Policies for Multiprocessor Ada* by Burns and Wellings). The other considered a lock free approach (*Lock-Free Protected Types for Real-Time Ada* by Bosch). The workshop discussed both approaches but felt they were both not yet mature enough to warrant suggested language changes at this time. Much of the discussion on the lock-free approach focused on the restrictions that had to be placed on the application code so that updates to the protected data could be achieved by a single machine instruction.

The workshop felt the approach was promising but wanted to see more detailed definitions of the restrictions (and how they would be checked) and whether other forms of lock-free approaches and algorithms were possible.

4 Language Improvements

This session considered how the current Ada Language could be improved. Three papers were discussed in this session:

I. *Programming Simple Reactive Systems in Ada: Premature Program Termination*, A.J. Wellings, A. Burns, A.L.C. Cavalcanti and N.K. Singh.

II. *Execution time timers for interrupt handling*, Kristoffer Nyborg Gregertsen and Amund Skavhaug.

III. *Deferred Setting of Scheduling Attributes for Periodic and Sporadic Tasks*, Sergio Sáez, Jorge Real and Alfons Crespo.

The first paper covered the use of Ada to develop simple reactive, deterministic automata, and the issues of termination of non-tasking programs. The paper identifies two main issues:

- Queuing of interrupts and the difficulty of determining the ordering of multiple events, and more fundamentally
- Program termination – the issue that prevents the simple reactive model from working.

The proposal to the workshop was that the termination semantics for Ada should be changed. Whilst the termination in the presence of attached interrupts was not seen as a major issue there was a general consensus that termination in the presence of active timing events was incorrect – as these had been programmed, and if they were not needed then they should be explicitly cancelled by the application. It was noted in the paper that if the termination semantics are changed as suggested it will break backwards compatibility as it is currently possible for programs to terminate with timers and attached interrupts. However, the change introduced in Ada 95 to handle interrupts via protected objects rather than tasks also introduced a compatibility issue. The workshop concluded that this was not a pressing issue given the simple work-around that exist and that there was little merit in making language changes in this area.

The second paper considered execution timer timers for interrupts. Ada 2012 introduced execution time clocks for interrupt handlers – the proposal made in the paper was that Ada should be extended to provide execution time timers for interrupt handlers. Identified issues with interrupts include:

- Hard to predict their rate of arrival;
- Hardware faults can result in bursts;
- In Ada 2012 it is only possible to measure the execution time of interrupt handlers (using the Clocks defined in Ada.Execution_Time.Interrupts);
- Interrupt timers can be efficient with respect to the alternative of polling the time to determine when it has been exceeded;
- There is also a related issue with timing events where the facilities are even more limited; here, unlike interrupts, it is neither possible to measure the execution time, nor to set an execution time timer.

In general it was felt that interrupt handler code should be straightforward and serial, and hence of limited and bounded duration, this in turn led to the concern that there might be significant overheads due to the facility that might detract from this position. This led to the question: are we really only interested in the total interrupt count and rate of arrival rather than the CPU time consumed? It was noted that there is probably more of an interest in providing timers for timing events as these are firmly in the application domain, the one where timers are more widely considered to be useful.

A number of issues were noted that had to be worked on to give a more coherent solution to these problems.

- The way in which the deferrable server would work was not entirely clear and a more complete description was required;
- The type model needs to be reworked to make the types for timers in general coherent;
- The model should be extended to also include timers for timing events;
- It is important that any implementation can ensure that its support for this feature results in zero overhead for any application that does not make use of the feature.

Given these issues are adequately addressed, interrupt timers could be a feature for inclusion in a future revision of the language.

The final paper for this session was concerned with the setting of task attributes. Over the past two IRTAWs the issue of setting multiple scheduling attributes simultaneously has been noted as a topic of some interest and importance. This paper is a follow-on from the previous IRTAW where the issue of setting the various attributes of a task atomically had been considered – the current model in Ada 2012 allows only for the setting of a single attribute at a time (except for period *and* deadline). In outline, the paper proposes a new type to capture a set of scheduling attributes, an instance of which is associated with each individual task, which can be passed to the underlying kernel in a single call, hence facilitating their simultaneous, atomic setting.

The proposal includes two basic options with respect to setting the attributes of a task: setting them immediately, and setting them and suspending for them to apply at the next release. In both cases, issues were raised regarding exactly how these might work. In the first case, there was the point that setting could not be immediate if the caller was in a protected operation – the application would have to be deferred until after the protected operation had been completed. In the second case, where the task becomes suspended, a number of significant points were raised. Given the complexity, an alternative approach was tentatively suggested. Why not replace the suspension by a timing event that sets the attribute in its protected operation? The fact that it is a PO will ensure atomicity of the attribute change, but it was noted that this is not necessarily the case where the affinity is changed. From this there was some discussion as to whether affinity is particularly difficult and should be treated as a special case – no specific conclusion emerged from this discussion.

5 Language Profiles

Most of the session was focused on discussing the opportunity to define a new Ada profile by adding execution-time control mechanisms to the Ravenscar profile.

The main motivation for such a profile is to overcome the limitations of the Ravenscar profile with respect to real-time fault tolerance. The features that could be included in the new profile are: execution-time timers, group budgets, asynchronous task control, dynamic priorities, asynchronous transfer of control and the abort statement.

Execution-time timers and group budgets are proposed as run-time mechanisms for detecting overruns. Asynchronous task control and dynamic priorities can be used to lower the priority of a faulty task, thus reducing its impact on the system, and asynchronous transfer of control and abort can provide further support for this purpose.

There was a lively discussion on the proposal. A basic consideration is the wish to keep the run-time system efficient and small, in order to facilitate certification when required. Robert Dewar made a point that adding a profile would not be too complex for compiler builders, but adding new restrictions might be. There was general agreement that abort and ATC are the most complex features to implement, whereas the rest would not pose so much of a problem.

Another topic is the possible uses of the extended profile. The Ravenscar profile forces a static environment that enables schedulability analysis to be carried out in critical systems, and was originally conceived as a replacement for cyclic executives that were dominant at the time. On the other hand, an extended profile may add flexibility for other possible uses. Geert Bosch commented that Ravenscar is too limited for some users, while Rod White observed that some non-critical applications use the Ravenscar runtime because it is small and simple. Amund Skavhaug stressed the interest of the extended profile in education, where it could be used in small student projects. There was however consensus that asynchronous task control is a complex issue that can be difficult to implement in a reduced runtime system.

Conclusions

The workshop concluded by summarising its achievements and recommendations, and by reiterating topics worthy of future study. It was agreed that a further workshop in approximately 18 months time would be worthwhile. Possible venues for IRTAW17 were identified and responsibilities were accepted to bring about the next workshop.

How to Use the Heap in Real-Time Systems: Panel Report

Chair: Erhard Plödereder

Rapporteur: Jørgen Bundgaard

Abstract

In the Ada-Europe 2013 conference, a panel was dedicated to the subject of how to use the heap in real-time systems.

This document provides a report of the presentations and discussion of the session.

Keywords: Heap, Real-Time Systems

1 Introduction

The session started with three presentations from renowned specialists, presenting different viewpoints, and a final wrap-up. The panellists were

- Ludovic Gauthier, from Atego Systems, Inc., USA, presenting how to extend the Java type system to enforce disciplined use of scope-allocated objects;
- S. Tucker Taft, from AdaCore, USA, presenting region-based storage management for parallel programming; and
- James Hunt, from aicas GmbH, Germany, presenting dynamic memory management in real-time, safety-critical systems.

After each presentation the floor was open to questions and comments from the audience.

The panel moderator was Erhard Plödereder, from the University of Stuttgart, Germany, who started by introducing the objectives and participants of the panel, and by encouraging the audience to ask aggressive and provocative questions.

This paper reports a brief summary of the presentations and of the questions (and answers) that followed.

2 Extending the Java Type System

The first presentation started by discussing why developers use heap memory technologies, contrasting with the needs of safety-critical software. This leads to important considerations for safety-critical heaps such as clearly understanding (and statically specifying) the memory requirements of the application, executing without run-time errors, or being able to verify memory behaviour on the modular composition of software components.

Ludovic Gauthier presented the stack-of-scopes execution model, where the stack scope of nested threads is created from the parent's stack, and nested threads can always access the outer scope's objects.

Afterwards, the presentation focused on the PERC Pico (Virtual Machine for Java real-time and safety critical systems) approach to safety-critical memory management, which enhances the Java type system to represent relevant memory management details. The traditional Java mindset is "not to be concerned" about memory, while the new approach adds resource constraint annotation (annotations to be inserted in the code), allowing programmers to "assert" bounds on loop iterations and recursion depths. A key point is to reduce the heap issue to a stack allocation problem.

The first comments from the floor were related to the problems which originate from the usage of heaps, and the burden on the programmer to need to annotate the code, compared to using languages which reduce such problems. Ludovic Gauthier answered that, by adding restrictions, we can still use the flexibility of the heap, and that the compiler can help considerably in the burden of analysing. Answering to doubts on the use of Java in safety-critical systems, he noted that Ada also enforces restrictions on the use of dynamic memory in safety-critical applications.

The audience also asked for experience with industrial projects, and in particular for reuse (one of the main Java's intended advantages). The answer was that there were experiences, mostly from the avionics domain, and that reuse had been required in one particular case. The work was significant, but achievable.

3 Region-based storage management

The second presentation dealt with the use of region-based storage management in multicore programming models. The presentation started by providing some insights in the challenges of using automatic storage management in parallel programs, which is something that programmers "love" to have. The presentation put forward that global garbage collected heap is bad in the context of parallel languages, presenting as alternatives the Rust memory model and region-based storage management.

Afterwards, Tucker Taft explained the latter, which is similar to stack-based memory management, presenting the concept of stack of regions with region chunks, global vs. region-based storage management (differences concerning locking, object locality, and object separation). Eliminating pointers simplifies the region-based storage management approach, without the need to worry about annotations.

The presentation ended with an example of pointer-free (binary) trees, and some of the ParaSail virtual machine statistics.

A few questions arose on how the approach deals with fragmentation, and explicit freeing. The answer was that by reclaiming storage before leaving the region, the fragmentation problem was bounded; if an object is set to null, the storage is reclaimed immediately. Answering to another question on how it is decided which region to use, Tucker Taft noted that this was decided depending on the scope of the object.

A question was asked on dimensioning the region and how to set the correct size. The answer was that regions are of fixed size. If exhausted, a new chunk is allocated, with each processor having its pool of region chunks. These operations are bounded (in time).

4 Dynamic memory management

The final presentation of the session started by providing some sample applications where dynamic memory management is necessary, such as path retracing or object recognition, which are now increasingly considered in critical domains. James Hunt then described the main dynamic memory (DM) vulnerabilities, and the DM safety objectives. Afterwards he discussed how different memory management techniques achieve these objectives, considering in particular deterministic garbage collection, since it allows reducing development time and improves safety.

Afterwards he focused on real-time garbage collection, describing some of the existent techniques: paced and slack garbage collection. In both he noted that the programmer must provide both maximum memory use and maximum allocation rate. He then presented the work-based garbage collector approach. One interesting note was that to allow for real-time garbage collector, it was not only necessary to use a deterministic approach, but also to use the real-time programming model, which the Real-Time Specification for Java has taken from Ada.

A concluding remark was that generating code from a state machine is best, if possible. For complex safety critical programs, deterministic garbage collection may work. But not all Java implementations are suitable.

The first question was if a specific Java compiler was being used, to which James Hunt replied positively. Afterwards, a question was raised if it is safe to use garbage collection for safety critical applications. What about the interaction between the application and the garbage collectors and certification? James Hunt replied that this garbage collector approach had already been used in avionics systems and qualification guidance is now in place.

Another comment was that the memory management infrastructure cannot do timely de-allocation unless the user sets the pointer to null? Is a memory leak inevitable? To this James Hunt replied that this was true for variables that do not go out of scope; the programmer still needs to worry about the application being written. He also noted that there was a region approach (the JamaicaVM garbage collector uses fixed size blocks for allocation to ensure very low latency) being used inside the garbage collector.

Another doubt was about the bounded behaviour of the collector. Worst case execution time is known for allocation and deallocations. Although complex, it is possible to analyze it.

Answering to a question on overhead, James replied that 20% extra memory is needed, maybe 10% for less demanding applications.

3 Wrap-up

Towards the end of the session, questions were posed to all panellists, who also had the opportunity to voice a final comment.

The first general comment was on the use of heap at all. A member of the audience noted that in the automotive domain, there are 4 wheels, 8 pistons, etc. Tucker noted nevertheless that autonomous cars must take a very dynamic environment into account, everything cannot be allocated statically. In other domains objects have very different sizes. Allocating objects of the exact same size will not be efficient, and will not be maintainable.

This was followed by the question whether heaps can indeed be used in safety critical applications. When can we be confident that they can be?

James Hunt replied that deterministic garbage collection will be proven in use, and then it will spread. Most safety critical applications are currently state machines, so do not require this change, but in future this will need to change (due to increased complexity). Tucker noted that it was not safe to use today but it will become safer, and eventually bodies such as FAA may consider it. Ludovic opined that nevertheless, for really critical applications, unrestrained dynamic memory allocation will probably never be allowed.

Another note from the audience was that even in complex systems the environment can be represented in 2 or 3 dimensions, and that is static. Tucker did not agree, noting the answer to a previous similar question. James added that loop/recursion behaviour is the biggest problem.

Finally, there was a question on how far it is possible to go with pre-conditions (non-null pointers). James answered that with pre-conditions a lot of the complexity of the garbage collector can be reduced by applying relative simple analysis in the compiler.

Afterwards each of the panellists summarized their view on the topic.

James Hunt considered that there are regulatory frameworks in place for avionics; we will get to a point where it will be accepted to use dynamic memory allocation.

Tucker Taft considered that it is not realistic to do everything with static allocation. Automation is necessary for safety critical applications, manual techniques do not scale. New languages, or annotations, will come. It will become part of our world.

Ludovic Gauthier considered that it will be possible to reduce Java to a subset where it will be acceptable for use in safety critical applications. Dynamics will be more and more necessary to manage systems.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/adaanswers/gems.

Gem #150: Out and initialized

Emmanuel Briot, AdaCore
Robert Dewar, AdaCore

Abstract. This Gem describes some perhaps unexpected cases where variables aren't necessarily updated following assignments, though it might not be obvious from the code.

Let's get started…

Perhaps surprisingly, the Ada standard indicates cases where objects passed to out and in out parameters might not be updated when a procedure terminates due to an exception. Let's take an example:

```
with Ada.Text_IO;  use Ada.Text_IO;
procedure Gem is

  procedure Local (A : in out Integer; Error : Boolean) is
  begin
    A := 1;

    if Error then
      raise Program_Error;
    end if;
  end Local;

  B : Integer := 0;

begin
  Local (B, Error => True);
exception
  when Program_Error =>
    Put_Line
       ("Value for B is" & Integer'Image (B));  -- "0"
end Gem;
```

This program outputs a value of 0 for B, whereas the code indicates that A is assigned before raising the exception, and so the reader might expect B to also be updated.

The catch, though, is that a compiler must by default pass objects of elementary types (scalars and access types) by copy and might choose to do so for other types (records, for example), including when passing out and in out parameters. So what happens is that while the formal parameter A is properly initialized, the exception is raised before the new value of A has been copied back into B (the copy will only happen on a normal return).

In general, any code that reads the actual object passed to an out or in out parameter after an exception is suspect and should be avoided. GNAT has useful warnings here, so that if we simplify the above code to:

```
with Ada.Text_IO;  use Ada.Text_IO;
procedure Gem2 is

   procedure Local (A : in out Integer) is
   begin
     A := 1;
     raise Program_Error;
   end Local;

  B : Integer := 0;

begin
  Local (B);
exception
  when others =>
    Put_Line ("Value for B is" & Integer'Image (B));
end Gem2;
```

We now get a compilation warning:

```
gem.adb:6:10: warning: assignment to pass-by-copy
formal may have no effect

gem.adb:6:10: warning: "raise" statement may result in
abnormal return (RM 6.4.1(17))
```

Of course, GNAT is not able to point out all such errors (see first example above), which in general would require full flow analysis.

The behavior is different when using parameter types that the standard mandates passing by reference, such as tagged types for instance. So the following code will work as expected, updating the actual parameter despite the exception:

```
procedure Gem3 is

  type Rec is tagged record
    Field : Integer;
  end record;

  procedure Local (A : in out Rec) is
  begin
    A.Field := 1;
    raise Program_Error;
  end Local;

  V : Rec;

begin
  V.Field := 0;
  Local (V);
exception
  when others => Put_Line
       ("Value of Field is" & V.Field'Img);     -- "1"
end Gem3;
```

It's worth mentioning that GNAT provides a pragma called Export_Procedure that forces reference semantics on out parameters. Use of this pragma would ensure updates of the actual parameter prior to abnormal completion of the procedure. However, this pragma only applies to library-level procedures, so the examples above have to be rewritten to avoid the use of a nested procedure, and really this pragma is intended mainly for use in interfacing with foreign code. The code below shows an example that ensures that B is set to 1 after the call to Local:

```
   package Gem4_Support is

     procedure Local (A : in out Integer; Error : Boolean);
     pragma Export_Procedure (Local,
                   Mechanism => (A => Reference));
   end Gem4_Support;

   package body Gem4_Support is

      procedure Local (A : in out Integer; Error : Boolean) is
      begin
         A := 1;
         if Error then
            raise Program_Error;
         end if;
      end Local;
end Gem4_Support;
```

```
with Ada.Text_IO;  use Ada.Text_IO;
with Gem4_Support; use Gem4_Support;
procedure Gem4 is
   B : Integer := 0;
begin
   Local (B, Error => True);
exception
   when Program_Error =>
      Put_Line ("Value for B is" & Integer'Image (B)); -- "1"
end Gem4;
```

In the case of direct assignments to global variables, the behavior in the presence of exceptions is somewhat different. For predefined exceptions, most notably Constraint_Error, the optimization permissions allow some flexibility in whether a global variable is or is not updated when an exception occurs (see Ada RM 11.6). For instance, the following code makes an incorrect assumption:

```
X := 0;     -- about to try addition
Y := Y + 1; -- see if addition raises exception
X := 1      -- addition succeeded
```

A program is not justified in assuming that X = 0 if the addition raises an exception (assuming X is a global here). So any such assumptions in a program are incorrect code which should be fixed.

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

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada in Sweden

Ada-Sweden
attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada Switzerland

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch