ADA USER JOURNAL

Volume 34
Number 4
December 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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

In this close of the year I would like to draw the attention of the reader to the announcement that the Rationale for Ada 2012 is being published, both online, freely downloadable, but also as a book as usual in the Springer Lecture Notes in Computer Science series, Volume 8338, to be released in mid-December 2013. We are much honored to have had the opportunity to publish the individual chapters of the Rationale in the Ada User Journal, even more because it was really enjoyable to be among the first to read the usual highly educative and insightful writing of John Barnes.

I would also like to refer the reader to the letter from the President of Ada-Europe (that the Journal publishes in the next page), calling for 2014 sponsors for the Ada-Europe organization. This sponsorship is fundamental to maintain the activities Ada-Europe is charted (and committed) to perform. Even if you personally may not be able to be a sponsor, please forward this information to potential sponsors in your network. You can find an electronic version of the letter in the Ada-Europe website at www.ada-europe.org.

The issue continues with the usual News Digest, Calendar and Forthcoming Events sections. The latter provides information on three events which will take place in 2014: the Ada Developer Room at the Free and Open source Software Developers' European Meeting (FOSDEM) next February in Brussels, Belgium; the call for papers and industrial presentations for the 19th International Conference on Reliable Software Technologies – Ada-Europe 2014, to take place June 2014 in the heart of Paris, France; and the ACM SIGAda's High Integrity Language Technology, to take place (tentatively) October 2014 in Portland, Oregon, USA.

The technical part of the Journal continues the publication of articles from the Industrial Track of Ada-Europe 2013; in this issue with a contribution from Jacob Sparre Andersen, from Denmark, presenting the experience of using Ada 2012 in the development of Alice, a core component of a hosted telephone reception system.

This issue also publishes the more detailed reports of the four sessions of the 16th International Real-Time Ada Workshop, which took place at York, UK, last April. The workshop discussed several important topics for the evolution of Ada such as fine-grain parallel models, multiprocessor locking protocols, deferred setting of attributes, execution time timers, or potential extensions to the Ravenscar profile. The reader is encouraged to analyze the discussion and results, and collaborate in the open topics for a next edition of the workshop which is tentatively planned for the fall of 2014.

Afterwards, the issue starts the publication of a set of articles on the Rationale for SPARK 2014, based on information and posts available at www.spark-2014.org, provided by Yannick Moy of AdaCore, France. These articles will allow the reader to know the main topics of this major SPARK evolution, which both aligns with the new contracts specification capabilities of Ada 2012 and provides new capabilities, profiles, and automation tools.

Finally, the topic of safer software is also present in the Ada Gems section, which provides two gems on Ada 2012 assertions, also by Yannick Moy.

Luís Miguel Pinho
Porto
December 2013
Email: AUJ_Editor@Ada-Europe.org

Dear All,

I am writing to you on behalf of Ada-Europe, the international organization that promotes the use, the maintenance and the evolution of the Ada language and technology.

As part of its charter, Ada-Europe organizes a high-quality yearly International Conference on Reliable Software Technologies (http://www.ada-europe.org/confs/ae), produces and distributes the Ada User Journal, a fine quarterly magazine (http://www.ada-europe.org/auj/home), and offers financial grants to initiatives that help further the resonance and relevance of Ada in engineering and scientific domains.

An important proportion of the Ada-Europe grant program supports the language maintenance process and, when the time comes, the production of the language Rationale and the Reference Manual books produced by Springer as part of their famous LNCS series.

The total volume of those financial undertakings is very significant for a not-for-profit organization such as us, which we can only sustain thanks to the generous support by our past, present and future sponsors.

This letter is a call for sponsors for the year 2014.

On behalf of Ada-Europe I would like you to consider becoming a sponsor. Our sponsorship program offers multiple ways in which you can flexibly design your sponsorship package, dependent on your business, your need for visibility, and your wish to selectively support specific initiatives, across the whole spectrum of Ada-Europe's activities, from the yearly conference, to the Ada User Journal and our web presence, to the language books and the related language maintenance initiatives.

Visibility at the annual conference is attractive for organizations that want to present product offerings or industrial capacity. Our yearly conference attracts over 100 delegates from Europe, the USA, and occasionally from Australia, South America and Asia, with equally sized presence from industry and academia, and has a contact list of more than 1,500 professionals.

Also the Ada User Journal and the Ada-Europe web site are vehicles for visibility, not only for technology vendors, but also for organizations who want to show their support for Ada. We know the latter are numerous and we would be delighted to see them become active sponsors of Ada-Europe.

Further financial aid is currently very much required in order that we can support the production of the Reference Manual for Ada 2012 as a Springer LNCS book, after completing the production of the Ada 2012 Rationale, which is also about to be printed by Springer. This is really for everyone, regardless of size and business, who wants the latest version of Ada to attain the prominence that it deserves.

I really hope you will consider becoming a sponsor for some of the above initiatives of Ada-Europe in the year 2014. Sponsorship packages can be designed to suit both large and small organizations, and start with as little as 350 EUR.

In case you wanted to know more or have some specific interest – even if only provisional at this time – may I kindly invite you to make contact with the Treasurer of Ada-Europe at <*treasurer@ada-europe.org*>.

In response to an expression of interest from you, our Treasurer will contact you by phone and discuss with you your possible sponsorship profile for the year 2014.

I do hope you will find this proposal of some interest and I look forward to including your Company as a valued 2014 sponsor.

Yours sincerely.

President, Ada-Europe:
Tullio Vardanega
University of Padova
Department of Mathematics
via Trieste 63
I-35121 Padova, Italy

phone: +39-049-8271359
fax: +39-049-8271499
email: president@ada-europe.org

Ada-Europe ivzw/aisbl
http://www.ada-europe.org
Legal address:
c/o Offis nv/sa – Aubay Group
Gatti de Gamondstraat 145
B-1180 Brussels, Belgium

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]

FOSDEM 2014

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Sun, 20 Oct 2013 17:48:46 +0000
Subject: CfP - Ada Developer Room at FOSDEM 2014, Brussels, Belgium
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Preliminary Announcement and Call for Presentations

5th Ada Developer Room at FOSDEM 2014

Saturday 1 February 2014, Brussels, Belgium

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/14/140201-fosdem.html

Organized in cooperation with Ada-Europe

Ada-Belgium [1] is pleased to announce that there will be a one-day Ada Developer Room on Saturday 1 February 2014 at FOSDEM 2014 in Brussels, Belgium. This Ada DevRoom is once more organized in cooperation with Ada-Europe [2].

General information

FOSDEM [3], the Free and Open source Software Developers' European Meeting, is a free and non-commercial two-day weekend event organized each February in Brussels, Belgium. It is highly developer-oriented and brings together 5000+ participants from all over the world.
The goal is to provide open source developers and communities a place to meet with other developers and projects, to be informed about the latest developments in the open source world, to attend interesting talks and presentations on various topics by open source project leaders and committers, and to promote the development and the benefits of open source solutions.

Ada Developer Room

At previous FOSDEM events, Ada-Belgium has organized very well attended Ada Developer Rooms, offering a full day program in 2006 [4], a two-day program in 2009 [5], and full day programs in 2012 [6] and 2013 [7]. One of our important goals is to present Ada also to people outside the traditional Ada community.

Our proposal for another dedicated Ada DevRoom was accepted recently, and now work continues to prepare the detailed program. We most probably will have a total of 8 schedulable hours between 10:00 and 18:00 in a room which holds up to 80 participants. More information will be posted later on the dedicated web-page on the Ada-Belgium site [8], and final announcements will of course also be sent to various lists and newsgroups.

Call for presentations

Ada-Belgium calls on you to:

- inform us at ada-belgium-board@cs.kuleuven.be about specific presentations you would like to hear in this Ada DevRoom;
- for bonus points, subscribe to the Ada-FOSDEM mailing list [9] to discuss and help organize the details;
- for more bonus points, be a speaker: the Ada-FOSDEM mailing list is the place to be!

Do you have a talk you want to give?

Do you have a project you would like to present?

Would you like to get more people involved with your project?

We're looking for proposals that are related to Ada software development, and include a technical oriented discussion.

You're not limited to slide presentations, of course. Be creative. Propose something fun to share with people so they might feel some of your enthusiasm for Ada!

Speaking slots are 25 or 50 minutes, including Q&A. Depending on interest, we might also have a session with lightning presentations (e.g. 5 minutes each).

We'd like to put together a draft schedule by the end of November. So, please act ASAP, and definitely before November 30, 2013.

We look forward to lots of feedback and proposals!

---

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

[2] http://www.ada-europe.org

[3] https://fosdem.org

[4] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/06/060226-fosdem.html

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/09/090207-fosdem.html

[6] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/12/120204-fosdem.html

[7] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/13/130203-fosdem.html

[8] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/14/140201-fosdem.html

[9] http://listserv.cc.kuleuven.be/archives/adafosdem.html

Meeting in Stockholm

From: Åke Ragnar Dahlgren <ake.ragnar.dahlgren@gmail.com>
Date: Fri, 4 Oct 2013 00:07:46 -0700
Subject: Ada meetup in Stockholm the 16:th of October
Newsgroups: comp.lang.ada

I was checking meetup.com the other day for things to do in Stockholm. Looks like there's an Ada meetup the 16:th of October.

http://www.meetup.com/Ada-Stockholm/

Best regards,

Åke Ragnar Dahlgren

GNAT Industrial User Day Presentations

From: Jamie Ayre <ayre@adacore.com>
Date: Tue Oct 8 2013
Subject: GNAT Industrial User Day Presentations
URL: http://www.adacore.com/developers/development-log/gnat-industrial-day-user-presentations/

The following slides are from presentations given at the GNAT Industrial User Day Conference on September 25, 2013 in Paris.

["AdaCore Roadmap for 2013-2015", "GNATdashboard" and "GNAT Pro for ARM". —sparre]

Vermont Tech CubeSat Launch Delayed

From: VTDigger
Date: Sun Oct 20 2013
Subject: Launch of Vermont Tech's lunar CubeSat delayed by government shutdown
URL: http://vtdigger.org/2013/10/20/launch-vermont-techs-lunar-cubesat-delayed-government-shutdown/

A small satellite built and programmed at Vermont Technical College will soon be orbiting Earth, but its launch date has been pushed back because of the government shutdown. Still, the college will be the first in New England to have its own cube satellite launched from NASA's Mid-Atlantic Regional Spaceport in Virginia, beating MIT and Harvard, among others.

[…]

The launch, which was scheduled for Nov. 4, has been delayed because of the government shutdown. Brandon is now waiting to get a new launch date from NASA. But he's not too concerned about the delay. His cube satellite will still be in space before MIT's.

[…]

[The Vermont Tech CubeSat is programmed in Ada and SPARK. —sparre]

Ada and Education

AdaCore University

From: AdaCore Press Center
Date: Wed Sep 25 2013
Subject: AdaCore Launches Free, Online Ada Educational Resource for the Software Development Community
URL: http://www.adacore.com/press/adacoreuniversity/

AdaCore today launched AdaCore University - a free, web-based resource center for anyone interested in learning about, or how to program in, the Ada programming language. The new website offers pre-recorded courses and other learning materials on Ada, with access to AdaCore's GNAT Ada toolset for writing and running example programs. It also utilizes the latest in website design and learning tool features. Students at all levels of experience and expertise can begin writing programs quickly and can proceed at their own pace.

AdaCore University courses educate through examples, allowing students to see, understand and experiment with most features of the Ada programming language. Drawing on the experience and teaching credentials of Ada experts, such as AdaCore founders and New York University Emeritus Professors Robert Dewar and Edmond Schonberg, the courses explain Ada's technical concepts with insight into the rationale and usage of particular features.

The initial curriculum includes two courses:

- Ada 001, "Overview" – a module that presents an overall picture of the language and that allows students to write small programs; and
- Ada 002, "Basic Concepts" – the first in a formal series of Ada classes, introducing basic Ada programming concepts and allowing students to write programs based on these features.

Both of these modules, and all future courses, provide sources and installation instructions for all learning materials and tools. The courses cover the latest version of the Ada language (Ada 2012), and students have access to AdaCore's GNAT Ada development environment and programming tools. The AdaCore University website also hosts a number of technical papers on Ada, offering insight into particular aspects of the language's design and usage.

AdaCore University is an ongoing, live project that will be expanded to include more advanced courses on Ada, and SPARK 2014 – an Ada-based programming language designed for high-integrity software (i.e., where reliability is essential and where safety and/or security certification may be required).

For more information on AdaCore University please visit http://u.adacore.com.

Ada-related Resources

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed Nov 13 2013
Subject: Repositories of Open Source software
To: Ada User Journal

AdaForge: 7 repositories [1]

Bitbucket: 54+ repositories [2,3,4]

Codelabs: 17 repositories [5]

GitHub: 438 repositories [6]
118 developers [7]

Rosetta Code: 570 examples [8]
26 developers [9]

Sourceforge: 224 repositories [10]

[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=library&language=ada

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

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

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

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

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

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

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

[See also "Repositories of Open Source Software", AUJ 34-3, p. 138. —sparre]

Ada-related Tools

YAMI4

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 3 Sep 2013 02:30:55 -0700
Subject: YAMI4 1.8.0 released
Newsgroups: comp.lang.ada

I'm pleased to announce that the new version of YAMI4, 1.8.0, was just released:

http://inspirel.com/yami4/

YAMI4 is a messaging solution for distributed systems. This new release provides a range of improvements for all supported programming languages; from the Ada point of view the most important is the extension of the data model, which now allows to create nested arrays of parameters objects.

[See also "YAMI4", AUJ 33-4, p. 236. —sparre]

Sparkel Programming Language

From: Jamie Ayre <ayre@adacore.com>
Date: Tue Sep 10 2013
Subject: Sparkel Programming Language
URL: http://www.open-do.org/2013/09/10/sparkel-programming-language/

Sparkel is a new parallel programming language inspired by the SPARK subset of Ada, and designed to support the development of inherently safe and secure, highly parallel applications that can be mapped to multicore, manycore, heterogeneous, or distributed architectures.

To learn more about Sparkel and to follow the project, please visit http://www.sparkel.org

From: S. Tucker Taft, AdaCore
Date: Thu Sep 26 2013
Subject: FrontPage - sparkel
URL: http://www.sparkel.org/

[…]

Sparkel Introduction

Sparkel is intended to be a lean, elegant, parallel language inspired by the SPARK subset of Ada. Sparkel is for both specifying and implementing parallel applications. As such, it includes high-level specification features, including parameterized types with full separation of interface from implementation, pre- and postconditions for individual operations of a type, invariants that apply across all operations of a type, and constraints that apply to individual subtypes.

Sparkel provides support for both implicit and explicit parallelism. Every Sparkel expression is defined to have parallel evaluation semantics. That is, given a Sparkel expression like F(X) + G(Y), the language rules ensure that it is safe to evaluate F(X) and G(Y) in parallel. The compiler makes the decision based on complexity or other criteria whether a given computation should be created as a potentially parallel activity. An underlying scheduler then maps these potentially parallel activities to particular processing resources, by default using a work-stealing approach, which provides load balancing across processors while also providing good locality of reference and minimal cache contention.

The primary approach to ensuring the safe parallelism is by simplification of the language, with the elimination of features that interfere with safe parallelization. In particular, Sparkel:

- eliminates global variables — operations may only access variables passed as parameters;
- eliminates parameter aliasing — two parameters passed to the same operation must not refer to the same object if either parameter is updateable within the operation;
- eliminates pointers — optional and expandable objects and generalized indexing provides an approach that allows safe parallelization;
- eliminates run-time exception handling — strong compile-time checking of preconditions and support for parallel event-handling provides a safer alternative;
- eliminates a global garbage-collected heap — automatic storage management is provided using region-based storage management which provides immediate, automatic reclamation of storage with none of the global contention and disruption associated with a global garbage-collected heap;
- eliminates explicit threads, lock/unlock, or signal/wait — parallel activities are identified automatically by the compiler, and language rules prevent data races between readers and writers of the same object, while explicitly protected objects can be used for safe synchronization when concurrent access from multiple readers and writers is required, without any need for explicit lock/unlock or signal/wait.

[…]

Additions to AVR-Ada

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon Sep 16 2013
Subject: [Avr-ada-devel] New stuff: I2C/TWI via USI, teensy++ 2.0, and Arduino Leonardo
To: avr-ada-devel@lists.sourceforge.net

Just a note that I pushed a bunch of changes to AVR-Ada repo.

Main highlights are:

- I2C master (TWI in Atmel terms) via USI interface of attiny MCUs
- Better support for at90usb1286 MCU and teensy++ 2.0[1] board (gnatmake -XBOARD=teensyplusplus2 …)
- Support for atmega32u4, which is on Arduino Leonardo[2], Arduino Micro and teensy 2.0 devices (no "board" support for these yet)

AVR.USI_TWI package might see (API) changes still, so use with care. Also, if you have improvement ideas, please send them to me.

at90usb1286 and atmega32u4 MCUs don't have all peripherals supported, but at least UART and Timer0/Timer1 should work.

In addition, the clock frequency of at90usb1286/teensy++ 2.0 is somewhat problematic since teensy++ 2.0 has 16MHz crystal but by default that is divided by 8, so running frequency is 2MHz.

teensy++ 2.0 author also encourages to change this frequency via MCU.CLKPR register, so the code get timings totally wrong if don't change CLKPR to the expected value at the beginning.

For now, teensyplusplus2 board expects 16MHz frequency, and I am using following code in my programs to set the frequency:

```
MCU.CLKPR := 16#80#;
MCU.CLKPR := 16#00#;

-- .. rest of the code
```

AVR-Ada for Teensy, Arduino Leonardo and Arduino Micro

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon 16 September 2013
Subject: AVR-Ada gets teensy 2.0, teensy++ 2.0, Arduino Leonardo and Arduino Micro support
URL: http://arduino.ada-language.com/avr-ada-gets-teensy-20-teensy-20-arduino-leonardo-and-arduino-micro-support.html

I just pushed initial support for atmega32u4 and improved support for at90usb1286 MCUs to AVR-Ada repository. This means you can now use AVR-Ada with your teensy and Arduino Leonardo devices.

Teensy 2.0, Arduino Leonardo, and Arduino Micro devices have atmega32u4 processor and teensy++ 2.0 has at90usb1286 processor. These processors have USB functionality built in the processor itself, so in theory you can let them simulate USB keyboards, mouses, and other devices relatively easily. However, AVR-Ada itself does not contain any USB code yet, so you are expected to do everything from scratch by yourself.

In addition, I also pushed some USI code for attiny processors and now you can use attiny2313 or attiny4313 as I2C master.

Request: Triplestore

From: Peter Brooks <peter.h.m.brooks@gmail.com>
Date: Mon Sep 16 2013
URL: http://www.linkedin.com/
Subject: Is there a triplestore written in Ada?

I have tried to find one, but it may just be that I've not looked hard enough. If so, sorry for wasting your time.

If there isn't one, is there a project working on writing one?

[See http://en.wikipedia.org/wiki/Triplestore for a definition. —sparre]

SDL Binding

From: Kevin Keith <krfkeith@gmail.com>
Date: Fri, 20 Sep 2013 21:30:07 -0700
Subject: SDL Bindings?
Newsgroups: comp.lang.ada

I've done some searches for SDL bindings, however they all seem to be somewhat incomplete. Moreover, they all appear to be of the thin variety. How

involved would writing a set of thick bindings for SDL be?

From: Oliver Kleinke <oliver.kleinke@c-01a.de>
Date: Sat, 21 Sep 2013 18:51:50 +0200
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

> [...]

Pro tip: Start with a thin binding, you can use that code in your thick binding later on ...

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 21 Sep 2013 21:59:27 +0000
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

> [...]

I've been thinking about doing this again for a while so I started binding SDL 2.0 yesterday. It'll be a variable thickness binding.

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sat, 21 Sep 2013 16:24:43 -0700
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

If you are using GNAT, did you consider the SDL bindings that they provide ?

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 22 Sep 2013 00:29:36 +0000
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

> [...]

1) I didn't realise AdaCore provided any, and

2) I wouldn't touch then with yours given their propensity for slapping pure GPL licences on their libraries.

From: Kevin Keith <krfkeith@gmail.com>
Date: Sun, 22 Sep 2013 22:59:34 -0700
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

The SDL bindings in GNAT are just cleaned up auto-generated thin bindings.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Tue, 24 Sep 2013 02:25:40 -0700
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

> [...]

And they work? If so that's great news for me. I've been looking for a development setup for complex game-like apps with the requirements:

1. main program written in Ada

2. cross platform, from desktop to mobile

I have considered Qt but the Ada bindings don't seem good enough. They are either at Qt 4, only old desktop GUI style, no mobile style interface, or Qt 5 only for Windows.

I have considered Gtk but it does not seem cross platform enough, and also only old desktop GUI style, no mobile style interface.

From: Kevin Keith <krfkeith@gmail.com>
Date: Tue, 24 Sep 2013 17:39:27 -0700
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

That's an excellent question actually, and I'm not sure. I'd have to do some more research. I *believe- the package is gnat-sdl.

From: Thomas Løcke <tl@ada-dk.org>
Date: Wed, 25 Sep 2013 07:38:38 +0200
Subject: Re: SDL Bindings?
Newsgroups: comp.lang.ada

On 09/24/2013 11:25 AM, Marius Amado-Alves wrote:

> From this thread and the web it seems that GNAT + SDL + Agar could be it.

I would love to hear from you about your Ada + Agar experience.

Please don't forget to tell the community about how things pan out. :o)

Galois Fields

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Sun, 22 Sep 2013 02:43:13 -0700
Subject: Re: Galois fields
Newsgroups: comp.lang.ada

> Are Galois Fields (I only need binary GFs) supported by any publicly available Ada library?

I have a GF binary library that I wrote for myself in my early Ada days. It is a generic package that you must instantiate with the desired GF size (up to 2^64, so it fits in a 64-bit integer).

Unfortunately I did not publish the code by itself, but you can find it "embedded" in an old (and, alas, sleeping) project of mine hosted on launchpad. Just go here

http://bazaar.launchpad.net/~riccardo-bernardini/+junk/pre-ppetp/files/head:/src/lib/Algebra/Galois/

(or here: http://bit.ly/1gOe996 if the URL above is too long) and take the two files gf_2p_varsize.{ads, adb}. You just need to instantiate it with something like

```
package GF32 is
   new Gf_2p_Varsize(Exponent   => 32,
                     Basic_Type => Unsigned_32);
```

and then use it like

```
X, Y, Z : GF32.Galois;
X = Y * Z;
```

Disclaimer: As I said, I wrote this package in my early Ada days, so maybe it is not the most elegant code around, but it works…

Should you decide to use it and have any question, please ask.

Excel Writer, GNAVI, Mathpaqs and Zip-Ada

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Mon Sep 30 2013
Subject: September 2013 releases
URL: http://gautiersblog.blogspot.dk/2013/09/september-2013-releases.html

New maintenance releases have been uploaded for…

- Excel Writer [1]

- GNAVI: GNU Ada Visual Interface [2]

- Mathpaqs [3]

- Zip-Ada [4]

Enjoy!

[1] http://excel-writer.sf.net/

[2] http://sourceforge.net/projects/gnavi/

[3] http://sf.net/projects/mathpaqs/

[4] http://unzip-ada.sf.net/

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

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

[See also "Mathpaqs, February 2011", AUJ 32-2, p. 72. —sparre]

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

PLplot

From: Jerry Bauck <lanceboyle@qwest.net>
Date: Tue, 1 Oct 2013 17:31:44 -0700
Subject: ANN: PLplot plotting library with Ada bindings
Newsgroups: comp.lang.ada

PLplot 5.9.10 has just been released. PLplot is an extensive plotting library with Ada bindings.

http://plplot.sourceforge.net/

In addition to the bindings, the Ada component adds substantial ease-of-use functionality that eliminates the need to learn and code a lot of set-up routines. These easy-to-use routines will work for most day-to-day plotting. For example, to make an x-y plot:

```
with PLplot;
use PLplot;
procedure Simple_Example is
  x, y : Real_Vector (-10 .. 10);
begin
  for i in x'range loop
    x(i) := Long_Float (i);
    y(i) := x (i) ** 2;
  end loop;
  Initialize_PLplot; -- Call this only once.
  Simple_Plot (x, y); -- Make the plot.
  -- Make more plots here.
  End_PLplot;       -- Call this only once.
end Simple_Example;
```

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 01 Oct 2013 17:45:57 -0700
Subject: Re: ANN: PLplot plotting library with Ada bindings
Newsgroups: comp.lang.ada

> Initialize_PLplot; -- Call this only once.

Why doesn't the elaboration of Plplot do this?

> End_PLplot; -- Call this only once.

Why not use finalization to do this?

From: Graham Stark <graham.stark@virtual-worlds.biz>
Date: Tue, 12 Nov 2013 07:27:37 -0800
Subject: Re: ANN: PLplot plotting library with Ada bindings
Newsgroups: comp.lang.ada

I'm looking for a plotter routine that could be used safely inside the AWS web server, so I can implement a 'callback' chart server (currently I have one written in Java).

Could this be used for that? Some simple tests suggest to me that it isn't thread safe. The Initialize_PLplot and associated procedures to set filenames, etc, seem to set global variables somewhere. Is there some trick I'm missing?

It does make lovely looking charts.

From: Jerry Bauck <lanceboyle@qwest.net>
Date: Wed, 13 Nov 2013 14:51:31 -0800
Subject: Re: ANN: PLplot plotting library with Ada bindings
Newsgroups: comp.lang.ada

Thanks for your interest in PLplot. I don't know how to answer your question about thread safety so I put it to the PLplot development list. There are two responses so far, from Alan and Hezekiah.

Alan

To answer the question at hand, I am virtually positive PLplot is not thread safe, but you should wait for Andrew's response for the definitive view on that, especially the question of what would need to be done to make PLplot thread safe and ideally a plan for getting there.

Just as important as thread safety in my option is security. If I were a webserver designer interested in safe plotting, then it is important to acknowledge that plotting software by its very nature is inherently insecure; the problem is that plotting software has lots of different user input channels (titles, text annotations, legends, colorbars, etc.,) that could be the source of potential buffer overflows or other intrusion possibilities. We do make some concious decisions for PLplot development to avoid obvious security issues, but at the same time security is not our primary interest and certainly not a fundamentally important area of expertise for us. And I am sure that is the case for developers of other plotting software as well; we are all primarily interested in making pretty pictures ("lovely looking charts") rather than designing secure software. :-)

So for any plot software including PLplot, the web designer should filter down the possible user input channels as much as possible (ideally no user-controlled input text allowed at all). After that, a full security audit (only possible with open-source plotting software such as PLplot) should be done of what is left to target by a malicious user after such filtering. And we would certainly be happy to accept patches that were the result of any such audit.

Hezekiah

PLplot is not thread safe. While you can use PLplot in a threaded program, only one thread per process may interact with PLplot at a given time. This limitation holds even if you are working with multiple plot streams in a single process.

Regarding Alan's follow-up - while it is possible to make PLplot thread-safe, the changes required are invasive and pervasive. They are all good changes to make! But there is a lot to be done and the result is a completely backwards-incompatible API. The three big pieces required are:

a) All PLplot functions will need to explicitly operate on a given plstream value representing the affected plot stream. This requires adding an additional stream argument to all PLplot functions and removing any global state from plot streams.

b) Remove all of the globals used through the PLplot code base in the actual plotting logic. One example is the contour/shading routines which use several global variables to track their state.

c) Confirm/ensure that each of our output devices can be and are used in a thread-safe manner.

Each of these big pieces is made up of several smaller chunks. (a) is where the API breakage would come in. It is also likely the simplest (simple being relative here!) to complete. (b) could be pretty hairy as the logic in the contouring routines in particular is tricky to translate to something which doesn't use globals. (c) should be attainable for at least the Cairo, Qt and built-in output drivers (SVG, PS, null). I would be happy to help in putting together a plan for this work. Unfortunately my PLplot time is very limited these days so it's unlikely I'll be able to provide much development assistance.

VTKAda

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Thu, 3 Oct 2013 04:50:11 -0700
Subject: I'm pleased to announce VTKAda version 6.0 free edition release 01/10/2013
Newsgroups: comp.lang.ada

VTKAda is an Ada-2012 binding to Visualization Toolkit by Kitware (VTK) and the Qt5 application and UI framework by Nokia.

VTK version: 6.0.0

Qt version: 5.1.1

Built with Microsoft Visual Studio 2012 in Windows and gcc in Linux x86-64. Package was tested with the GNAT-GPL-2012 Ada compiler (-gnat12 option) on Windows 8 64bit and Debian 7 x86-64.

With VTKAda(+QtAda) you can build any desktop applications with powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, Internet browsing and many others things.

Current state of VTKAda is 42064 procedures and function distributed in 672 packages. 135 examples. All QtAda examples are Qt5 applications.

Current state of QtAda is 11925 procedures and function distributed in 324 packages. There are many new packages and examples in this release.

VTKAda you can be used without the QtAda subsystem.

QtAda is an Ada binding to the Qt5 framework and can be used as an independent system.

VTKAda and QtAda for Windows and Linux (Unix) free edition with prebuilt Qt 5.1 and VTK 6/0 are available from

http://www.multiupload.nl/ST1R8GBDGW

[See also "VTKAda", AUJ 33-3, p. 144. —sparre]

Embedded Web Server

From: Simon Wright <simon@pushface.org>
Date: Wed, 16 Oct 2013 16:58:19 +0100
Subject: Embedded Web Server 20131016
Newsgroups: comp.lang.ada

This release[1] of EWS allows the software to be built as a library.

It requires an Ada 2012 capable compiler (GCC 4.8 or later, GNAT GPL 2012 or later (but "make install" requires GNAT GPL 2013's gprinstall)).

The licence has been changed to GPL v3, with the GCC Runtime Library Exception v3.1 in place of GMGPL.

It no longer requires the Booch Components (XML/Ada is still required).

[1] https://sourceforge.net/projects/embed-web-srvr/files/ews-20131016/

[See also "New release of the Embedded Web Server", AUJ 32-1, p. 11. —sparre]

AdaMAnT (MAT I/O in Ada)

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Fri, 18 Oct 2013 13:09:20 -0700
Subject: Re: Ada library to read/write matlab files?
Newsgroups: comp.lang.ada

[…] I "threw together" a solution good enough for me:

https://launchpad.net/adamant

Currently it is very limited (but you can copy it! :-) [did you get the joke? no?!? Then, what are you doing in this newsgroup? :-) :-) :-)]

It allows you only to write Matlab files and only matrices of double (of any dimensionality) and strings. Cells, structs and reading are for a future release (maybe)

It requires Ada 2012 since I spotted the occasion of using a dynamic predicate (too cool!). If you want to use it with a non Ada 2012 compiler, just go to the specs of Matlab.IO and remove the predicate from Matlab_Name.

Request: Health Level 7/MLLP implementation?

From: Peter Brooks <peter.h.m.brooks@gmail.com>
Date: Fri, 18 Oct 2013 22:20:24 -0700
Subject: HL7/MLLP - any work on reliable implementation in Ada?
Newsgroups: comp.lang.ada

Does anybody know if there is a working group anywhere to produce a reliable version of the interchange protocol MLLP that supports the Health Level 7 standard?

Are there any implementations of HL7 components in Ada - or work to produce these?

Ada 2012 Grammar

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 25 Oct 2013 21:07:04 -0500
Subject: Re: Ada 2012 grammar
Newsgroups: comp.lang.ada

> […] a complete grammar for Ada 2012, lets say, in Flex+Bison or similar compiler construction tool-chain?

Website: http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html

The grammar there has conflicts, so it relies on a generalized LALR parser (http://en.wikipedia.org/wiki/Generalized_LR_parser), which I implemented in Emacs lisp.

The grammar file can be parsed by a new OpenToken feature, and it will generate either Ada OpenToken source code, or Emacs lisp for a parser table; that version of OpenToken is available on the Ada mode website (not the OpenToken website http://stephe-leake.org/ada/opentoken.html).

However, OpenToken only has a (non-generalized) LALR parser. It would be interesting to implement a generalized parser for Opentoken; let me know if you'd like to use that and/or help implement it.

It might not be too hard to eliminate the conflicts so you can use the OpenToken parser (or some other parser); I did not do that because it seemed easier (not to say way more fun) to implement the parser :). In addition, the grammar source is closer to the Ada LRM Annex P this way.

I've improved Ada mode a bit since the last post on the website; it's quite usable now. The latest is in monotone; see the Ada mode website for access info. It is time to post another release; maybe this weekend.

Also note that I fixed several bugs in OpenToken, so if it was not working for you before, you should try it again.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 11 Nov 2013 08:55:15 -0600
Subject: Re: Ada 2012 grammar
Newsgroups: comp.lang.ada

> Version 4.0b 29 Jun 2010?

That's the "current stable release" version at the OpenToken website http://stephe-leake.org/ada/opentoken.html, and also in Debian distributions.

The experimental, alpha release version that supports creating a parse table for a generalized parser is at http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html

OS-Lovelace

From: Xavier Grave <xavier.grave@ipno.in2p3.fr>
Date: Tue, 29 Oct 2013 19:25:42 +0100
Subject: Re: Try to compile OS-Lovelace with gcc-4.6
Newsgroups: comp.lang.ada

> I was looking the internet for some code, but I didn't find OS-Lovelace. Can you provide a link to the code, please?

You should use monotone :

Prompt> mtn --db=./base_lovelace.db db init

Prompt> mtn --db=./base_lovelace.db pull

mtn://www.ada-france.org?org.os-lovelace.*

Prompt> mtn --db=./base_lovelace.db list branches

org.os-lovelace.micro-kernel

org.os-lovelace.multi

org.os-lovelace.toy

org.os-lovelace.tutorial

The tutorial branch contains some test code for C, Ada for x86 and ARM architectures.

The micro-kernel one is the implementation of the code tested in the previous branch.

The toy branch is an archive of the Ada translation of SOS (Simple OS, written in C), there is support for Ada tasking, OO and exceptions in it.

To commit a branch :

Prompt> mtn --db=./base_lovelace.db co -b org.os-lovelace.micro-kernel

Hope it will help, Xavier

GNATColl ORM foreign key twins

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed Oct 30 2013
Subject: GNATColl ORM foreign key twins
URL: http://repositories.jacob-sparre.dk/gnatcoll-orm-foreign-key-twins

Demonstration source text for using the GNATColl ORM generator, when you have a table for linking pairs of rows in another table together.

FNV1 and FNV1a Hash Functions

From: Simon Belmont <sbelmont700@gmail.com>
Date: Wed, 30 Oct 2013 16:31:39 -0700
Subject: FNV-1
Newsgroups: comp.lang.ada

I did up the following implementation of the FNV1 and FNV1a hashes after needing a decent cross-platform, compiler-independent way to hash things (64-bit pointers, specifically), but it ought to be good for other things as well. I haven't done any extensive testing, but it seems to suit my needs; perhaps it will suit yours as well. Any suggestions are welcome, especially tips on better ways to use static expressions, or any cross-platform 'gotchas'. The code is public domain.

```
-- Fowler/Noll/Vo hash functions
with Ada.Containers;

package FNV is
   -- FNV-1 Hash
   generic
      type T is private;
   function FNV1 (Item : T) return
            Ada.Containers.Hash_Type;

   -- FNV-1a Alternative Hash
   generic
      type T is private;
```

```
   function FNV1a (Item : T) return
            Ada.Containers.Hash_Type;
end FNV;

pragma Assertion_Policy (Check);
with Ada.Storage_IO;

package body FNV is
   subtype FNV_Hash_Type is
            Ada.Containers.Hash_Type;
   use type FNV_Hash_Type;

   -- Prime Values
   Prime_32   : constant :=
            2**24  + 2**8 +16#93#;
   Prime_64   : constant :=
            2**40  + 2**8 +16#b3#;
   Prime_128  : constant :=
            2**88  + 2**8 +16#3b#;
   Prime_256  : constant :=
            2**168 + 2**8 + 16#63#;
   Prime_512  : constant :=
            2**344 + 2**8 +16#57#;
   Prime_1024 : constant :=
            2**680 + 2**8 +16#8d#;

   subtype FNV_Prime_Type is
            FNV_Hash_Type range 1 ..
            FNV_Hash_Type'Last;

   K_Prime : constant := (case
            FNV_Prime_Type'Size is
                  when   32 => Prime_32,
                  when   64 => Prime_64,
                  when  128 => Prime_128,
                  when  256 => Prime_256,
                  when  512 => Prime_512,
                  when 1024 => Prime_1024,
                  when others => 0);

   Prime  : constant FNV_Prime_Type :=
            K_Prime;

   -- Start Offset Values
   Offset_32   : constant := 2166136261;
   Offset_64   : constant :=
            14695981039346656037;
   Offset_128  : constant :=
144066263297769815596495629667062367
629;
   Offset_256  : constant :=
100029257958052580907070968620625704
837092796014241193945225284501741471
925557;
   Offset_512  : constant :=
965930312949666949800943540071631046
609041874567263789610837432943446265
799458293219771643844981305189220653
980578449532823934008387619192870158
3869517785;
   Offset_1024 : constant :=
141977950649476210687220706414032183
208806227954419339608784749146175827
232522967323037177221508640965212023
555493656281746691085718147604710150
761480297559698040773201576924585630
032153049571501574036444603635505054
127112859663616102678680828938239637
904393364110868845841077350106769l5;

   subtype FNV_Offset_Type is
            FNV_Hash_Type range 1 ..
            FNV_Hash_Type'Last;

   K_Offset : constant := (case
            FNV_Offset_Type'Size is
                  when   32 => Offset_32,
                  when   64 => Offset_64,
                  when  128 => Offset_128,
                  when  256 => Offset_256,
                  when  512 => Offset_512,
                  when 1024 => Offset_1024,
                  when others => 0);

   Offset : constant FNV_Offset_Type :=
            K_Offset;


   function FNV1 (Item : T) return
            Ada.Containers.Hash_Type is
      package Data_Buffers is new
         Ada.Storage_IO (Element_Type => T);
      Buffer : Data_Buffers.Buffer_Type;
   begin
      Data_Buffers.Write (Buffer => Buffer,
                  Item   => Item);

      return Hash : FNV_Hash_Type := Offset
      do
         for Byte of Buffer loop
            Hash := Hash * Prime;
            Hash := Hash xor
               FNV_Hash_Type(Byte);
         end loop;
      end return;
   end FNV1;

   function FNV1a (Item : T) return
            Ada.Containers.Hash_Type is
      package Data_Buffers is new
         Ada.Storage_IO (Element_Type => T);
      Buffer : Data_Buffers.Buffer_Type;
   begin
      Data_Buffers.Write (Buffer => Buffer,
                  Item   => Item);

      return Hash : FNV_Hash_Type := Offset
      do
         for Byte of Buffer loop
            Hash := Hash xor
               FNV_Hash_Type(Byte);
            Hash := Hash * Prime;
         end loop;
      end return;
   end FNV1a;
end FNV;
```

Mathematics and Statistics

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu Oct 31 2013
Subject: Mathematics and Statistics
URL: http://repositories.jacob-sparre.dk/mathematics-and-statistics

Various packages for mathematics and statistics.

[…]

Recent activity:

- Minor code clean-up.
- Fix for warnings from GNAT GPL 2013.
- Raw import of random number generator packages.
- Using fancy overflow checking.
- […]

Comfignat

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Wed, 6 Nov 2013 22:48:36 +0100
Subject: Comfignat 1.2 released
Newsgroups: comp.lang.ada

In August I published the first release of Comfignat, the makefile foundation and the abstract GNAT project for common, convenient, command-line-controlled compile-time configuration of software built with the GNAT tools on Unix-like operating systems. Today I have released Comfignat version 1.2. These are the most noteworthy changes:

- The interaction between directory variables and directories projects has been corrected so that a directories project overrides the default values of some directory variables, but an explicitly set Make variable overrides the corresponding variable in the directories project.
- A directory variable named “alidir” has been added so that installing users can control the placement of ALI files and binary libraries independently. This proved necessary in Debian, where ALI files are not kept in immediate subdirectories of libdir, but farther down.
- Comfignat's behaviour in sub-Makes has been fixed so that subprocesses working in subdirectories use the right build and staging directories.
- The persistent configuration feature has been improved so that Make variables that can be overridden by environment variables can also be configured from the environment. In subsequent Make invocations environment variables override values that were configured from the environment, and variables set on the command line override all configured values.
- A command “make show_configuration” has been added, making it easier to see the configured variables.

Comfignat resides at https://www.rombobjörn.se/Comfignat/.

Agar

From: Kevin Keith <krfkeith@gmail.com>
Date: Sat, 9 Nov 2013 04:52:10 -0800
Subject: Has anyone here used libAgar?
Newsgroups: comp.lang.ada

If so, would you recommend it? Are the bindings to it complete?

[Agar is a powerful open-source, cross-platform toolkit for graphical applications in C, C++ or Objective-C (bindings to Perl and Ada are also available). Designed for ease of integration, Agar follows the philosophy of building the GUI around the application, and not the other way around. Agar applications should work seamlessly under any platform, and Agar itself should work without relying on other libraries (Agar 1.4 can be compiled without dependencies, and has even been used on embedded devices without filesystems or operating systems at all). When compiled with optional threads support, Agar's entire documented API is thread-safe. —sparre]

Ada-related Products

Vector Software Achieves TÜV SÜD Re-Certification for Safety Related Software Development

From: Vector Software Press Releases
Date: Mon Aug 12 2013
Subject: Vector Software Achieves TÜV SÜD Re-Certification for Safety Related Software Development
URL: https://www.vectorcast.com/news/vector-software-press-releases/2013/press-release-vector-software-achieves-tuv-sud-re-certification

Vector Software, the world's leading provider of innovative software solutions for testing safety and mission critical embedded applications, today announced that it has re-certified its VectorCAST embedded software test platform and expanded its safety certification support to include railway applications certifiable to SIL 4 (safety integrity level).

TÜV SÜD Rail GmbH assessed VectorCAST as suitable for development processes that must comply with the stringent CENELEC EN 50128 standard. TÜV SÜD also confirmed the re-certification of the latest VectorCAST release to IEC 61508-3 (general industrial) and ISO 26262-8 (automotive). VectorCAST was first certified in February 2011.

TÜV SÜD Rail GmbH, an international service corporation focusing on consulting, testing, certification and training, assessed the VectorCAST tools for dynamic testing with respect to functional safety. The TÜV SÜD assessment and resulting tool qualification of the Vector Software products offer safety related development organizations the required evidence to demonstrate compliance with IEC 61508 and ISO 26262, and now CENELEC EN 50128.

VectorCAST is a family of co-operating software tools used for test automation in the development of embedded software applications. VectorCAST/C++, VectorCAST/Ada, VectorCAST/Cover, and VectorCAST/Manage support automated test execution for unit- and integration testing, test coverage analysis, and the management of test projects.

"We are pleased that VectorCAST is now a TÜV SÜD certified product for railway applications. This expands the certification we received over 2 years ago for automotive and industrial applications and validates the way our products are built," said William McCaffrey, Chief Operating Officer at Vector Software, Inc. "This is not a minor achievement. This certification separates us from our competitors and it is recognized by all Vector employees and departments that our clients can feel confident selecting our VectorCAST products for their most critical software development.

Ada Plugin for SonarQube

From: Maurizio Martignano
Date: Mon Oct 14 2013
Subject: Spazio IT just finished to port its Ada Plugin to SonarQube version 3.7.2!
URL: http://www.linkedin.com/

Spazio IT just finished to port its Ada Plugin to SonarQube version 3.7.2!

http://www.spazioit.com/pages_en/sol_inf_en/code_quality_en/

Ada and Operating Systems

New and Updated FreeBSD Ports

From: John Marino <dragonlace.cla@marino.st>
Date: Tue Jul 02 2013
Subject: New and Updated FreeBSD ports
URL: http://www.dragonlace.net/posts/New_and_Updated_FreeBSD_ports/

There has been a fair amount of activity for Ada in the FreeBSD Ports collection. Two new ports have been added:

- libsparkcrypto - Cryptographic library implemented in SPARK security/libsparkcrypto
- matreshka - Ada framework for information systems development devel/matreshka

Some existing ports have been updated:

- xmlada was updated from version 4.2 to 4.4
- gtkada was updated from version 2.22 to 2.24.4
- Gnat Programming Studio was updated from version 5.0.1 to 5.2.1

More Ada ports will be coming soon!

MacOS X Mavericks

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Fri, 25 Oct 2013 02:34:51 +0100
Subject: GNAT GPL 2013 on OS X Mavericks
Newsgroups: comp.lang.ada

I have installed Mavericks on my second computer (I'm not stupid 8-) and find that gnatmake fails at the link stage: ld fails to find libraries and reports them as missing from /usr/lib. They are present in Xcode for OS X 10.8, but not in 10.9. Adding the library location in Xcode to LIBRARY_PATH seems to make no difference. Copying one or two from Xcode to /usr/lib makes ld go on to complain about others; copying the lot borks Mavericks.

Reinstallation looms ...

BTW, doinstall in Mavericks demands that the Xcode licence terms be agreed to before it will complete the installation!

From: Martin Dowie <martin@thedowies.com>
Date: Thu, 24 Oct 2013 23:12:58 -0700
Subject: GNAT GPL 2013 on OS X Mavericks
Newsgroups: comp.lang.ada

No need!!!

Installing the command line tools is different. Try this http://www.computersnyou.com/2025/2013/06/install-command-line-tools-in-osx-10-9-mavericks-how-to/

I installed them via the Apple developer website but this should work. The CLTs used to be install able from within Xcode itself but that seems to have changed.

From: Martin Dowie <martin@thedowies.com>
Date: Thu, 24 Oct 2013 23:35:51 -0700
Subject: GNAT GPL 2013 on OS X Mavericks
Newsgroups: comp.lang.ada

https://developer.apple.com/downloads/index.action

The new Mavericks command line tools are there too.

References to Publications

Contract-based Programming

From: Benjamin M. Brosgol, AdaCore
Date: Sat Aug 24 2013
Subject: Contract-based programming: making software more reliable
URL: http://www.embedded.com/design/programming-languages-and-tools/4420114/Contract-based-programming--making-software-more-reliable

The elements of 'contract-based programming' – assertions of program properties that are part of the source text – have been available in some programming languages for many years but have only recently moved into the mainstream of software development. The latest version of the Ada language standard, Ada 2012, has added contract-based programming features that can be verified either dynamically with run-time checks, or statically through formal analysis tools. Both approaches help make programs more reliable by preventing errors from getting into production code.

[…]

Measuring Current With an Arduino

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon 26 Aug 2013
Subject: Measuring current with INA219 sensor
URL: http://arduino.ada-language.com/measuring-current-with-ina219-sensor.html

To optimize the power usage of my Arduino devices, I first needed to measure the power usage.

I could have done it traditionally with multimeter, but then I found Adafruit's INA219 sensor breakout board.

[image:http://farm4.staticflickr.com/3804/9358734151_73017e60f7.jpg]

Adafruit also has nice tutorial for the sensor (which I actually bought from oomlout.co.uk to avoid customs/extra taxes as European).

So, for the setup I gathered following parts:

- Arduino UNO, for reading the INA219 sensor
- The INA219 sensor itself
- Olimexino-328 as a target device for measurements
- Sparkfun Breadboard Power Supply for providing power to Olimexino-328.
- A small breadboard and a few jumper wires

When assembled together, the setup looks like this:

[image:http://farm8.staticflickr.com/7344/9361731678_8b441a72f2.jpg]

Code

The code consists of my I2C package and translation of Adafruit's INA219 code (written in C) to Ada.

The INA219 Ada package interface looks like this:

```
package INA219 is
   procedure Init;

   function Get_Bus_Voltage return
            Interfaces.Unsigned_16;
   function Get_Shunt_Voltage return
            Interfaces.Unsigned_16;
   function Get_Current return
            Interfaces.Unsigned_16;
   function Get_Power return
            Interfaces.Unsigned_16;

   function Get_Error return Boolean;

   procedure Set_Calibration_32V_2A;
end INA219;
```

Procedure Init needs to be called before other functions. It initializes the I2C bus and sets the default calibration.

Once the device is initialized and calibrated, you can query voltage, current, and power values with the related functions.

```
INA219.Init;
…
loop
   AVR.UART.Put ("   ***");
   AVR.UART.CRLF;
   AVR.UART.Put
            (INA219.Get_Bus_Voltage);
   AVR.UART.CRLF;
   AVR.UART.Put
            (INA219.Get_Shunt_Voltage);
   AVR.UART.CRLF;
   AVR.UART.Put (INA219.Get_Current);
   AVR.UART.CRLF;
   AVR.UART.Put (INA219.Get_Power);
   AVR.UART.CRLF;
   delay 1.0;
end loop;
```

Using the above program and opening the serial console, we can see how much power our target device draws. If you want to know how to change this value, you need to either do it by yourself or wait for future articles about AVR power saving using AVR-Ada :).

The code for the program is available in my arduino-blog repository (see examples/ina219 directory).

SPARK 2014 Rationale: Pre-call and Pre-loop Values

From: Yannick Moy
Date: Sat Sep 14 2013
Subject: SPARK 2014 : SPARK 2014 Rationale: Pre-call and Pre-loop Values
URL: http://www.spark-2014.org/entries/detail/spark-2014-rationale-pre-call-and-pre-loop-values

Subprogram contracts are commonly presented as special assertions: the precondition is an assertion checked at subprogram entry, while the postcondition is an assertion checked at subprogram exit. A subtlety not covered by this simplified presentation is that postconditions are really two-state assertions: they assert properties over values at subprogram exit and values at subprogram entry.

[…]

SPARK 2014 Rationale: Mixing SPARK and Ada Code

From: Yannick Moy
Date: Sun Oct 20 2013
Subject: SPARK 2014 : SPARK 2014 Rationale: Mixing SPARK and Ada Code
URL: http://www.spark-2014.org/entries/detail/spark-2014-rationale-mixing-spark-and-ada-code

The first step before any formal verification work with SPARK is to delimitate the part of the code that will be subject to formal verification (the code in SPARK) within the overall Ada application (which could also contain parts coded in C, in Java, in assembly, etc.).

[…]

How not to Design Safety Critical Software

From: Junko Yoshida
Date: Fri Oct 25 2013
Subject: Toyota Case: Single Bit Flip That Killed
URL: http://www.eetimes.com/document.asp?doc_id=1319903

Could bad code kill a person? It could, and it apparently did.

[A textbook example of how not to design safety critical software. —sparre]

From: Michael Dunn
Date: Mon Oct 28 2013
Subject: Toyota's killer firmware: Bad design and its consequences
URL: http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences

On Thursday October 24, 2013, an Oklahoma court ruled against Toyota in a case of unintended acceleration that lead to the death of one the occupants. Central to the trial was the Engine Control Module's (ECM) firmware.

[…]

[More on the Toyota engine control system. —sparre]

Saving Power with AVR-Ada

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue Nov 5 2013
Subject: Saving power with AVR-Ada
URL: http://arduino.ada-language.com/saving-power-with-avr-ada.html

As I got my INA219 sensor working, the next logical step was to actually find out how you can save some power with Arduinos.

Sparkfun provides a nice article about the subject and atmega328p datasheet is also an useful information source.

To save power, I basically did following things:

- Turned off all unnecessary peripherals.
- Turned off brown-out-detection (BOD, either via software or via FUSE bits).
- Slow down CPU frequency
- Put the processor in power save mode.

[Followed by detailed descriptions of each of the steps in the process — including source code. —sparre]

Summary:

When you need to a lower power Arduino, Diavolino or Olimexino-328 looks like the best bet.

If my measurements are correct, you can run Diavolino on 400mAh battery for one month or even more, if you run the board at 3.3V instead of 5V.

Also, I didn't even try all the available power saving tricks, so it should be possible to go even lower. (See the Sparkfun tutorial for details.)

The complete code is available from my arduino-blog repository [1], under examples/sleeper directory.

[1] https://bitbucket.org/tkoskine/arduino-blog/

Ada 2012 Language Rationale Published

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Tue, 12 Nov 2013 13:00:10 +0000
Subject: Press Release - Ada 2012 Language Rationale Published
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FOR IMMEDIATE RELEASE

Ada 2012 Language Rationale Published

New educational resource by Ada expert John Barnes explains key Ada 2012 concepts

---

PITTSBURGH, Pa., November 12, 2013 - Today at ACM SIGAda's HILT 2013 (High Integrity Language Technology) Conference, the Ada Resource Association (ARA) and Ada-Europe announced the publication of the Ada 2012 Rationale and its free availability for downloading. Sponsored in part by Ada-Europe, the Ada Resource Association, and AdaCore, the Ada 2012 Rationale was written by longtime Ada authority John Barnes. It summarizes the new Ada 2012 features, shows examples of their use, describes compatibility with earlier versions of the language standard, and explains the reasons behind critical language design decisions. This new Rationale will be a valuable resource for anyone interested in learning the innovations introduced by the Ada 2012 standard.

The Rationale may be downloaded at no cost from [1], [2], and [3]. The book may also be purchased through its commercial publisher, Springer, as volume LNCS 8338 in their Lecture Notes in Computer Science series, to be released in mid-December 2013.

[1] www.adaresource.com/rationale-2012/

[2] www.ada-europe.org/resources/online/

[3] www.adacore.com/rationale-2012/

The Ada 2012 Rationale contains the following chapters:

- Introduction, covering the development of Ada 2012 and giving a brief overview of the main changes from Ada 2005.
- Contracts and Aspects, describing the contract mechanism, one of the major enhancements in Ada 2012. It explains subprogram preconditions and postconditions, type invariants, and subtype predicates, and also presents the new unifying concept of "aspects".
- Expressions, describing the new flexible forms of expressions introduced in Ada 2012. These new forms - conditional expressions, quantified expressions, and expression functions - are especially useful in conjunction with contracts.
- Structure and Visibility, describing various improvements including the generalization of parameter modes to functions, additional flexibility with incomplete types, and new forms for "use" clauses and return statements.
- Tasking and Real-Time, describing various enhancements including control over task allocation on multiprocessor architectures, improvements to the scheduling mechanisms, and control of budgets with regard to interrupts.
- Iterators, Pools, etc., describing various improvements in a number of general areas in Ada 2012. These include important new features regarding indexing and accessing that simplify iterating over containers, and a subpool facility for additional flexibility in storage management.
- Predefined Library, describing a variety of minor improvements in areas including string and character handling, directory processing, locale, and streams.
- Containers, describing enhancements to the Containers library, including a new facility for bounded containers that does not require dynamic storage management, more elegant mechanisms for element access and iteration, support for multiway trees, a more general sorting facility, and queues that can be manipulated in a well-defined fashion by multiple tasks.

"John Barnes has the rare ability to take complex material, distill it down to its essence, and explain it in an understandable and often entertaining manner," said Ben Brosgol, ARA President. "Ada 2012 has advanced the state of the art in language design, and the new Rationale will help developers understand and appreciate the language's innovations."

"To encourage Ada 2012's adoption, educational material needs to be widely and easily accessible," said Tullio Vardanega, Ada-Europe President. "The Ada 2012 Rationale is an excellent training resource, and we hope that both students and professional developers will take advantage of its free availability."

[…]

Ada Inside

SparForte

From: Ken Burtch <koburtch@gmail.com>
Date: Sun, 8 Sep 2013 07:15:16 -0700
Subject: ANN: SparForte 1.4
Newsgroups: comp.lang.ada

SparForte 1.4

Type: Programming Language

Platforms: Linux i386/x86_64/Pi and FreeBSD

License: GPL

Home URL: http://www.sparforte.com

NEW

SparForte is an Ada-based command shell, template engine and scripting language. It natively interprets Bourne shell commands and basic database commands at the command prompt and has an integrated debugger. There are 23 built-in packages including MySQL, PostgreSQL, CGI and Memcache and over 80 example scripts.

Major Features in Version 1.4

- software lifecycle awareness
- unused identifier and style checks
- simple exception handling
- database fixes/improvements
- sound support switched to GStreamer
- pragma blocks and chains

See ChangeLog in sources for a complete list.

See http://www.pegasoft.ca/coder/coder_august_2013.html blog for a more detailed overview.

Started in 2001 as the Business Shell, SparForte is an open project being supported in my spare time. If you enjoy SparForte and find it useful, contact me

and I'll continue to support it. Volunteers are welcome to contribute.

Email me at my pegasoft account if you find errors. However, I'm working on a second university degree at night so please be patient as it may take some time to provide fixes.

[See also "SparForte", AUJ 33-2, p. 85. —sparre]

Job: Proprietary Trading

From: Duncan Sands <duncan.sands@deepbluecap.com>
Date: Mon, 30 Sep 2013 09:18:39 -0700
Subject: Ada job available in Amsterdam
Newsgroups: comp.lang.ada

DeepBlueCapital is a proprietary trading firm, trading on most of the world's stock markets for its own account. All of our trading is done automatically, by programs executing algorithms our researchers have developed. The trading programs are written in Ada, with a smattering of other languages. We are growing fast and need additional developers in order to keep up with the growth.

We are looking to hire an Ada developer, preferably one with experience of soft real-time and highly reliable systems. Knowledge of finance and trading is not required, but would be nice to have. The main job would be to help our researchers turn their algorithms into reliable and efficient code (this is why you really need to be in Amsterdam, so you and our researchers can work together in the same place). But we would also want you to work on our core code. Some examples of possible tasks are: boosting performance and scalability, improving our testing infrastructure; adding interfaces to new stock exchanges; writing analysis tools and graphical user interfaces. The company is quite small, less than 20 people, so you can easily make an impact.

See http://www.deepbluecap.com/recruitment.html for more.

Ada in Context

Constraint Error When "out" Parameter Has Wrong Discriminant at Call Time

From: Pascal Malaise <pascal.malaise@gmail.com>
Date: Sun, 25 Aug 2013 07:07:39 -0700
Subject: Constraint error when "out" parameter has incorrect initial content
Newsgroups: comp.lang.ada

I am wondering if GNAT is correct when raising Constraint_Error in the following case:

```
procedure Out_Discr is

  type T (B : Boolean := False) is record
    case B is
      when True => null;
      when False => null;
    end case;
  end record;
  subtype Tt is T(True);

  procedure P (V : out Tt) is
  begin
    V := (B => True);
  end P;

  M : T;

begin
  M := (B => False);
  P (M); -- <-- Here
end Out_Discr;
```

>> raised CONSTRAINT_ERROR : out_discr.adb:20 discriminant check failed

Indeed M is not Tt when calling P, but M is "out" parameter. Shouldn't it be overwritten without any constraint check?

From: Christoph Karl Walter Grein <christ-usch.grein@t-online.de>
Date: Sun, 25 Aug 2013 07:31:11 -0700
Subject: Re: Constraint error when "out" parameter has incorrect initial content
Newsgroups: comp.lang.ada

RM 6.4.1(16) A formal parameter of mode in out or out with discriminants is constrained if either its nominal subtype or the actual parameter is constrained.

Thus the compiler may assume that variables supplied to calls already have the correct subtype.

Web-based User Interfaces

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed, 28 Aug 2013 11:35:36 +0200
Subject: Web-UI for Ada applications (Was: Object Pascal vs Ada -- which is better for a hobbyist?)
Newsgroups: comp.lang.ada

> […] browser is the future […]

The problem with this is that you both add an extra "OS" layer (the browser) _and_ require GUI operations to be interpreted (to some extent) rather than executed as binary code.

> […] application consists of user entering some data, then performing different calculations (using 3rd party C lib as well) and render that data graphically on the screen.
>
> After data is rendered, user should be able to easily change some parameters to refine rendered data as well as do further calculations along with some other simulation going in 'real' time.
>
> Do you have any hint how to do it in browser?

Use Dart (or some other event-oriented language) for the processing in the browser.

Make the calculation (and rendering?) server available through HTTP, pushing results over a websocket.

This way the users can push tasks from the UI to the server, and the UI will receive the results (activating an event handler) once they are available for display.

We are developing an application with a structure similar to this at AdaHeads.

From: Simon Clubley <clubley@eisner.decus.org>
Date: Wed, 28 Aug 2013 11:32:43 +0000
Subject: Re: Web-UI for Ada applications
Newsgroups: comp.lang.ada

> I'm not sure I get the 2nd part? [interpreted vs. binary code —sparre]

When a program which uses, say, the GTK toolkit wants to draw a text box on the screen, it does it by making a direct subroutine call to the GTK function from the program itself.

OTOH, if you use HTML input to render a text box in the browser, then the HTML code is treated as source code input to the browser's rendering engine and needs to first be translated to a internal format before the rendering engine can process the HTML code.

IOW, every time the rendering engine reads the HTML input, it needs to treat it in the same way as, say, a Python or bash interpreter would treat it's input.

Given the high level nature of pure HTML itself, this should not be too great a overhead for pure HTML code. However, this can change if the input also contains a scripting language (such as Javascript) section as well as the HTML code.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed, 28 Aug 2013 16:08:38 +0200
Subject: Re: Web-UI for Ada applications
Newsgroups: comp.lang.ada

> Is using Dart/JS less evil than 3rd party GUI lib via Ada bindings?

JS is definitely more evil.

I'm not quite sure if Dart or bindings to a non-Ada GUI library are preferable in general.

> I was already told here about that option and using e.g. ZeroMQ or something…

I don't know if ZeroMQ talks HTTP and web-sockets. We use AWS for that purpose.

[…]

Style and Optimisation

From: kennethesills@gmail.com
Date: Tue, 1 Oct 2013 19:58:40 -0700
Subject: Optimizing Ada
Newsgroups: comp.lang.ada

Just started using Ada. Two code samples are on the bottom, one is Ada, one is Rust. The functions simply return a map of words and how often they are used in a given string. The Ada one is far slower even though almost all other test cases are faster, and I wanted to know how I might be able to optimize it a bit.

And for those who want a bit more detail:

I decided to give Ada a spin recently and started working on Martyr's Mega Project List to learn the language. I use this to judge the languages performance, readability, etc. And have implemented in quite a few languages.

I've also recently been experimenting with Rust, which (if you don't already know) is another safety-oriented language, but far newer and some bit more C++-ish. So simultaneously, I've been working on the Rust version.

So far, Ada has been more verbose (by far), but the tooling (compiler and GPS), but also comes out easier for me to read looking back at the code. The only real snags I've hit with Ada is trouble with finding documentation on the standard library.

The performance Ada has been putting out is actually very impressive, as well. Consistently demolishing Rust's performance, and sometimes even C++:

Count_Vowels:

Rust => 19 ns

Ada => .5 ns (-O2 actually optimizes the benchmark away entirely).

Reverse String:

Rust => 111 ns

Ada => 29.3 ns

Is Palindrome:

Rust => 119 ns (Rust version is actually case sensitive, so technically “broken”.)

Ada => 56.1 ns

To Pig Latin (Best/Worst/Middle):

Rust => 91 - 141 - 140 ns

Ada => 34 - 44 - 35 ns

I just finished implementing the Count Word Use function (pretty self-explanatory - makes a map of how often words are used) for both languages, and expected the same result in terms of performance. However, the actuals were completely opposite to my expectations:

Count Word Use:

Rust => 2763 ns (Again, case sensitive, so technically it's “broken”.)

Ada => 7436.9 ns

So I was just wondering if anyone could help me optimize my Ada code a bit, as well as tell me how I'm doing in terms of general code style and idiomatic code writing.

Here are the two code samples:

Ada Code:

```
package Word_Count_Maps is new
    Ada.Containers.Indefinite_Hashed_Maps
     (Key_Type        => String,
      Element_Type    => Natural,
      Hash            =>
         Ada.Strings.Hash_Case_Insensitive,
      Equivalent_Keys =>
        Ada.Strings.Equal_Case_Insensitive,
       "="            => "=");

function Count_Words(Original : in String)
return Word_Count_Maps.Map is
  Word_Map : Word_Count_Maps.Map;
  Word_Start : Natural := Original'First;

  procedure Add_Word(Word : in String) is
    use Word_Count_Maps;

    Location  : Cursor;
    Added   : Boolean;

    procedure Increment_By_One(
           Key : in String;
           Occurances : in out Natural) is
    begin
      Occurances := Occurances + 1;
    end Increment_By_One;

  begin
    Word_Map.Insert(Key      => Word,
                    New_Item => 0,
                    Position => Location,
                    Inserted => Added);
    Word_Map.Update_Element(
                   Position => Location,
                   Process  =>
                     Increment_By_One'Access);
  end Add_Word;
  pragma Inline(Add_Word);


  Is_Delimiter : Array(Character) of
Boolean:=
              (ASCII.LF| ' '| '+'| ';'| '/'| '\'| '.'| '!'|
              '$'| '%'| '&'| '*'| '('| ')'| '['| ']'| '{'|
              '}'| ':'| '<'| '>'| '?'| '"'| '|'| '`'| '~'|
              '@'| '#'| '^'| '-'| '_'| '=' | ',' => True,
              Others => False);

begin
  for I in Original'Range loop
    if Is_Delimiter(Original(I)) then
      -- Prevent stacked delimiters from being
      -- considered words.
      if I = Word_Start then
        Word_Start := I + 1;
      else
        Add_Word(Original(
                       Word_Start .. I - 1));
        Word_Start := I + 1;
      end if;
    end if;
  end loop;
  if Word_Start /= Original'Last then
    Add_Word(Original(Word_Start ..
          Original'Last));
  end if;

  return Word_Map;
end Count_Words;
```

Rust Code: http://pastebin.com/1ZVL7Pjf

From: kennethesills@gmail.com
Date: Tue, 1 Oct 2013 20:53:44 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> Ada is the fastest correct implementation you have.

Yes. However, is case sensitivity the reason for a 2.7x slow down? Highly unlikely. In fact, using case-sensitive comparisons in Ada only reduce the time taken by around 50ns. So I just disregarded that fact.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 01 Oct 2013 21:13:59 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

[…]

I agree that the implementation of Indefinite_Hashed_Maps is probably the culprit, but until you have an apples-to-apples comparison, you have no complaint. (Actually, I can't think of any application that could use such a function where the difference would prevent it from meeting reasonable timing requirements, so you probably have no complaint anyway.)

From: kennethesills@gmail.com
Date: Tue, 1 Oct 2013 21:24:43 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

[…] is the code itself clean/idiomatic enough? Not teaching myself any bad habits?

From: Egil Harald Høvik <ehh.public@gmail.com>
Date: Wed, 2 Oct 2013 00:01:29 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> The only real snags I've hit with Ada is trouble with finding documentation on the standard library.

The Standard Library is defined in Annex A of the reference manual,

http://www.adaic.org/resources/add_content/standards/12rm/html/RM-A.html

Specifically, see section A.18.4 for a description of Maps:

http://www.adaic.org/resources/add_content/standards/12rm/html/RM-A-18-4.html

> […]

You could try to preallocate space in the Word_Map by calling Word_Map.Reserve_Capacity(some_fairly_large_number);

Also, you're calling Update_Element even for new words. Depending on your dataset, that could potentially be a lot of unneccessary callbacks. Try this:

```
Word_Map.Insert(Key => Word,
          New_Item => 1,      -- <----
          Position => Location,
          Inserted => Added);
if not Added then
  Word_Map.Update_Element(
          Position => Location,
          Process   =>
              Increment_By_One'Access);
end if;
```

Rust probably have a faster hash function (seems they use SipHash 2-4). Try to implement the same algorithm in Ada, see if that makes a difference.

From: Simon Wright <simon@pushface.org>
Date: Wed, 02 Oct 2013 08:16:02 +0100
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> […]

Looks pretty good to me (we disagree about having a space between the subprogram name and the opening paren, but then the ALRM is wrong about that too :-)

I'm not sure that pragma Inline is meant to work when applied to a subprogram body, so I'd apply it to the spec (I always write specs, because I've set the GNAT style options to standard (-gnaty), and that warns me about missing specs - and about missing spaces, see above).

GNAT may not have implemented your Inline anyway. There's a GNAT pragma Inline_Always, and there are switches (-gnatn, -gnatN I think) that affect inlining. The last time I did a check, inlining made my program slower - cache effects, presumably. That was on powerpc

AdaCore's house rules don't allow I as a variable name (confusion potential), they start at J.

I wonder whether your performance problem is caused by using a function? The internal Word_Map is, I think, going to be copied to the destination and then finalized. Could you use an out parameter? (remembering to clear it before adding ne new content).

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed, 02 Oct 2013 10:11:07 +0200
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> […]

There are some style issues:

- space before parenthesis (i.e. “procedure Increment_By_One (…”).
- lower-case keywords (i.e. “array”, “others”, …).

You may get a performance gain by using an extended return statement:

```
  return Word_Map :
Word_Count_Maps.Map do
    …
  end return;
```

From: kennethesills@gmail.com
Date: Wed, 2 Oct 2013 07:24:21 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> You may get a performance gain by using an extended return statement:

That was exactly what I needed, apparently; Cut the run-time in half.

From: kennethesills@gmail.com
Date: Wed, 2 Oct 2013 07:43:15 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> I wonder whether your performance problem is caused by using a function? […]

That was exactly it, actually. As per Jacob's suggestion - I used an extended return and it dropped down to 2900ns.

Another suggestion was that Rust uses a faster hash function (SipHash 2-4).

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 02 Oct 2013 09:41:38 -0700
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> […]

The code reads fine. There are some stylistic differences from how I'd write it, but that's true for everyone. I haven't seen that version of Insert used very much, so I might add a comment to remind the reader how it works when the key is already in the map. I would probably avoid calling Update_Element when Insert succeeds. Perhaps most important, I would be leery of writing a function that returns a map, but that may be a requirement out of your control

From: John B. Matthews <trashgod@gmail.com>
Date: Wed, 02 Oct 2013 14:58:07 -0400
Subject: Re: Optimizing Ada
Newsgroups: comp.lang.ada

> […]

For comparison with Indefinite_Hashed_Maps, this example [1] uses an instance of Ada.Strings.Bounded.Generic_Bounded_Length as the Key_Type in an instance of Ada.Containers.Hashed_Maps. Your problem domain may suggest a suitable maximum length.

[1] http://home.roadrunner.com/~jbmatthews/jumble.html

Writing “.all” or not

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 18 Oct 2013 21:36:21 -0500
Subject: Re: Dereferencing and style guides
Newsgroups: comp.lang.ada

[…]

But I'd be very unhappy if we required it [writing “.all” —sparre]. The fact that it is *not* required is very important to giving the illusion that you can index a container object directly in Ada 2012 (and modify the elements in place). And in general, you don't really want to expose the fact that access types are used in reference objects — the idea is to allow direct modifications of the object — the mechanism used to achieve that should be irrelevant (especially to a reader).

Ergo, you never really want to write “.all”, because if you have to, you're using too many access types in your program. :-)

Exceptions in Predicates

From: Simon Wright <simon@pushface.org>
Date: Thu, 31 Oct 2013 21:52:36 +0000
Subject: Exceptions in (dynamic) predicates
Newsgroups: comp.lang.ada

A StackOverflow answer contains the following code:

```
subtype XYZ is ABC
  with Dynamic_Predicate =>
          ((XYZ.A in Positive) and
          (XYZ.B not in Positive)) or else
           raise Constraint_Error;
```

(actually, the original didn't have the 'else', with unhelpful results :)

I can't see where in the ARM “raise Constraint_Error” can be a (component of a) boolean expression? or is this a GNATism?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 2 Nov 2013 01:23:04 -0500
Subject: Re: Exceptions in (dynamic) predicates
Newsgroups: comp.lang.ada

> […]

>

> http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0022-1.txt

>

> Looks like it's a planned addition to Ada 202x.

Well, actually it's an after-the-fact addition to Ada 2012. (AI12-0022-1 is a Binding Interpretation, not an Amendment 1.) We realized that we needed it at the last meeting before sending out the Standard wording, but we couldn't get the details right at the meeting and decided to look at it later. Within a few weeks after the meeting, we had figured out the appropriate semantics.

The problem is that without it, you can't replace existing natural language text specifications (that is, comments) with preconditions and predicates, because the

exception raised would change. That doesn't seem helpful.

The Ada 2012 Rationale Epilogue discusses this (and the following) — although you'll have to wait until next week for it to be on-line at ada-auth.org.

Note that for a predicate, you really should use the new Predicate_Failure aspect rather than putting the exception in the predicate proper, because otherwise memberships and validity checks would raise the exception instead of returning the appropriate True or False answer. (That took a lot longer to work out, but that's less jarring as aspects can be added at any time and by implementers.)

```
subtype XYZ is ABC
  with Dynamic_Predicate =>
         (XYZ.A in Positive) and
        (XYZ.B not in Positive),
       Predicate_Failure =>
          raise Constraint_Error;
```

See the Rationale Epilogue for a better explanation that I can put here.

Not sure exactly when GNAT will support Predicate_Failure (we only nailed it down at the June meeting), but I'd expect it to be soon.

Use of Fixed Point Types

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Mon, 11 Nov 2013 07:04:12 -0800
Subject: A curiosity about decimal fixed point types…
Newsgroups: comp.lang.ada

Just a silly curiosity about decimal fixed point types, e.g.,

```
type Euro is delta 0.01 digits 20;

-- Would be 20 enough? :-)
```

Are they currently used? Where? Since most of the examples are about money (mine and RM's) included, I guess that they were thought for financial applications, but I am unsure if nowadays financial software uses this type of types or just floating point.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue, 12 Nov 2013 13:31:53 +0100
Subject: Re: A curiosity about decimal fixed point types…
Newsgroups: comp.lang.ada

[…]

> Are they currently used?

Yes.

> Where?

In 182 out of 119_391 Ada source files (including duplicates) on my laptop.

Reduced to unique type declarations, there is:

```
type Decim is delta 0.1 digits 5;
type Duration is delta 0.001 digits 9 >
     range -24.0 * 3600.0 .. 48.0 * 3600.0;
type Duration is delta 1.0 digits 9
     range -24.0 * 3600.0 .. 48.0 * 3600.0;
type Kroner is delta 0.01 digits 10;
type Megabucks is delta 0.01 digits 15;
type My_Float is delta 0.01 digits 10;
type Percent is delta 0.01 digits 5
     range 0.0 .. 100.0;
type Push_Data_Type is delta 0.01
     digits 7;
type Real_Val_To_Print is delta 0.01
     digits 8;
```

> […]

This indicates that they are used for a few more cases than just money (2 of 9 are money types), but it appears that they aren't all that common.

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.

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 20-23 — 12th **Australasian Symposium on Parallel and Distributed Computing** (AusPDC'2014), Auckland, New Zealand. Topics include: multicore systems; GPUs and other forms of special purpose processors; middleware and tools; parallel programming models, languages and compilers; runtime systems; reliability, security, privacy and dependability; applications; 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.

- Jan20-21 — 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.
- ☺ Jan 21 — **Workshop on Programming Languages meets Program Verification** (PLPV'2014). Topics include: research at the intersection of programming languages and program verification; attempts to reduce the burden of program verification by taking advantage of particular semantic or structural properties of the programming language; all aspects, both theoretical and practical, of the integration of programming language and program verification technology. Includes: invited talk on "Programming Languages for High-Assurance Autonomous Vehicles".

♦ February 01 **Ada at the Free and Open-Source Software Developers' European Meeting** (FOSDEM'2014), Brussels, Belgium. FOSDEM 2014 is a two-day event (Sat-Sun 01-02 February). This years' edition includes again an Ada Developer Room, organized by Ada-Belgium in cooperation with Ada-Europe, which will be held on Saturday 1 February.

February 03-07 **Conference on Software Maintenance, Reengineering and Reverse Engineering** (CSMR-WCRE'2014), Antwerp, Belgium. Topics include: development of new and maintainable systems; evolution, migration and reengineering of existing systems; recovering information from software, software engineering documents and systems artifacts; using this information in system renovation and program understanding.

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.

☺ February18-21 SIAM **Conference on Parallel Processing for Scientific Computing** (PP'2014), Portland, Oregon, USA. Deadline for submissions: January 3, 2014 (posters).

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.

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 19-21 Fachtagung Fachbereich "**Sicherheit - Schutz und Zuverlässigkeit**" (Conference on Safety and Security), Vienna, Austria. Contributions in German, but also welcome in English. Topics include: all aspects of IT Security, Safety and Dependability, such as (in German) Fehlertoleranz, insbesondere in verteilten Systemen; Hoch zuverlässige/verfügbare Systeme; Modellierung und Verifikation von Sicherheit; Sicherheit eingebetteter und mobiler Systeme; Sicherheitskritische Systeme; Software/System Testing; Sprachbasierte Sicherheit; Verifikation & Validierung; Verlässliche Echtzeitsysteme; Zertifizierung funktionaler Sicherheit; etc. Deadline for early registration: January 31, 2014.

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.
- March 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.

Mar 29 - Apr 03 26th Annual IEEE **Software Technology Conference** (STC'2014), Long Beach, California, USA. Theme: "Meeting Real World Challenges through Software Technology". Topics include: software

resiliency; software engineering processes, including process improvement and quality management; agile development methods; open source; model-based software engineering; verification, validation, and testing; software risk management; software reliability; cybersecurity issues and approaches for complex systems; software assurance; software engineering competency, education and training; technology transfer between academia and industry; etc.

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 17th **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.

April 07-11 20th **International Conference on Tools and Algorithms for the Construction and Analysis of Systems** (TACAS'2014). Topics include: specification and verification techniques; analytical techniques for real-time systems; analytical techniques for safety, security, or dependability; static and dynamic program analysis; abstraction techniques for modeling and verification; system construction and transformation techniques; tool environments and tool architectures; applications and case studies; etc.

April 07-11 17th **International Conference on Fundamental Approaches to Software Engineering** (FASE'2014). Topics include: software engineering as an engineering discipline; specification, design, and implementation of particular classes of systems (embedded, distributed, ...); software quality (validation and verification of software using theorem proving, model checking, testing, analysis, refinement methods, metrics, ...); model-driven development and model transformation (design and semantics of domain-specific languages, consistency and transformation of models, ...); software evolution (refactoring, reverse and re-engineering, ...); etc.

April 12 11th **International Workshop on Formal Engineering approaches to Software Components and Architectures** (FESCA'2014). Topics include: modelling formalisms, temporal properties and their formal verification, interface compliance and contractual use of components, static and dynamic analysis, industrial case studies and experience reports, etc.

☺ April TBD **Programming Language Approaches to Concurrency and communication-cEntric Software** (PLACES'2014). Topics include: the general area of programming language approaches to concurrency, communication and distribution, such as design and implementation of programming languages with first class support for concurrency and communication; concurrent data types, objects and actors; verification and program analysis methods for concurrent and distributed software; high-level programming abstractions addressing security concerns in concurrent and distributed programming; multi- and many-core programming models, including methods for harnessing GPUs and other accelerators; integration of sequential and concurrent programming techniques; programming language approaches to web services; etc.

April 07-10 23rd **Australasian Software Engineering Conference** (ASWEC'2014), Sydney, Australia. Topics include: dependable and secure computing; domain-specific models and languages, and model driven development; engineering/operating large-scale distributed systems; formal methods; legacy systems, software maintenance and reverse engineering; modularisation techniques; open source software development; programming languages and techniques; quality assurance; real-time and embedded software; software analysis; software architecture, design and patterns; software processes and quality; software risk management; software reuse and product lines; software security, safety and reliability; software verification and validation; standards; etc. Deadline for submissions: January 31, 2014 (doctoral symposium).

April 22-26 13th **International Conference on Modularity** (Modularity'2014), 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: February 2, 2014 (ACM student research competition), February 7, 2014 (demonstrations), March 2, 2014 (posters).

April 23-25 27th **Conference on Software Engineering Education and Training** (CSEET'2014), Klagenfurt, Austria.

Apr 29 - May 01 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.

May 12-16 19th **International Symposium on Formal Methods** (FM'2014), Singapore. Topics include: interdisciplinary formal methods (techniques, tools and experiences demonstrating formal methods in interdisciplinary frameworks); formal methods in practice (industrial applications of formal methods, experience with introducing formal methods in industry, tool usage reports, etc); tools for formal methods (advances in automated verification and model-checking, integration of tools, environments for formal methods, etc); role of formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, method integration, qualitative or quantitative improvements); theoretical foundations (all aspects of theory related to specification, verification, refinement, and static and dynamic analysis). Deadline for submissions: January 16, 2014 (industry track).

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.

☺ May 19-23 28th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2014), Phoenix, Arizona, USA. Topics include: parallel and distributed algorithms, applications of parallel and distributed computing, parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, parallel programming paradigms, programming environments and tools, etc. Deadline for submissions: January 1, 2014 (PhD forum posters).

☺ May 31- Jun 07 36th **International Conference on Software Engineering** (ICSE'2014), Hyderabad, India.

June 03-06 9th **International Federated Conferences on Distributed Computing Techniques** (DisCoTec'2014), Berlin, Germany. Includes the COORDINATION, DAIS, and FORTE conferences.

June 03-06 14th IFIP **International Conference on Distributed Applications and Interoperable Systems** (DAIS'2014). Topics include: all aspects of distributed applications and systems, throughout their lifecycle; design, architecture, implementation and operation of distributed computing systems, their supporting middleware, appropriate software engineering methods and tools, as well as experimental studies and practical reports; language-based approaches; parallelization; domain-specific languages; design patterns and methods; etc. Deadline for submissions: February 1, 2014 (abstracts), February 7, 2014 (papers).

June 09-11 ACM SIGPLAN **Conference on Programming Language Design and Implementation** (PLDI'2014), Edinburgh, UK. Topics include: programming languages, their design, implementation, development, and use; innovative and creative approaches to compile-time and runtime technology, novel language designs and features, and results from implementations; language designs and extensions; static and dynamic analysis of programs; domain-specific languages and tools; type systems and program logics; checking or improving the security or correctness of programs; memory management; parallelism, both implicit and explicit; debugging techniques and tools; etc.

☺ June 12-13 ACM SIGPLAN/SIGBED **Conference on Languages, Compilers, and Tools for Embedded Systems** (LCTES'2014). Topics include: programming language challenges (features to exploit multicore architectures; features for distributed and real-time control embedded systems; language capabilities for specification, composition, and construction of embedded systems; language features and techniques to enhance reliability, verifiability, and security; virtual machines, concurrency, inter-processor synchronization, and memory management; ...); compiler challenges (interaction between embedded architectures, operating systems, and compilers; support for enhanced programmer productivity; support for enhanced debugging, profiling, and exception/interrupt handling; ...); tools for analysis, specification, design, and implementation (distributed real-time control, system integration and testing, run-time system support for embedded systems, support for system security and system-level reliability, ...); etc. Deadline for submissions: January 31, 2014.

June 16-20 26th **International Conference on Advanced Information Systems Engineering** (CAiSE'2014), Thessaloniki, Greece. Theme: "Information Systems Engineering in Times of Crisis". 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: March 17, 2014 (visionary short papers, demo papers, case study reports).

June 23-25 26th ACM **Symposium on Parallelism in Algorithms and Architectures** (SPAA'2014), Prague, Czech Republic. Topics include: parallel and distributed algorithms; multi-core architectures; compilers and tools for concurrent programming; synergy of parallelism in algorithms, programming, and architecture; etc. Deadline for submissions: January 22, 2014 (abstracts), January 25, 2014 (full papers).

♦ June 23-27 19th **International Conference on Reliable Software Technologies - Ada-Europe'2014**, Paris, France. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN. Deadline for submissions: January 19, 2014 (industrial presentations).

☺ July 8-11 26th **Euromicro Conference on Real-Time Systems** (ECRTS 2014), Madrid, Spain. Contributons on all aspects of real-time systems are welcome. These include, but are not limited to: applications, hardware/software co-design, multicore and manycore architectures for real-time and safety, operating systems, run-time environments, software architectures, programming languages and compiler support, component-based approaches, distribution technologies, modelling and formal methods for design and analysis, safety, reliability, security and survivability; mixed critical systems, etc.

July 8 10th **International Workshop on Operating Systems Platforms for Embedded Real-Time Applications** (OSPERT 2014).

July 8 5th **International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems** (WATERS 2014).

July 18-22 26th **International Conference on Computer Aided Verification** (CAV'2014), Vienna, Austria. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems. Deadline for submissions: January 15, 2014 (CAV Award nominations), January 31, 2014 (abstracts), February 7, 2014 (papers).

July 21-25 38th Annual **International Computer Software and Applications Conference** (COMPSAC'2014), Västerås, Sweden. Topics include: software engineering, security and privacy, quality assurance and assessment, embedded and cyber-physical environments, etc. Deadline for submissions: January 13, 2014 (paper abstracts), January 31, 2014 (full papers), March 23, 2014 (workshop papers), April 8, 2014 (fast abstracts, posters, doctoral symposium papers).

☺ Jul 28 - Aug 01 28th **European Conference on Object-Oriented Programming** (ECOOP'2014), Uppsala, Sweden. Topics include: all areas of object technology and related software development technologies, such as concurrent and parallel systems, distributed computing, programming environments, versioning, refactoring, software evolution, language definition and design, language implementation, compiler construction, design methods and design patterns, aspects, components, modularity, program analysis, type systems, specification, verification, security, real-time systems, etc.

August 29-31 9th **International Conference on Software Engineering and Applications** (ICSOFT-EA'2014), Vienna, Austria. Topics include: software integration, software testing and maintenance, model-driven engineering, software quality, software and information security, formal methods, programming languages, parallel and high performance computing, software metrics, agile methodologies, risk management, quality assurance, certification, etc. Deadline for submissions: March 18, 2014 (regular papers), May 21, 2014 (position papers).

September 01-05 12th **International Conference on Software Engineering and Formal Methods** (SEFM'2014), Grenoble, France. Topics include: abstraction and refinement; programming languages, program analysis and type theory; formal methods for real-time, hybrid and embedded systems; formal methods for safety-critical, fault-tolerant and secure systems; software verification and validation; formal aspects of software evolution and maintenance; light-weight and scalable formal methods; tool integration; applications of formal methods, industrial case studies and technology transfer; education and formal methods; etc. Deadline for submissions: March 14, 2014 (abstracts), March 21, 2014 (papers).

♦ Oct 20-22 **ACM SIGAda Annual International Conference on High Integrity Language Technology (HILT'2014)**, Portland, Oregon, USA. Dates are approximate. Sponsored by ACM SIGAda, in cooperation with Ada-Europe and the Ada Resource Association (approvals pending).

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

Preliminary Call for Participation
Ada Developer Room at FOSDEM 2014
1 February 2014, Brussels, Belgium

Organized by Ada-Belgium
in cooperation with Ada-Europe

FOSDEM[1], the Free and Open source Software Developers' European Meeting, is a free two-day annual event organized in Brussels, Belgium, that offers open source communities a place to meet, share ideas and collaborate. It is renowned for being highly developer-oriented and brings together 5000+ "geeks" from all over the world. The 2014 edition takes place on Saturday 1 and Sunday 2 February, 2014.

For the 5th time, Ada-Belgium[2] organizes a series of presentations related to Ada and Free Software in a s.c. Developer Room. The "Ada DevRoom" at FOSDEM 2014 is held on the first day of the event, i.e. on Saturday 1 February. The program offers introductory presentations on the Ada programming language, including features of the new Ada 2012 standard, as well as more specialised presentations on focused topics. We also provide time for discussion and interaction, and organize the by now famous "Adaists dinner" on Saturday evening...

More details are available on the Ada at FOSDEM 2014 web-page, such as the full list with abstracts of presentations, biographies of speakers, and the concrete schedule. For the latest information at any time, contact <Dirk.Craeynest@cs.kuleuven.be>, or see:

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/14/140201-fosdem.html

[1] https://fosdem.org/2014
[2] http://www.cs.kuleuven.be/~dirk/ada-belgium

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 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
Jamie Ayre
AdaCore
ayre@adacore.com

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

Local Chair
Magali Munos
ECE
munos@ece.fr

In cooperation with
ACM SIGAda, SIGBED, SIGPLAN

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

15 December 2013 **(Extended)**	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

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.

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 not 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 the *Exhibition 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.

ACM SIGAda Annual International Conference
High Integrity Language Technology HILT 2014
Preliminary Call for Technical Contributions

Developing and Certifying Critical Software

Portland, Oregon, USA
October 20-22, 2014
(date is approximate)

Sponsored by ACM SIGAda in cooperation with
Ada-Europe and the Ada Resource Association
Contact: SIGAda.HILT2014@acm.org www.sigada.org/conf/hilt2014

SUMMARY

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 2014 will provide a forum for experts from academia/research, industry, and government to present the latest findings in designing, implementing, and using language technology for high integrity software. **We are soliciting technical papers, experience reports, and tutorial proposals on a broad range of relevant topics.**

POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

- New developments in formal methods
- Multicore and high integrity systems
- Object-Oriented Programming in high integrity systems
- High-integrity languages (e.g., SPARK)
- Use of high reliability profiles such as Ravenscar
- Use of language subsets (e.g., MISRA C, MISRA C++)
- Software safety standards (e.g., DO-178B and DO-178C)
- Typed/Proof-Carrying Intermediate Languages
- Contract-based programming (e.g., Ada 2012)
- Model-based development for critical systems
- Specification languages (e.g., Z)
- Annotation languages (e.g., JML)
- Teaching high integrity development
- Case studies of high integrity systems
- Real-time networking/quality of service guarantees
- Analysis, testing, and validation
- Static and dynamic analysis of code
- System Architecture and Design including Service-Oriented Architecture and Agile Development
- Information Assurance
- Security and the Common Criteria / Common Evaluation Methodology
- Architecture design languages (e.g., AADL)
- Fault tolerance and recovery

KINDS OF TECHNICAL CONTRIBUTIONS

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM *Ada Letters*. The Proceedings will be entered into the widely consulted ACM Digital Library accessible online to university campuses, ACM's mare than 100,000 members, and the wider software community.

EXTENDED ABSTRACTS discuss current work for which early submission of a full paper may be premature. If your abstract is accepted, a full paper is required and will appear in the proceedings. Extended abstracts will be double-blind refereed. In 5 pages or less, clearly state the work's contribution, its relationship with previous work (with bibliographic references), results to date, and future directions.

EXPERIENCE REPORTS present timely results and "lessons learned". Submit a 1-2 page description of the project and the key points of interest. Descriptions will be published in the final program or proceedings, but a paper will not be required.

PANEL SESSIONS gather groups of experts on particular topics. Panelists present their views and then exchange views with each other and the audience. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation and, if selected, a subsequent abstract for a 30-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for ACM *Ada Letters*.

WORKSHOPS are focused sessions that allow knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. Workshop proposals, up to 5 pages in length, will be selected based on their applicability to the conference and potential for attracting participants.

TUTORIALS can address a broad spectrum of topics relevant to the conference theme. Submissions will be evaluated based on applicability, suitability for presentation in tutorial format, and presenter's expertise. Tutorial proposals should include the expected level of experience of participants, an abstract or outline, the qualifications of the instructor(s), and the length of the tutorial (half day or full day).

***HOW TO SUBMIT*: Except for Tutorial proposals use www.easychair.org/conferences/?conf=hilt2014**

Submission	*Deadline*	*Use Easy Chair Link Above*
Technical articles, extended abstracts, experience reports, panel session proposals, or workshop proposals	**June 7, 2014**	For more info contact: **Tucker Taft**, Program Chair `taft@adacore.com`
Industrial presentation proposals	**July 3, 2014** (overview)	
Send **Tutorial** proposals to	**June 7, 2014**	**John McCormick**, Tutorials Chair `mccormick@cs.uni.edu`

At least one author is required to register and make a presentation at the conference.

FURTHER INFORMATION

CONFERENCE GRANTS FOR EDUCATORS: The ACM SIGAda Conference Grants program is designed to help educators introduce, strengthen, and expand the use of Ada and related technologies in school, college, and university curricula. The Conference welcomes a grant application from anyone whose goals meet this description. The benefits include full conference registration with proceedings and registration costs for conference tutorials/workshops. Partial travel funding is also available from AdaCore to faculty and students from GNAT Academic Program member institutions, which can be combined with conference grants. For more details visit the conference web site or contact **Prof. Michael B. Feldman** (`MFeldman@gwu.edu`)

OUTSTANDING STUDENT PAPER AWARD: An award will be given to the student author(s) of the paper selected by the program committee as the outstanding student contribution to the conference.

SPONSORS AND EXHIBITORS: Please contact **Greg Gicca** (`gicca@verocel.com`) to learn the benefits of becoming a sponsor and/or exhibitor at HILT 2014.

IMPORTANT INFORMATION FOR NON-US SUBMITTERS: International registrants should be particularly aware and careful about visa requirements, and should plan travel well in advance. Visit the conference website for detailed information pertaining to visas.

ANY QUESTIONS?

Please send email to SIGAda.HILT2014@acm.org or Conference Chair (**Michael Feldman**, mfeldman@gwu.edu), or Program Chair (**Tucker Taft**, taft@adacore.com).

FOR IMMEDIATE RELEASE

Ada 2012 Language Rationale Published

New educational resource by Ada expert John Barnes explains key Ada 2012 concepts

PITTSBURGH, Pa., November 12, 2013 – Today at ACM SIGAda's HILT 2013 (High Integrity Language Technology) Conference, the Ada Resource Association (ARA) and Ada-Europe announced the publication of the Ada 2012 Rationale and its free availability for downloading. Sponsored in part by Ada-Europe, the Ada Resource Association, and AdaCore, the Ada 2012 Rationale was written by longtime Ada authority John Barnes. It summarizes the new Ada 2012 features, shows examples of their use, describes compatibility with earlier versions of the language standard, and explains the reasons behind critical language design decisions. This new Rationale will be a valuable resource for anyone interested in learning the innovations introduced by the Ada 2012 standard.

The Rationale may be downloaded at no cost from www.adaresource.com/rationale-2012/, from www.ada-europe.org/resources/online/, and from www.adacore.com/rationale-2012/. The book may also be purchased through its commercial publisher, Springer, as volume LNCS 8338 in their *Lecture Notes in Computer Science* series, to be released in mid-December 2013.

The Ada 2012 Rationale contains the following chapters:

- *Introduction*, covering the development of Ada 2012 and giving a brief overview of the main changes from Ada 2005.
- *Contracts and Aspects*, describing the contract mechanism, one of the major enhancements in Ada 2012. It explains subprogram preconditions and postconditions, type invariants, and subtype predicates, and also presents the new unifying concept of "aspects".
- *Expressions*, describing the new flexible forms of expressions introduced in Ada 2012. These new forms – conditional expressions, quantified expressions, and expression functions – are especially useful in conjunction with contracts.
- *Structure and Visibility*, describing various improvements including the generalization of parameter modes to functions, additional flexibility with incomplete types, and new forms for "use" clauses and return statements.
- *Tasking and Real-Time*, describing various enhancements including control over task allocation on multiprocessor architectures, improvements to the scheduling mechanisms, and control of budgets with regard to interrupts.

- *Iterators, Pools, etc.,* describing various improvements in a number of general areas in Ada 2012. These include important new features regarding indexing and accessing that simplify iterating over containers, and a subpool facility for additional flexibility in storage management.
- *Predefined Library*, describing a variety of minor improvements in areas including string and character handling, directory processing, locale, and streams.
- *Containers*, describing enhancements to the Containers library, including a new facility for bounded containers that does not require dynamic storage management, more elegant mechanisms for element access and iteration, support for multiway trees, a more general sorting facility, and queues that can be manipulated in a well-defined fashion by multiple tasks.

"John Barnes has the rare ability to take complex material, distill it down to its essence, and explain it in an understandable and often entertaining manner," said Ben Brosgol, ARA President. "Ada 2012 has advanced the state of the art in language design, and the new Rationale will help developers understand and appreciate the language's innovations."

"To encourage Ada 2012's adoption, educational material needs to be widely and easily accessible," said Tullio Vardanega, Ada-Europe President. "The Ada 2012 Rationale is an excellent training resource, and we hope that both students and professional developers will take advantage of its free availability."

About Ada

The Ada programming language was designed for high-integrity systems – critical applications where reliability is essential and where compliance with safety and/or security standards may be required. Its compile-time and run-time checks help detect errors early in the software development life cycle, avoiding vulnerabilities such as buffer overflow that are prevalent in other languages. Ada is an international (ISO) standard, with the latest version (Ada 2012) introducing direct support for contract-based programming among other new features. Ada continues to see a growing usage in safety-certified applications, including commercial aircraft avionics, military systems, air traffic management/control, railroad systems, and medical devices, and in security-sensitive domains such as financial services.

About the Ada Resource Association

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

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of the Ada programming language in academia, research and industry in Europe. Its flagship event is the annual international conference on reliable software technologies, a high-quality technical and scientific event that has been successfully running in the current format since 1996. Ada-Europe has member organizations all over the continent, in Belgium, Denmark, France, Germany, Spain, Sweden, and Switzerland, as well as individual members in many other countries. For information about Ada-Europe, its charter, activities and sponsors, please visit: www.ada-europe.org. Ada-Europe is headquartered in Brussels, Belgium.

Organization Contacts

Ada Resource Association

Ben Brosgol, ARA President
brosgol@adacore.com

Ada-Europe

Tullio Vardanega, Ada-Europe President
tullio.vardanega@math.unipd.it

Press Contacts

Ada Resource Association

Jessie Glockner
Rainier Communications
Tel: +1-508-475-0025 x140
jglockner@rainierco.com
http://twitter.com/JessieGlockner

Ada-Europe

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

Alice in Adaland

Jacob Sparre Andersen

JSA Research & Innovation, Vesterbrogade 148K, 1620 København V, Denmark; Phone: +45 21 49 08 04; E-mail: jacob@jacob-sparre.dk

Abstract

This paper will present a number of examples where using new features in Ada (2012) [1] has made an application [2] more reliable and easier to understand. In addition to this, there will be a short overview of the external Ada libraries which are instrumental in making the creation of the case application – if not a walk in the park, then at least – reasonably easy.

The case application, Alice, is the core component in a hosted telephone reception system. Alice manages where a PBX[1] directs calls when they arrive from the outside and brings the receptionists live information about the organisations being called. A hosted telephone reception service is the core of out-sourced handling of incoming phone calls; figuring out which employee(s) to redirect a call to, receiving messages, directing calls to voice-mail, etcetera. The intended users of Alice are companies specialised in receiving incoming phone calls for lots of (mostly) SME customers.

The customer co-funding the development of Alice considers the complete system mission critical, and intends to use it for a long time. As Alice is interacting with human callers and receptionists, it is treated as a soft real-time system. Alice is implemented in Ada 2012 to the extent it is supported in GNAT GPL 2013 (and relevant for the application). Surrounding Alice are Bob (user interface), Chloe (management interface) and FreeSWITCH (PBX).

Alice is Open Source software.

1 Introduction

This section will present the system the Alice is a part of together with arguments for implementing it in Ada.

Alice, Bob and Chloe form a hosted telephone reception system.

- Alice manages where a PBX directs calls when they arrive from the outside and brings Bob live information about the organisations being called.
- Bob is the user interface seen by the receptionists doing the actual work of talking to the callees, taking messages and figuring out where calls should be directed.
- Chloe is the administration interface seen by the staff setting up receptions for new (and existing) customers.

[1] "Private Branch Exchange", i.e. a telephone exchange.

When we decided to implement Alice in Ada, it was based on a number of factors. The most important of them is probably that all the developers on the initial team like to program in Ada. The technical excuses we have found to back up our personal preferences map to the design goals of Ada as they appear in the LRM:

> . . . three overriding concerns: program reliability and maintenance, programming as a human activity, and efficiency.

The customer co-funding the development considers the complete system mission critical, and as such is it very important that the resulting system is **reliable**. One might argue that writing the system in SPARK or RavenSPARK would make it even more reliable, but since the complete system also depends on external components, there is only a limited value in making the custom components much more reliable than the external ones. We assume that Ada will give us a good trade-off, making the custom components more reliable than the external components, without having to figure out how to work with the restrictions in SPARK or RavenSPARK.

The customer intends to use the developed system for a multiple decades, and we all expect that the requirements for the system will change while it is in operation. **Maintainability** of the system is thus likely to be a significant factor in lowering the total cost of use of the system. Ada is the only programming language we are aware of, where maintainability is an explicit design goal.

The least plausible of our arguments for using Ada is that the system should be viewed as a soft real-time system. The system is interacting with human callers and receptionists which expect responses within a fixed time-frame after which the value of the response falls off. A more fair argument might be that we want the application to be sufficiently fast and that it is documented that Ada compilers can generate quite **efficient** executables [3].

Altogether these technical arguments are a sound reason for implementing Alice in Ada.

As the customer wants the user interface to run in HTML 5 supporting web browsers, we have decided to implement Bob and Chloe in a combination of Dart and HTML.

2 Examples

This section will present some real-life examples using Ada 2012 features and discuss how they improve software reliability and maintainability. These Ada 2012 features will be covered in the following:

```
function Create
  (Title      : in   String;
   Start_At   : in   String;
   End_Points : in   Receptions.
       End_Point_Collection.Map;
   Decision_Trees : in Receptions.
       Decision_Tree_Collection.Map)
  return Instance
  with Pre => (not End_Points.Is_Empty);
```

Figure 1: Dial-plan object constructor. (Lines 30–36 in "receptions-dial_plan.ads" from [4].)

```
procedure Status_Data
  (Instance : in out Object;
   Request : in  AWS.Status.Data)
with Post => Instance.Has_Status_Data;
-- Set the client request data. This makes the
     response object aware of
-- Cookies, Sessions, GET/POST request
     parameters and everything else that
-- the AWS.Status.Data object contains.
```

Figure 2: Adding information to a HTTP response object. (Lines 126–132 in "response.ads" from [2].)

- Pre- and postconditions
- Subtype predicates
- Functions with **out** parameters
- Expression functions
- Set notation
- **for** ... **of** ... **loop** notation

2.1 Preconditions

We use precondition aspects to document and check assumptions and requirements of subprograms.

In figure 1 we see the specification of a function creating a dial-plan object. A dial-plan consists of a title (for identifying the customer) and a number of actions, one of which is the first action to process. The actions are either decision-trees (for processing conditions) or end-points (which the incoming call can be connected to). A dial-plan does not make sense if it contains infinite loops or if it doesn't have at least one end-point. The precondition aspect assures that the dial-plan object will have at least one end-point. The check for loops happens on-the-fly when the dial-plan object is used.

We consider aspects on subprograms equivalent to other parts of subprogram specifications, and as such, we intend to keep checks declared in aspects active in production builds of Alice. At the same time, we acknowledge that some executable checks may be prohibitively costly to perform in a production system. In case we introduce such a check, we will have to give it individual consideration.

2.2 Postconditions

We use postconditions to document and check the promises subprograms make to their callers.

In figure 2 we see the specification of a procedure which copies information from a HTTP request object to a HTTP response object. This information is required for building a full HTTP response and there is a matching precondition on the "Build" function generating the response to be passed to AWS [5] and onwards to the client.

```
subtype Organization_URI is String
  with Dynamic_Predicate => (Organization_URI'
      Length <= 256);
```

Figure 3: Putting an upper limit on the length of a subtype of String. (Lines 25–26 in "model.ads" from [2].)

2.3 Subtype predicates

We use subtype predicates as an extension of the kinds of constraints one can put on a subtype.

In figure 3 we create a bounded length subtype of type String. The bound is introduced to match the constraints in the database backend used for storing customer information. We see this as a cheap alternative to instantiating "Ada.Strings.Bounded"; no package instantiation and no type conversions.

2.4 Functions with out parameters

We have found a critical error in how GNAT-GPL-2013 implements functions with **out** parameters, which means that this feature currently is banned (but not yet completely exorcised) from our software.

Figure 4 shows a source text snippet where we made use of functions with **out** parameters. The intent was to have a function for checking each kind of error state – in some case with an **out** parameter describing the state – each with a matching function generating a descriptive HTTP error message.

2.5 Expression functions

We use expression functions when there is no need to hide the implementation of a function – or maybe even a benefit from publishing the implementation.

Figure 5 shows a primitive operation of an (abstract) interface to a PBX. In a live system the function should return a value from the system clock, just as its specification shows is does. As we may want to work with a simulated PBX – for example when testing dial-plans – we allow actual PBX interfaces to override the "default" implementation.

```
function Bad_Or_Missing_Message return
    Boolean;
function No_Contacts_Selected return
    Boolean;
function Contact_Does_Not_Exist
        (ID : out Contact_In_Organization)
             return Boolean;
function
    Contact_Without_Messaging_Addresses
        (ID : out Contact_In_Organization)
             return Boolean;

function Bad_Or_Missing_Message return AWS.
    Response.Data;
function No_Contacts_Selected return AWS.
    Response.Data;
function Contact_Does_Not_Exist
        (ID : in Contact_In_Organization)
        return AWS.Response.Data;
function
    Contact_Without_Messaging_Addresses
        (ID : in Contact_In_Organization)
        return AWS.Response.Data;
function Message_Sent return AWS.Response.
    Data;
```

```
if Bad_Or_Missing_Message then
  return Bad_Or_Missing_Message;
elsif No_Contacts_Selected then
  return No_Contacts_Selected;
elsif Contact_Does_Not_Exist (ID) then
  return Contact_Does_Not_Exist (ID);
elsif Contact_Without_Messaging_Addresses (
    ID) then
  return
      Contact_Without_Messaging_Addresses
      (ID);
else
  -- Send message and then ...
  return Message_Sent;
end if;
```

Figure 4: Checking for and reporting various parameter errors before sending a message. The first block of source text shows the specifications of the functions called in the second block. (Lines 162–177 and 421–432 in "handlers-message.adb" from revision `adfbe8f943` of [2].)

2.6 Set notation

We use set notation as a readable extension/addition to ranges.

Figure 6 shows two snippets from a phone number normalisation package. In both cases we "save" an **or** operator and manage with a readable list of the possibilities[2].

2.7 for ... of ... loop notation

Whenever the task of a for loop is a matter of processing the individual elements in a collection/array – without having to reference prior or following elements – a **for** ... **of** ... **loop** simplifies how we write and read the loop.

Figure 7 gives a different view of the phone number normalisation package references in the previous example. Each

[2]As a mathematician I wonder if it would improve the readability to require that the sets are surrounded by "curly brackets" ({...}).

```
function Clock (PBX : in Instance) return Ada.
    Calendar.Time is
  (Ada.Calendar.Clock);
```

Figure 5: Public default implementation. The intent is that a PBX interface used for simulations can override the system clock. (Lines 35–36 in "receptions-pbx_interface.ads" of [4].)

```
function Is_Whitespace (Item : in Character)
    return Boolean is
  use Ada.Characters.Latin_1;
begin
  return Item in Space | No_Break_Space | HT;
end Is_Whitespace;
```

```
      elsif First and then C in '+' | '0' .. '9'
          then
```

Figure 6: Set notation examples. (Lines 23–27 and 37 in "phone_numbers.adb" of [2].)

character from the source string is processed in order, and a normalised version of the passed phone number is returned – unless the passed string is deemed not to be a phone number.

3 External libraries

We use a number of external, Open Source libraries in Alice[3].

AWS (Ada Web Server) provides a reasonably complete – and efficient [7] – HTTP server implementation. You just have to add the business logic and data.

We use the GNAT Component Collection [8] (GNATcoll) for accessing SQL databases. One of the nice features of GNATcoll is that it can generate an Ada interface to a database simply by querying the server about the schema[4].

Our external dial-plan language is based on XML. We use XML/Ada [9] to process the XML formatted dial-plans before we convert them to the internal data model.

We use the convenience library Yolk [10] for centralised logging and configuration handling, as well as various utilities on top of AWS and GNATcoll.

Without these libraries it would be practically impossible to create Alice, as the investment would have been prohibitive compared to the expected return.

[3]We have also factored out some potentially reusable parts of Alice in "libdialplan" [4] (dial-plan processing) and "libesl" [6] (FreeSWITCH interface).

[4]GNATcoll also offers an "Object-Relational Mapping" facility, which provides a type-safe database interface. Once that facility is sufficiently mature (i.e. can handle foreign key tuples), we expect to start using it.

```
   for C of Item loop
     if Is_Whitespace (C) then
       null; -- removing it
     elsif First and then C in '+' | '0' .. '9'
         then
       First := False;
       Filled_To := Filled_To + 1;
       Buffer (Filled_To) := C;
     elsif C in '0' .. '9' then
       Filled_To := Filled_To + 1;
       Buffer (Filled_To) := C;
     else
       return Item; -- not a (normal) phone
           number
     end if;
   end loop;
```

Figure 7: Processing the characters of a string in order. (Lines 34–47 and 37 in "phone_numbers.adb" of [2].)

4 Conclusion

Our experience so far is that the new features in Ada2012 features definitely provide improved readability. Especially set notation and the new **for** ... **of** ... **loop** notation shine in this respect.

It looks like functions with **out** parameters and preconditions also are features which can improve how well readers comprehend source text, but we don't have much evidence to support it yet.

Preconditions, postconditions and subtype predicates should improve the reliability of our software. In practice we haven't yet had a case where we have located an error based on a check of one of these kinds, so we are not completely sure about their actual value.

In addition to all the new features, we are of course very happy with the basics of Ada; tasking and strong typing.

- We use tasks to manage logically parallel execution. It may speed up the execution, but that is (generally) not why we do it.
- Strong typing is a useful tool to avoid mixing up different kinds of objects (even when they are non-composite).

All in all we are very happy to be implementing Alice in Ada 2012.

References

[1] ISO/IEC JTC 1/SC 22/WG 9 Ada Rapporteur Group (2012), *Ada Reference Manual – ISO/IEC 8652:2012(E)*, http://www.adaic.org/ada-resources/standards/ada12/.

[2] AdaHeads K/S (2013), "Alice", https://github.com/AdaHeads/Alice.

[3] B. Fulgham et al. (2013), *Computer language benchmarks game*, http://benchmarksgame.alioth.debian.org/u64q/ada.php.

[4] AdaHeads K/S (2013), "libdialplan", https://github.com/AdaHeads/libdialplan.

[5] AdaCore *et al.*, "Ada Web Server'," http://libre.adacore.com/tools/aws/.

[6] AdaHeads K/S (2013), "libesl"', https://github.com/AdaHeads/libesl.

[7] T. Løcke (2011), *Ada Web Server (AWS) vs node.js*, News from Ada in Denmark.

[8] AdaCore et al., "GNAT Component Collection", http://libre.adacore.com/tools/gnat-component-collection/.

[9] AdaCore et al., "XML/Ada"', http://libre.adacore.com/tools/xmlada/.

[10] T. Løcke (2013) et al., "Yolk", https://github.com/ThomasLocke/Yolk.

Overview of the 16th International Real-Time Ada Workshop

17-19 April 2013
Kings Manor, York, England

Contents *

Workshop Session Summaries

- L. M. Pinho, S. Michell and B. Moore, "*Session Summary: Parallel and Multicore Systems"*
- A. Burns and A. Wellings, *"Session Summary: Locking Protocols"*
- T. Vardanega and R. White, *" Session Summary: Improvements to Ada"*
- J. Real and J. A. de la Puente *" Session Summary: Open Issues"*

Program Committee

Mario Aldea Rivas, John Barnes, Ben Brosgol, Alan Burns (Program Chair), 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 (Workshop Chair) and Rod White.

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, Canada
Brad Moore, General Dynamics, Canada
Luis Miguel Pinho, Polytechnic Institute of Porto, Portugal
Juan Antonio de la Puente, Technical University of Madrid, Spain
Jorge Real, Universitat Politècnica de València, Spain
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
Juan Zamorano, Technical University of Madrid, Spain

Sponsors

* The Proceedings of the 16th International Real-Time Ada Workshop are published in the August 2013 issue of ACM Ada Letters.

Session Summary: Parallel and Multicore Systems

Chair: Luís Miguel Pinho

Rapporteur: Stephen Michell and Brad Moore

1 Introduction

The majority of the session was based on the position papers submitted by Michell, Moore and Pinho. An addition paper [1] had also been distributed to the workshop participants as necessary reading to understand the papers submitted to the workshop, as it deals with the general model of fine-grained concurrency proposed by the authors. The second part of the session was mostly based on the position paper submitted by Zamorano and de la Puente.

2 Discussions – Fine-grained Parallelism for Ada

Papers:

- Burns - "Parallel Ada – A Requirement for Ada 2020" [1]
- Michell, Moore and Pinho - "Tasklettes – a Fine-Grained Parallelism for Ada on Multicores" [2]
- Moore, Michell and Pinho – "Parallelism in Ada – General Model and Ravenscar" [3]

Miguel Pinho opened the discussion in the morning, and laid out the format of the discussion.

Alan Burns had proposed to not have a separate discussion on his position paper, saying that it served as a placeholder to generate discussion, but that the material that it covered was also present in the other papers.

Stephen Michell then continued to present the basic model proposed for supporting augmenting Ada to support parallel computation models. The motivation for a parallel solution in Ada is two-fold, in response to changes in computer chip architectures currently available, as well as future directions. The first important change to note, 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. The second factor is related and has to do with how chip manufacturers are responding to practical limits in CPU clock speed, by increasing the number of cores in the computer chip.

The term *Parallelism OPportunity* (POP) was introduced which was invented for the IRTAW papers to represent the locations in the programmers code that are suitable for parallel execution.

The goal for the general model is to allow for POP's to be explicitly identified in the programmer's code. To illustrate the use and need for POP's, the example of a parallel loop was given, which seemed like a good choice given that loops are very prevalent in application code, and that applying a divide and conquer strategy is perhaps easier to understand than perhaps a recursive subprogram example.

There was significant discussion about the first example. Some attendees had the impression that the work being presented only dealt with parallelism of control flow artifacts, such as loops, as illustrated below:

```
for I in 1 .. N
  with parallel, chunk_size => X
loop
  F(i)
end loop;
```

Discussion followed about the wisdom of giving any directive further than *with parallel* for the programmers to control the details of how parallelism is configured, executed and potentially mapped in the 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 behaviour 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. 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, to other possibilities. Underlying memory buses and memory organization, however, mean that there can be orders of magnitude difference in accessing any particular memory location from various cpu's, 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 pretty ingrained in Ada.

The presenter explained some of the terminology associated with parallelism, in particular, the term Reduce, is described as a special subprogram needed to combine results from multiple workers into a single overall result. In addition, the term Identity value is described as a value that

when applied as one of the arguments to the reducer function produces the identical result. This terminology is commonly used in fine-grained parallelism approaches.

It follows that the syntax of the proposal can be implemented entirely through the use of the addition of special parallelism aspects, although one of the other language syntax changes would involve adding the ability to specify aspects on a loop, in order to support parallel loops to connect the programmers code to the backend parallelism model.

The additional controls that could be specified for the fine-grained parallelism were presented, with the idea that defaults were always provided (or selected by the implementation based upon the number of cores and memory layout specifics. Examples of controls that a programmer might wish to specify include:

- the reduction function;
- identity value;
- parallelism strategy (e.g. work-stealing, work-seeking, work-sharing);
- chunk size;
- worker count;
- ceiling priority;
- affinity;
- worker task storage size; and
- task pool size and behaviours (such as dynamic or static).

As an example of a reducing loop, the example of a loop that calculated the sum of integers from 1 to N was given, where Sum is a variable declared in a global scope outside the loop. Ordinarily computing the sum in parallel would cause problems due to concurrent access to the Sum variable, but this can be avoided if each worker computes a local Sum value for each worker task, which is then combined (Reduced) into a single value by the time all the workers have completed their work.

There was significant of 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*, and *Fibre*. 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. No decision was taken, so this summary uses the term tasklette to stay consistent with the workshop papers. The reader is invited to substitute strand or fibre as they choose.

Andy Wellings presented a glimpse of the model of the Multicore Association "Multicore Programming Practices" and in particular the model of differentiated control level parallelism [3] from data level parallelism. Although the presentation started off with discussing data-level parallelism constructs such as parallel loops and parallel recursion, the group felt that the case for control-level parallelism was more important and relevant for discussion in a real time context. Miguel and Steve point out that the proposal is about providing some basic building blocks for parallelism, which included both control-level parallelism and data-level parallelism. Nonetheless, the group expressed interest in focusing on control-level parallelism, which was the subject for the remaining part of the discussion.

Some participants objected to the "bottom up" approach taken by the authors. There was a discussion that programs often take a top-down design of the software (such as an object/method view of the world and disassemble or refine these objects as needed), and that parallelism models should be developed from the application models.

A request was made to discuss the parallelism model without discussing the underlying implementation. This was agreed in general and the rest of the morning's discussion largely stayed away from the underlying implementation model.

A discussion was held about how exceptions in the proposal were handled. The authors agreed that exceptions were not explicitly discussed in their papers, but stated that exceptions could take the following form:

- An exception raised in a tasklette is returned to the tasklette parent and the tasklette ceases to exist
- Any other exception raised in another tasklette would detect that an exception had already been raised in this POP and the tasklette ceases to exist.
- Any tasklettes that have not commenced execution of their portion will not be started, even if the values that they process would have executed in the sequential model.
- When the parent resumes execution from the end of the POP, it does so in an exception handler following the standard Ada model.
- The semantics of parallel exception handling will be different from the sequential model, but it was noted that, in Ada, one cannot rely on any data values being updated in a construct that is the subject of an exception.

A belief was expressed that parallel loop operations seem to be always on an array. This led to a discussion of the characteristics of loop POP's. The most obvious loops that

[3] The literature calls Task-level parallelism, but we use the term control-level parallelism to make it clear that we are talking about the parallelism of control structures, not task-based parallelism.

can be parallelized are *for loops* that span a predetermined (i.e. before the execution of the first pass of the loop) count of iterations. These match very closely with arrays so it is natural to give simple examples over arrays.

It was pointed out that the example of finding out if a number is prime, is an example where a loop could be used in parallel, without any association with an array.

It is also possible to parallelize loops that do not have a known stopping point, but the most efficient parallelization techniques may create a significant amount of execution that must be discarded once the actual exit condition is calculated (i.e. all iterations corresponding to execution beyond the exit condition).

A discussion item was raised on the possibility of having fine grained and coarse grained parallelism within same programming domain? The presenter responded that the model being presented accommodates both simultaneously.

A point of view was given that perhaps all of the needed functionality could be provided through libraries, i.e. no new language syntax. The presenter responded that libraries alone (i.e. with no supporting language syntax) almost always require the programmer to rewrite the algorithm to take advantage of the libraries and that this often makes reading and maintaining the algorithm problematic. The authors also pointed out that the model not only provides what a set of libraries would provide, but also gives the user the ability to plug in or provide the functionality to handle more challenging environments, such as uneven memory systems, real time systems and even hard real time systems.

It was agreed by the workshop that they needed to understand what other languages were doing in this domain. Miguel presented the parallelism proposals for other languages such as C, C++, C#, including the Cilk and Cilk Plus functionality for C and C++, Open MP, Thread Building Blocks (TBB) and a little on ParaSail.

It was noted that Cilk Plus uses a *strict* fork-join model. Strict means that a *Cilk task*[4] cannot jump into the middle of a parallel computation, and no execution can proceed beyond the end of the POP until all tasklettes have completed. Cilk Plus has an explicit Cilk_Sync that should be used before any variables written by the tasklettes are consumed, but that there is an implicit sync before the block or function containing the POP returns. It was explained that the Ada model being proposed contains only implicit synchronizations and that it must occur before the result of the POP is consumed.

One advantage of the Cilk Plus fork-join model is that, if you remove the *Cilk_Spawn*, *Cilk_For* and the *Cilk_Sync* commands, the program executes completely sequentially.

[4] The C++ usage of the term task to refer to what Moore, Pinho and Michell call tasklettes is a source of confusion. Therefore, when talking about the C++ usage, the term Cilk task or a C++ task is used.

For the proposals for Ada, removal of the *with Parallel* aspect results in the normal sequential execution.

Another advantage of the strict fork-join model is that the strands have full visibility into the stack of the task that contains the POP, with the knowledge that the stack frame cannot be finalized until after all strands have finished. Models that use *futures* must create explicit return objects for the POP to deliver results into (likely on the heap) which can then be consumed at the explicit discretion of the programmer.

The issue of functions without side effects was raised, i.e. no **in out** or **out** parameters, no aliased parameters, and no access types passed as parameters, unless there is a mechanism to show that such actuals are not written to during the execution of the *strands*. The issue of pure functions was discussed, but no conclusions were reached.

Discussions were held about whether or not tasklettes should be named entities within Ada. There was interest that explicit algorithms could be created that used such named entities. The presenters explained that there is a clear separation between *concurrency*, which is captured by tasks, and *parallelism*, which is the transformation of the sequential code so that it could be executed by as many execution resources as are needed at the time. After significant discussion, it was agreed that *tasklettes* need not be named entities.

The issue of the language Parasail generated further discussion. Parasail permits all constructs that are not explicitly made sequential to be executed in parallel with other parallel statements or constructs. Loops can be executed in parallel, unless designated *forward* or *reverse*. The workshop considered if

```
for I in reverse 1 .. N
  with parallel loop
   . . .
end loop;
```

meant that the loop must be executed sequentially for Ada. It was noted that, since Ada already had the *reverse* keyword, one could not automatically enforce a rule that all such loops must be sequential. It was also noted that there were viable parallel algorithms for such cases, meaning that the use of such keywords to signify directed sequential behaviour would likely not work.

A discussion was held about whether or not parallel code should be executed explicitly by library routines, such as *Paraffin*. It was pointed out by the presenters that the library mechanism did not provide automatic transformation of POP code. It takes significant rewrite of the sequential code to fit it into the library call mechanism, and the code becomes more fragile, more difficult to read and more difficult to maintain when using libraries. Syntax provided by the presenters, on the other hand, becomes aspects of the POP structures that provide direction to the compiler in how to map the sequential code for parallel execution.

A subtopic of the discussion of tasklettes, was what happens to exceptions that are raised inside of tasklettes. Since tasklettes represent simply a parallel execution of the 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.

Another issue discussed was how much support that compilers can give to programmers in identifying code that cannot be successfully parallelized by the compiler. This could be because of data dependencies between tasklettes, aliasing of parameters, non associativity of operations, etc. It was noted that in other languages that compilers are not required to make such checks, but with Ada's stricter language rules it may be possible to have more language support to at least detect and report parallelization errors.

Real Time Properties of Tasklettes

As the discussion moved towards the real-time aspects of the model, the workshop began to focus on what properties of tasklettes were needed in the semantic model.

Many participants saw tasklettes as exclusively a run-to-completion model, where tasklettes could only execute code to the synchronization point, and should not block, i.e. call barriers, suspension objects, entries, delays or file IO. This notion is at odds with what competing languages are doing, as C++ examples show many tasklettes (tasks in C++) performing HTML-based calls over the internet, which certainly is a blocking operation. It also is at odds with the notion of higher priority tasklettes interrupting lower priority tasklettes. It was noted that one of the reasons why those non-Ada models of parallel computations went other ways than run-to-completion may be due to their missing concurrency in the original language, hence causing the need to address concurrency and parallelism in the same entity space.

There are, however, reasons for wanting a run-to-completion model for tasklettes that is derived from performance considerations of massively parallel machines. Effectively, processors with dozens or hundreds of cores cannot maintain a strict cache coherence between all cores, and although they can construct a model of shared completely shared memory, the reality is that the time to access any given address in the system may vary by orders of magnitude between different cores, and cache flushes may have dramatic adverse effects on neighbouring but independent variables. One way to mitigate such effects is to copy all relevant code and data needed for an algorithm to a worker task (or worker CPU), have it execute the algorithm, then copy back the results when finished.

Another issue supporting the *no blockin*g approach is that such blocking involves the scheduler that manages tasks, but anonymous tasklettes do not have task control blocks, hence may not be schedulable. Even if each tasklette is executed by a worker task as proposed by the presenters, it is an open issue whether the blocking of a tasklette would result in a block of the *carrying worker task*, or if that task or if that worker task would simply pick up another tasklette for execution. There is an obvious impact in analysability, depending what approach is taken.

The issue was not resolved, but there was brief mention made that such blocking behaviour could be selectable by an aspect.

In the same vein, significant discussion was held about what the runtime should return if a call was made to *Current_Task*, or to get or set task attributes within a POP, resulting in tasklettes making such calls. There was some opinion that in such cases, tasklettes should act as if it was the parent task making the call, for example returning the Parent's *Task_ID* for *Current_Task*. Since no polls were taken on these subjects, it remains open.

Another issue discussed was whether or not tasklettes could be aborted. Since tasklettes cannot be named in the program, there is no way to explicitly abort a tasklette.

Nested Parallelism

There were discussions as to whether or not tasklettes could spawn other tasklettes. The issue of recursive subprograms or subprograms being executed by a tasklette and containing a POP shows clearly that tasklettes must be able to spawn more tasklettes.

Explicit Programmer Control

There was a discussion about the need for explicit programmer control of the various factors that impact the performance of parallelism, but also the explicit needs of real-time systems. Some of the issues that programmers may need to control include

1. Data locality
2. Aliasing of data
3. Reuse of already-calculated objects
4. Calculation deadlines of the parent task
5. Derived calculation deadlines of POP's
6. Blocking or non-blocking of tasklettes

Some that implement compilers and runtimes raised the issue that many times programmers try to control an algorithm but often hinder the implementation's ability to manage all of the issues effectively. This is especially true when the same code can be executed on widely varying underlying hardware. The opinion was expressed that programmers should give high-level guidance to implementations on the management issues and leave it to the implementation to perform the actual layout and management.

Those that build real-time systems raised the issue that regulators will not permit them to “trust the implementation”. They work in an environment where they must be able to account for all behaviours produced by the

program and the implementation; hence must be able to specify and control such behaviours.

It was generally agreed that the management of POP's needs to support multiple modes of control. The three identified were:

- The compiler decides everything as much as possible
- The programmer provides general guidance; and
- The programmer provides explicit control of how the POP is implemented.

There was some support that the aspect mechanism provided by the presenters had many of the characteristics needed, and that more discussion of the individual aspects of the proposal was required, but is still considered an open issue.

3. Other parallel architecture issues

Paper: Juan Antonio de la Puente and Juan Zamorano - "On Real Time Partitioned Multicore Systems"

The authors presented their position that there are ways that high criticality systems and low criticality systems can reside on the same system. The implementation requires that all levels of criticality be separated into their own partitions. These partitions are separated from each other in time (partition scheduler), and by memory space (MMU). Individual partitions are scheduled locally.

Various approaches have been used in prototyping such systems. One approach is to place partitions onto virtual cores, and to map the virtual cores using the Hypervisor virtual machine system. Physical processors were statically scheduled, with predictable scheduling within each partition.

The potential difficulties with this approach are memory access contention and maintaining cache coherence.

The presenters have implemented a demonstration system on a single board using an Intel processor and a Leon 32 processor sharing common memory and running Hypervisor. They also analyzed Ada 2012 with respect to mixed criticality systems, and report that there are no new language features needed (beyond those available in Ada 2012) in Ada for such systems.

As systems move to many-core systems, proposals have been made to place a single Ravenscar task on each core and analyzing the system using that paradigm. In high criticality systems, however, there is deep concern that bus contention, memory contention, and cache coherence issues make timing analysis and behaviour reasoning unreliable.

Significant issues remain in designing and implementing such systems. Communication between partitions is a concern, in that safety-related partitions must not rely on data from low criticality partitions, and security-related partitions cannot pass secure data to less secure systems. Similarly, the possibility that the individual MMU's can be compromised, or that shared buses can be overloaded by the low criticality systems are significant concerns. There was also discussion on approaches where high and low criticality code were executed inside the same partition (an example was presented), but it was felt that the correct model should be to separate criticalities in different partitions.

One of the issues raised was if the inter-partition communication model of Ada is appropriate for these types of systems. It was felt that other models (such as publish-subscribe based) could also be interesting. It was agreed that other inter-partition models would be a reasonable future direction for workshop submissions.

These systems are being investigated, but for now multiprocessor mixed criticality multicore systems are not possible. For now all high criticality systems disable all but a single cpu in their systems.

It was recommended that IRTAW follow this thread as it progresses. Of interest is what the aviation community is doing, as well as the automotive industry.

Paper: Pinho, Michell and Moore – "Ada and Many-core Platforms"

Miguel Pinho led the discussion, raising the idea that partitions could also be units of concurrency or parallelism. It was questioned whether the Ada single memory-space / few task model was really capable of describing where technology was moving with thousands of processors, possibly with non-uniform instruction sets, and non-uniform memory structures.

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, but that 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 in a communication. The solutions proposed by the authors were discussed, but no consensus was reached in this session.

4. Conclusions

The following summarize the agreements reached at the workshop about the applicability of fine-grained parallelism to Ada programs.

It would be useful to have a syntax and a semantic model for control-oriented parallelism, and such a model could be based on the notion of an unit of potential parallelism. In such a model:

1. Tasklette need support of some schedulable entity that gains cores for execution.
2. Tasklette/Strand do not have identities and do not have their own existence
3. Any attributes or invocations such as Current_Task could be the Creator task
4. Their executing time is not accounted

5. The underlying entity that executes a tasklette may be a task, but may be other constructs.
6. The creator task should not block but should continue executing, usually by executing one or more tasklettes and execution time accounting is done only for the parent task. This could lead to busy waiting just because of execution time accountancy.
7. The model should be a strict fork-join model. The entity that created tasklettes may needs to wait for their completion. This could be a busy wait to satisfy execution time accounting.
8. In the priority model, tasklettes inherit the priority of the task and may be executed non-preemptively. It was noted that issues associated with processor affinities and dispatching domains must be revisited.
9. Exceptions could be treated in the same way that Cilk is treating them – the first exception is flagged to be raised in the parent and others are discarded. This may create different behaviour from a sequential program.
10. The nominal units for parallelization are:
    - subprogram calls, including in expressions
    - for loops
    - Ada whole operations, such as assignment of aggregates
    - but we need syntax to address conflicts, such as overlapping ranges.

References

[1] S. Michell, B. Moore, L.M. Pinho (2013), *Tasklettes – a Fine-Grained Parallelism for Ada on Multicores*, Ada Europe 2013.

Session Summary: Locking Protocols

Chair: Alan Burns

Rapporteur: Andy Wellings

1 Introduction

The session considered two main issues: the introduction of the deadline floor locking protocol into a future version of Ada and multiprocessor locking policies.

2 The Deadline Floor Protocol

Ada 2005 introduced EDF scheduling across priority bands. A version of Baker's Stack Resource Control Protocol (called the Preemption Level Control Protocol) was also introduced so that ceiling priorities could be used within an EDF context. However, the Preemption Level Control 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.

Alan Burns introduced the protocol and explained its main motivations and features. 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 related 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.

Several issues were raised in the discussion and these are summarised below.

- The impact of release jitter on the correctness of the protocol. Michael Gonzalez Harbour explained that care had to be taken when tasks could be subject to release jitter as this could result in the delayed execution of a shorter deadline tasks that then could preempt a longer deadline tasks while it was active in the protected object. It was, therefore, necessary to use the values of *Deadline — Jitter* for each task rather than its simple deadline. Failure to do this would invalidate the protocol, and mutual exclusion would not be guaranteed by the protocol itself. Hence, for safety it is also necessary to provide a mutex lock to control protected object access. It was noted, that a similar problem occurs with jitter and the priority ceiling protocol. However, there more priority inversion results instead of the breaking of mutual exclusion. It was also noted that it was possible to optimize the lock so that it was a single bit that indicate that the protected object occupied. Any attempt to access an occupied protected object would result in an exception being raised.
- The meaning of an inherited deadline. In a real-time system there are usually consequences that must be managed if a task misses its deadline. With the deadline floor protocol, a task may inherit a deadline, which will be shorter than its application-defined deadline. The workshop discussed the consequences of a task missing its inherited deadline. It was agreed that inherited deadlines were required to control scheduling and missing them had no repercussions for the application tasks. For example, the default floor for a protected object is *Time_Span_First*, and hence it is quite possible that an absolute deadline computed using this floor value is missed. Consequently, the workshop recommended that, similar to priorities, that there should be a notion of *base* and *active* deadline. The programmer would have no visibility of the active deadline of a task. Any application-level deadline detection mechanisms involves its base rather than its active deadline.
- Protected objects shared between EDF-scheduled and priority-scheduled tasks. In order to fit into the Ada framework for scheduling mixed systems, it is necessary to allow some protected objects to have *both* a priority ceiling and a deadline floor. The rules are simple, if the ceiling of the protected object is a FIFO-within priority level, the task's active deadline is not updated while executing within the protected object (i.e. there is no need to have a deadline floor). If the ceiling priority is an EDF-within priority level, the task's active deadline is updated (i.e. it does need a floor). Nested protected object across levels require further consideration.
- Dynamic changes to the base deadline. It was noted that asynchronous changes to the base deadline of a task does not result in the recalculation of any active deadline associated with the task. Also a new optional check could be specified when using `Delay_Until_And_Set_Deadline` to ensure that the new deadline is longer than or equal to now plus the relative deadline of the tasks (as set by the pragma `Relative_Deadline`).
- Deadlines and other inheritance points in Ada. For completeness, the workshop agreed that in principle a server task should run with an active deadline which is the shortest of its own deadline and the deadline of the calling tasks during a rendezvous between two tasks. Similarly, deadline inheritance should occur during task activation.

Following the above discussion, the workshop agreed that the deadline floor protocol would be a useful addition to Ada and that the Preemption Level Control Protocol should be made obsolete. This could be achieved with a new dispatching policy and/or new locking policy.

3 Multiprocessor Issues

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 too immature to warrant suggested language changes at this time. For the Burns-Wellings paper, further experiments and evaluation were needed including a prototype Ada implementation.

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. This was compared to an approach of having library-supported atomic operations on primitive types (e.g. operations on atomic integers). The main advantage of using protected objects was that the application got to define its own atomic regions rather than having pre-defined operations. 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.

Session Summary: Improvements to Ada

Chair: Tullio Vardanega

Rapporteur: Rod White

1 Introduction

This session took place in the afternoon of the 18th April. It was introduced by Tullio who outlined the papers being considered and how they might lead to improvements in the Ada language and some of the potential challenges they posed.

Three papers were discussed in this session:

- Programming Simple Reactive Systems in Ada: Premature Program Termination, Andy Wellings, Alan Burns, A.L.C. Cavalcanti and N.K. Singh
- Execution time timers for interrupt handling, Kristoffer Nyborg Gregertsen and Amund Skavhaug
- Deferred Setting of Scheduling Attributes for Periodic and Sporadic Tasks, Sergio Sáez, Jorge Real and Alfons Crespo.

2 Programming Simple Reactive Systems in Ada

The paper covers the use of Ada to develop simple reactive, deterministic automata, and the issues of termination of non-tasking programs. 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 to be defined thus:

> The environment task should terminate when all of its dependent tasks have terminated, *and the partition has:*
>
> - *No active timers, and*
> - *No handlers attached to interrupts that are serviced by the partition*.

(Proposed changes in italics)

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.

In the case of the active timers there was a consensus that the termination in their presence is probably an incorrect, and possibly unintended, behaviour. The interrupt issue is slightly less clear, it can be addressed by either handlers being attached and detached dynamically, or by permitting statically attached handler to be detached dynamically – possibly a somewhat counter intuitive concept. The paper also recognised that the problem can be overcome within the context of the current language facilities – a kludge is possible: the main procedure can either perform either a delay until Time'last or a wait on a suspension object that is never set true.

Two possible approaches were initially suggested that would solve the problem without impacting backwards compatibility.

- An indication via a pragma (or aspect) that the environment thread was not to terminate, or
- The ability to control termination – for cases where termination is required.

In this discussion only single processor programs were considered, restricting the discussion to task-free programs – inclusion of multiple processors and tasks would add further complexity.

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.

There was some concern over the need to check for outstanding timing events – when and where should this be done? There was another concern regarding the pattern whereby timing events are programmed to give a periodic behaviour; this pattern would never terminate, but explicit cancellation could address this.

It was noted that the problem has its origin in the change to the interrupt handling model that occurred between Ada 83 and Ada 95 – in Ada 83 interrupts were handled directly by tasks – hence there was no problem with interrupt handlers being left attached after the tasks had terminated. This change in the way interrupts are addressed by the language has been one of the biggest issues in the migration of applications from Ada 83 to Ada 95.

The group concluded that this was not a pressing issue given the simple work-arounds that exist and that there was little merit in making language changes in this area.

It was also agreed that the termination issue should be noted in the assessment of concurrency vulnerabilities.

3 Execution time timers for interrupt handling

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.

The proposal was for there to be a timer for each Interrupt_Id but not one for the overall time consumed by all interrupts (Ada 2012 also supports the concept of a single execution time clock for all interrupts). A prototype of such a solution has been implemented in the GNAT compiler for the AVR32 processor.

Whilst the paper viewed this as an extension to Ravenscar it was noted that it would fall outside of Ravenscar as execution time timers were not in the Ravenscar profile owing to the lack of an effective model of use that would fit the spirit of the profile.

It was proposed that the timer type should be a derived type of the task timer, but the group felt that this was inappropriate/incorrect as the as here was a mismatch between the two forms: the task timer contains a Task_Id whilst the interrupt timer required an Interrupt_Id. It was suggested (and agreed) that the best approach would be to define a now root type for timers that could be specialised for the specified of the task and interrupt timers; this approach would then allow for the inclusion of timers for timing events in a similar manner.

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 major concern expressed quite widely was the potential cost/overhead of the feature. The authors explained the advantages of hardware support to provide timers, but this clearly was not going to be a universal solution. There was a concern regarding the overhead of accessing the hardware clock, which for some modern processors is seen as being potentially significant.

Fundamentally the group agreed that the goal must be to retain predictable behaviour.

It was felt that, with the inclusion of the timers for timing events, this was a useful facility that would be of use now; the inclusion of counters was seen as being a useful addition. There was general support for the basic idea if not the detail – given we already have half the facility (clocks for interrupts) it seems sensible to provide this kind of extension.

A number of issues were noted that had to be worked on to give a more coherent solution.

- 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.

4 Deferred Setting of Scheduling Attributes for Periodic and Sporadic Tasks

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 seen as being an issue – 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 Ada code is relatively straightforward:

- A simple extendable type holding the attributes for the task appropriate to its dispatching regime and the execution platform, e.g. priority, affinity for FIFO_Within_Priorities dispatching, and extending to include relative and absolute deadlines where EDF dispatching is used;

- Some helper subprograms to set/get the various attributes in a the local copy of the attributes object; and
- A pair of subprograms to commit/recover the current attributes to the underlying OS.

The OS can be source of much of the problem, and in the case of general purpose operating systems the provision of appropriate OS support for the Ada tasking model, and its semantics and attributes is the hardest part to solve.

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, that where the task becomes suspended, two significant points were raised:

- Firstly, does the suspension take place in the context of the new or the old attributes? This leads to a number of more detailed considerations such as: where the affinity is changed in the attributes is the task suspended on the original, or new processor?
- Secondly: what should the behaviour be for zero and negative delay values? The Ada behaviour is not necessarily the same as that of operating system interfaces such as POSIX where these cases may not result in a dispatching point.

It was agreed that the suspend form of the operation could only be applied to the current task (i.e. itself) and thus the Task_Id parameter was redundant – this principle was not extended to the immediate form of the operation.

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.

There was some concern about where this facility would feature in the ARM. It was agreed that it would be in an Annex, probably Annex D, and that its implementation would have to be all or nothing at the level of the individual child package. Thus it would be an optional feature.

It was noted that the parameters must be scheduling parameters, the "At_Time" field was viewed as being a helper, for the EDF extension the absolute and relative deadlines should be discrete fields in the record.

In summary:

- The possibilities are not well tied down – there is a high degree of operating system dependence in the current proposal.
- The facility is highly dependent on the underlying OS for its support – if the OS does not support the concept of task attributes in a way that is compatible with the Ada model then simply don't support the facility.
- Experimental changes need to be made to the Linux kernel to facilitate the feature – results should be reported at the next IRTAW. (It was felt that it would be easier to make the change to Linux than to get POSIX changed for a feature that is essentially needed for real-time operation – the POSIX real-time community is seen as being less active than that of Linux).
- In terms of the code, the unnecessary references to Task_Id should be removed.
- Attributes must be true scheduling parameters – not "helpers" – thus for EDF dispatching both relative and absolute deadlines should be captured;
- The feature should be developed for inclusion in Annex D.

Session Summary: Open Issues

Chair: Jorge Real

Rapporteur: Juan Antonio de la Puente

1 Introduction

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 basis for the proposal was the position papers by Gregertsen. Related work includes the paper by Gregertsen and Skavhaug [1] on execution-time control mechanisms.

The session started by the chair recalling a statement from a proposal presented at IRTAW-15 [2]:

> *To make it worthwhile to define a new profile, there must be*
>
> - *a clear application need,*
> - *a computational model that reflects this need,*
> - *and an implementation strategy that leads to a run-time footprint significantly smaller*
>
> *than that needed by the full language.*

The above criteria were considered meaningful by the group.

2 An extended Ravenscar profile

Kristoffer Gregertsen presented his proposal of an extended Ravenscar profile with execution-time control mechanisms. The main motivation 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;
- 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 vivid 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.

The discussion went on by considering some specific details of the proposal. Dynamic priorities and asynchronous task control were considered as mechanisms for dealing with faulty tasks. Tullio Vardanega pointed out three possible policies after a deadline overrun:

- the faulty task can be made non-eligible for running;
- if it can still do some useful work, it can be allowed to run at a low priority;
- it can be restarted, or a mode changed can be triggered.

There was consensus that asynchronous task control is a complex issue that can be difficult to implement in a reduced runtime system.

3 Conclusions

The proposal of defining a new profile that adds flexibility and run-time control mechanisms to Ravenscar while keeping a reduced size and complexity seems interesting and the group agrees that it deserves further investigation. Especially asynchronous control and dynamic priorities have to be studied in detail in order to find all the possible implementation issues. Further work is also needed on the definition of useful fault recovery policies.

References

[1] K. N. Gregertsen and A. Skavhaug (2011), *Implementation and usage of the new Ada 2012 execution-time control features,* 15th International Real-Time Ada Workshop, Liébana, Spain.

[2] A. Burns, A. Wellings, and A. H. Malik (2011), *TTF-Ravenscar: A profile to support reliable high-integrity multiprocessor Ada applications*, 15th International Real-Time Ada Workshop, Liébana, Spain.

SPARK 2014 Rationale

Yannick Moy

AdaCore, France

1 Introduction

SPARK has an enviable industrial track record. Over the past 25 years it has been applied worldwide in a range of industrial applications such as civil and military avionics, railway signaling, cryptographic and cross-domain solutions. SPARK 2014 is the next generation of the language. Below we describe some of the major new features, further information can be found on www.spark-2014.org.

- Convergence with Ada2012 Syntax

The latest version of the Ada language now contains contract-based programming constructs as part of the core language: preconditions, postconditions, type invariants and subtype predicates. SPARK 2014 uses the same syntax for contracts, meaning that a program written in Ada 2012 can be verified by the SPARK 2014 verification tools without having to rewrite the contracts. Subprograms in SPARK and in full Ada can now coexist more easily.

Using the Ada 2012 aspect notation, SPARK 2014 strengthens the specification capabilities of the language by the addition of contracts for:

  - Data dependencies
  - Information flows
  - State abstraction
  - Data and behaviour refinement

- Bigger Language Subset

The SPARK 2014 language comprises a much bigger subset of Ada than its predecessors. The only features excluded are those which are not amenable to sound static verification, which principally means access types, function side effects, aliasing, goto's, controlled types and exception handling.

Relative to previous versions of the language, the main additions to SPARK 2014 include:

  - Generic subprograms and packages
  - Discriminated types
  - Types with dynamic bounds
  - Array slicing
  - Array concatenation
  - Recursion
  - Early exit and return statements
  - Computed constants
  - A limited form of raise statements

- Selectable Language Profiles

Previous versions of SPARK embodied a set of restrictions essentially targeted at highly constrained run-time environments. SPARK 2014 provides the user with flexibility to choose their own language profile to suit their application environment: stay with the full language for server-based applications or apply the Strict profile for embedded applications with limited memory or minimal run-time support. Alternatively you can tailor the pre-defined profiles to prohibit particular language features according to project-specific constraints and regulations.

- Executable Contracts

Functional contracts (pre- and postconditions) have a dual purpose in SPARK 2014. As in previous versions of SPARK, they can be used to specify the functional behaviour required from a subprogram, against which its implementation can be statically verified (i.e. pre-compilation) by the proof system that forms part of the toolset. In SPARK 2014, the same contracts can also be compiled and executed, which in practice means that the compiler turns them into run-time assertions. The executable semantics have a number of applications, not only hybrid verification, but also as an aid to the validation and development of the contracts themselves.

- Hybrid Verification

Hybrid Verification is an innovative approach to demonstrating the functional correctness of a program using a combination of automated proof and unit testing. Once the functional behaviour or low-level requirements of a program have been captured as SPARK 2014 contracts, the verification toolset can be applied to automatically prove that the implementation is correct and free from run-time exceptions. Only where verification cannot be completed automatically is it necessary to write unit tests - with the same contracts used to check the correct run-time behaviour of the relevant subprograms.

- Generative Mode for Data Dependencies

When the implementation of a unit is available, the SPARK tools can extract the information flow and data dependencies for those subprograms in the unit. The user has the choice to specify information flow contracts on the code where they must be enforced, but otherwise let the tools generate the missing contracts to allow overall analysis to be completed.

- Formal Container Library

SPARK 2014 excludes data structures based on pointers, because they make formal verification intractable. Instead, users can either hide pointers from client units by making the data structures private, or benefit from the library of formal containers provided with SPARK 2014. These generic containers (vectors, lists, maps, sets) have been specifically designed to facilitate the proof of client units.

2 Contract Cases

Besides the usual expression of a subprogram contract as a pair of a precondition and a postcondition, SPARK 2014 provides a way to express such a contract by cases. A little history helps understanding how we came up with this new feature.

For example, one might specify by cases the work plan of a 15th century castle guard opening the gate to visitors:

```
procedure Open_Gate (V : Visitor) with
 Contract_Cases => (
     Is_Beggar (V)          => Is_Open (No_Gate),
     Is_Serf (V)            => Is_Open (Side_Gate),
     Is_Merchant (V)        => Is_Open (Small_Gate),
     Is_High_Ranking (V)  => Is_Open (Big_Gate));
```

The cases can be read as follows:

- if the visitor is a beggar, then no gate should be opened,
- if the visitor is a serf, then the side gate should be opened,
- if the visitor is a merchant, then the small gate should be opened,
- if the visitor is high-ranking, then the big gate should be opened.

At first sight, it could seem that the above contract can also be expressed as a regular postcondition with an if-expression:

```
procedure Open_Gate (V : Visitor) with
 Postcondition => (
     if Is_Beggar (V) then Is_Open (No_Gate)
     elsif Is_Serf (V)   then Is_Open (Side_Gate),
     elsif Is_Merchant (V) then Is_Open
                                 (Small_Gate),
    elsif Is_High_Ranking (V) then Is_Open
                                 (Big_Gate));
```

But there is a bit more than that, which makes contract cases more than syntactic sugar for a an if-expression, and a little history might help to understand it.

Previous versions of SPARK only had preconditions and postconditions. This was carried to Ada 2012. But SPARK 2014 also draws its inspiration from other specification languages, such as JML[1] and ACSL[2], which define the notion of subprogram behavior.

JML is the main specification language for Java. In JML, the specification of a subprogram is either *lightweight* (made up of a precondition and postcondition), or *heavyweight* (made up of several behaviors). Each behavior corresponds to a separate contract for the method, with its own precondition (introduced by requires) and postcondition (introduced by ensures). For example, the contract given for Open_Gate would be written as follows in JML:

```
/*@ public normal_behavior
  @   requires is_beggar(v);
  @   ensures  is_open(no_gate);
  @ also
  @ public normal_behavior
  @   requires is_serf(v);
  @   ensures  is_open(side_gate);
  @ also
  @ public normal_behavior
  @   requires is_merchant(v);
  @   ensures  is_open(small_gate);
  @ also
  @ public normal_behavior
  @   requires is_high_ranking(v);
  @   ensures  is_open(big_gate);
  @*/
 public void openGate(visitor v);
```

Each of the behaviors given above is independent from the others, as shown by the desugaring process that transforms this contract into the equivalent:

```
/*@ public normal_behavior
  @   requires is_beggar(v) || is_serf(v) ||
  @              is_merchant(v) || is_high_ranking(v);
  @   ensures (\old(is_beggar(v))      ==>
  @                 is_open(no_gate)) &&
  @              (\old(is_serf(v))       ==>
  @                 is_open(side_gate)) &&
  @              (\old(is_merchant(v))    ==>
  @                 is_open(small_gate)) &&
  @              (\old(is_high_ranking(v)) ==>
  @                 is_open(big_gate));
  @*/
 public void openGate(visitor v);
```

Note that the precondition is not directly visible on the original contract, as it is the disjunction of all requires clauses of all behaviors.

Note also that the precondition allows calling openGate to a lord coming to ask for money, which would fit both the descriptions of the beggar and the high-ranking visitor, leaving the poor guard worry that he may be blamed for both leaving the gates closed on a high-ranking visitor, or letting in a beggar.

Both issues have been solved in ACSL, a specification language for C that builds on the lessons from JML. In ACSL, the specification of a function can contain both a plain precondition/postcondition pair, and a set of behaviors. For example, the contract given for Open_Gate would be written as follows in ACSL:

```
/*@ requires is_beggar(v) || is_serf(v) ||
  @              is_merchant(v) || is_high_ranking(v);
  @ behavior beggar:
  @   assumes is_beggar(v);
  @   ensures is_open(no_gate);
  @ behavior serf:
  @   assumes is_serf(v);
  @   ensures is_open(side_gate);
  @ behavior merchant:
  @   assumes is_merchant(v);
  @   ensures is_open(small_gate);
  @ behavior high_ranking:
  @   assumes is_high_ranking(v);
  @   ensures is_open(big_gate);
  @ complete behaviors;
  @ disjoint behaviors;
  @*/
 void open_gate(visitor v);
```

The precondition is now in one place, and the case of a begging high-ranking visitor is ruled out by the annotation "disjoint behaviors", which requires that only one behavior applies at any time. The other annotation "complete behaviors" requires that at least one behavior applies at any time.

For SPARK 2014, we started with a design very close to the one of ACSL, with individual contract cases matching the behaviors of ACSL. So initially, the contract of Open_Gate was written:

```
procedure Open_Gate (V : Visitor) with
  Contract_Case => (Name     => "beggar",
                    Requires => Is_Beggar (V),
                    Ensures  => Is_Open (No_Gate)),
  Contract_Case => (Name     => "serf",
                    Requires => Is_Serf (V),
                    Ensures  => Is_Open (Side_Gate)),
  Contract_Case => (Name     => "merchant",
                    Requires => Is_Merchant (V),
                    Ensures  => Is_Open (Small_Gate)),
  Contract_Case => (Name     => "high-ranking",
                    Requires => Is_High_Ranking (V),
                    Ensures  => Is_Open (Big_Gate));
```

Our discussions on the (now closed) public mailing list of project Hi-Lite revealed that:

1. the above notation is less readable than the equivalent if-expression
2. the property that each execution matches a unique contract case should be the default

The final synthetic syntax was proposed by Tucker Taft, and we added the rule that contract cases in SPARK 2014 are always disjoint and complete. Et voilà!

Since then, I have found it extremely valuable that contract cases are disjoint and complete, both during proof and testing (yes, these properties are checked at run time when compiling with switch -gnata in GNAT). The concise syntax and the additional expressive power make it indeed valuable to use contract cases in many cases!

3 Specification Functions

Specifying a program's behavior is seldom expressible in a satisfiable way without the capability of abstraction provided by function calls. Yet, specification functions must obey specific constraints like absence of side-effects and termination, that have led to different solutions in various specification languages. Here is what we did in SPARK 2014.

Consider a Reset procedure which sets a valid initial state for a variable X of type T. Rather than stating in the postcondition the individual constraints satisfied by every component of X, it is much better to abstract these details away under calls to functions Is_Valid and Is_Initial:

```
procedure Reset (X : in out T) with
  Post => Is_Valid (X) and then Is_Initial (X);
```

If Reset is part of the public API of private type T, this is the only way to define a contract for Reset, as the details of implementation of T are not visible.

The *specification functions* like Is_Valid and Is_Initial that are called in contracts and other annotations (assertions, loop invariants, etc.) must obey specific constraints:

- They must not perform side-effects, like writing to a global variable, which could change the behavior of the program depending on whether annotations are executed or not.
- They must always terminate, so that the contract in which they appear can be logically interpreted.

Both come easily when specifications are not executable, like in SPARK 2005 or the ACSL specification language for C: a logic function cannot have side-effects, and it is defined to always compute a result.

This is not so easy when specifications are executable, and specification functions are the same functions as the ones called in code. For example, the programming language Eiffel recommends that functions used in annotations are free from side-effects, but does not provide means to enforce it. The specification language JML for Java goes further by requiring that specification functions are declared *pure* [1], which indicates that they are free from side-effects and they terminate. So Is_Valid could be declared in JML as follows:

```
/*@ pure @*/ boolean isValid(T x);
```

JML tools check that a pure method only calls other pure methods, which guarantees that a pure method does not have side-effects, but termination is not checked. In other words, a non-terminating implementation of isValid could invalidate all proof results:

```
/*@ pure @*/ boolean isValid(T x) {
  return not isValid(x);  /* does not terminate */
}
```

The Spec# specification language for C# has borrowed from JML the notion of purity for the absence of side-

effects, that can be both declared by the programmer or inferred by the Spec# verifier:

```
[pure] public bool IsValid(T x) {
  return not IsValid(x);  // does not terminate
}
```

Spec# does not consider termination at all though, so the same non-terminating implementation of IsValid as above could invalidate all proof results.

So how does SPARK 2014 compares to all that? For one thing, all functions in SPARK 2014 are free from side-effects, like in previous versions of SPARK. It is as simple as that. If you want a subprogram that modifies a parameter or a global variable, you should define it as a procedure instead of a function:

```
procedure Is_Valid_Log_Result (X : T;
                               Result : out Boolean);
```

A procedure cannot be called in an expression in Ada, hence cannot appear in an annotation. But Ada allows functions that have side-effects, so the formal verification tool GNATprove checks specifically that SPARK 2014 functions cannot have side-effects. Take for example the following implementation of Is_Valid:

```
function Is_Valid (X : T) return Boolean is
  Result : Boolean;
begin
  ...                -- compute result here
  Log := Result;  -- store result in global variable
  return Result;
end Is_Valid;
```

GNATprove issues the following error on this code:

```
p.ads:5:13: function with side-effects is not in SPARK
```

GNATprove does not attempt to prove termination. So, like in Eiffel, JML and Spec#, a non-terminating implementation of Is_Valid could theoretically invalidate all proof results:

```
function Is_Valid (X : T) return Boolean is
     (not Is_Valid (X));
```

In practice, GNATprove uses two mechanisms to limit the extent to which an incorrect specification function invalidates proof results:

1. Non-termination caused by other reasons than recursion (for example, a loop that does not terminate, or raising an exception) do not invalidate proof results. This is obtained by only generating axioms in the proof system for *expression functions* (like Is_Valid above) whose definition is given by a single expression, not for other more complex functions.
2. Only results for subprograms that use directly or indirectly the incorrect specification function can be invalidated. This is obtained by restricting visibility in the proof system to axioms of entities used directly or indirectly in the component being proved.

To completely avoid issues with incorrect specification functions, users can either check manually that specification functions are not recursive, or adopt a general coding standard that forbids recursion completely (like the Recursive_Subprogram rule checked automatically by the coding standard tool GNATcheck).

What we have achieved with these two mechanisms is that GNATprove does not generate global incorrect axioms in the proof system for subtly wrong specification functions. Take for example the following function that returns a number between 0 and its parameter Max:

```
function Pseudo_Random_Value (G : Generator;
     Max: Natural) return Natural with
  Post => Pseudo_Random_Value'Result >= 0 and
     then Pseudo_Random_Value'Result < Max;
```

If we generated an axiom for such a function, it would be something like (in the syntax of the Why intermediate language):

```
function pseudo_random_value (g:generator,
     max:natural): natural

axiom pseudo_random_value_def:
  forall g:generator. forall max:natural.
    pseudo_random_generator g max >= 0 /\
      pseudo_random_generator g max < max
```

Can you spot the problem? If not, take the value 0 for max, and you get

```
pseudo_random_generator g 0 >= 0 /\
pseudo_random_generator g 0 < 0
```

so the value *pseudo_random_generator g 0* is both non-negative and negative, which is a contradiction. If an automatic prover manages to discover such a contradiction, it can then prove anything, even on code that does not use Pseudo_Random_Value. The problem in the original contract for Pseudo_Random_Generator is that it cannot always return a value between 0 (included) and Max (excluded) if Max is of type Natural. So either Max should be of type Positive, or the postcondition should allow returning a value between 0 included and Max included.

GNATprove avoids these problems by not generating such wrong axioms. Instead, callers of the function Pseudo_Random_Generator will get access in their context to the postcondition of the function.

4 Pre-call and Pre-loop Values

Subprogram contracts are commonly presented as special assertions: the *precondition* is an assertion checked at subprogram entry, while the *postcondition* is an assertion checked at subprogram exit. A subtlety not covered by this simplified presentation is that postconditions are really two-state assertions: they assert properties over values at subprogram exit and values at subprogram entry. A special attribute Old is defined in Ada 2012 to support these special assertions. A special attribute Loop_Entry is

defined in SPARK 2014 to support similar special assertions for loops.

Take the very simple example of a procedure Increment:

```
procedure Increment (X : in out Integer) with
  Post => X = X'Old + 1;
```

The postcondition of Increment states that the value of X at subprogram exit, denoted X, is one above the value of X at subprogram entry, denoted X'Old. We're using a special attribute Old in SPARK 2014 (and in Ada 2012) to denote the value of an object at subprogram entry. By using X'Old in the postcondition, we instruct the compiler to create a copy of X at subprogram entry, that can be dynamically tested when exiting the subprogram to check that the postcondition holds.

This special attribute has many equivalent constructs in other languages:

- the special expression old in the Eiffel language
- the special function \old in the JML specification language for Java
- the special function \old in the ACSL specification language for C
- the special function old in the Spec# specification language for C#
- the special function OldValue in CodeContracts for .NET

Like most its counterparts, the Old attribute can only be used in postconditions (and consequence expressions in contract cases, that have the same scope as postconditions). But in Ada 2012, its use was restricted to avoid a common pitfall found in all other languages.

Take the example of a procedure Extract, which copies the value of array A at index J in parameter V, and zeroes out this value in the array, but only if J is in the bounds of A:

```
procedure Extract (A : in out My_Array;
                   J : Integer; V : out Value) with
  Post => (if J in A'Range then V = A(J)'Old
       and A(J) = 0);  --  INCORRECT
```

Clearly, the value of A(J) at subprogram entry is only meaningful if J is in the bounds of A. If we allowed the code above, then a copy of A(J) would be made on entry to subprogram Extract, even when J is out of bounds, which would raise a run-time error. Therefore, use of Old in expressions that are potentially unevaluated (like the then-part in an if-expression, or the right argument of a shortcut boolean expression) is restricted to plain variables: A is allowed, but not A(J). The GNAT compiler issues the following error on the code above:

```
example.ads:5:44: prefix that is potentially unevaluated
must denote an entity
```

The correct way to specify the postcondition in that case is:

```
procedure Extract (A : in out My_Array;
                   J : Integer; V : out Value) with
  Post => (if J in A'Range then V = A'Old(J)
       and A(J) = 0);  --  CORRECT
```

For formal verification with SPARK 2014, the attribute Old is not sufficient: the postcondition is not the only two-state assertion, loop invariants (special assertions used by the formal verification tool to summarize the effect of a loop) have the same property that they need to relate the state of the program before the loop starts, and the state of the program after a given number of loop iterations. The attribute Loop_Entry was added in SPARK 2014 for that purpose.

Take the example of a procedure Increment_N, which calls N times the previous procedure Increment:

```
procedure Increment_N (X : in out Integer;
                       N : Positive) is
begin
  for J in 1 .. N loop
    Increment (X);
    pragma Loop_Invariant (X = X'Loop_Entry + J);
  end loop;
end Increment_N;
```

The loop invariant expresses that the value of X after the J'th iteration is the initial value of X at loop entry, denoted X'Loop_Entry, plus J. With this loop invariant, the formal verification tool GNATprove is able to prove that the contract of Increment_N is fulfilled:

```
procedure Increment_N (X : in out Integer;
                       N : Positive) with
  Post => X = X'Old + N;
```

To avoid similar pitfalls as the one mentioned above for attribute Old, attribute Loop_Entry is similarly restricted in expressions that are potentially unevaluated, and it can only be used in assertions, loop invariants and loop variants in the top-level list of statements in a loop.

Note that Old and Loop_Entry do not apply to any expression like (X + Y), but only to name expressions (in Ada grammar), such as a component selection X.C'Old, a dereference X.all'Old, a call F(X,Y,Z)'Old, etc.

For more details on the use of attributes Old and Loop_Entry, see:

- the definition of attribute Old in Ada Reference Manual [3].
- the definition of attribute Loop_Entry in SPARK 2014 Reference Manual [4].

5 Loop Invariants

Formal verification tools like GNATprove rely on two main inputs from programmers: subprogram contracts (preconditions and postconditions) and loop invariants. While the first ones are easy to understand (based on the "contract" analogy, in which a subprogram and its caller have mutual obligations), the second ones are not so

simple to grasp. This post presents loop invariants and the choices we made in SPARK 2014.

The need is the same though: like calls are "opaque" to the formal verification tool, hence the need for contracts on subprograms, loops are also "opaque" to the formal verification tool, hence the need for loop invariants.

Note that static analysis tools on the contrary do not require either contracts or loop invariants. It is due to the difference in technology between static analysis tools and formal verification tools: the first ones do not require annotations, but they are less powerful, leading to more a posteriori manual work (the review of false positives) if one wants to use them as verification tools instead of simply bug-finding tools.

A loop invariant is a special assertion, expressed with a pragma, that is true at each iteration of the loop. It is executed as a regular assertion, but used differently from assertions by the formal verification tool. For example, here is a function Get_Prime searching for the smallest prime number between Low and High, and the loop invariant giving the range of values of J, and expressing that no integer between Low and the current value J is prime:

```
function Get_Prime (Low, High : Positive)
                    return Natural is
  J : Positive := Low;
begin
  while J <= High loop
    if Is_Prime (J) then
      return J;
    end if;
    pragma Loop_Invariant
      (J in Low .. High
        and
      (for all K in Low .. J => not Is_Prime (K)));
    J := J + 1;
  end loop;
  return 0;
end Get_Prime;
```

The loop invariant states here properties related to the loop index J: because the value of J changes during the loop, the formal verification tool knows about J only the properties that are stated in the loop invariant. On the code above, GNATprove proves the loop invariant in two stages:

- It proves first that the loop invariant is true at the first iteration.
- It proves then that, assuming the loop invariant held at the previous iteration, it still holds at the next iteration.

This strategy looks a lot like the proof by induction that all students learn in school. Here are the two corresponding checks that GNATprove proves:

```
loopinv.adb:33:7: info: loop invariant initialization
proved
loopinv.adb:33:7: info: loop invariant preservation
proved
```

Now, loop invariants in SPARK 2014 are a bit different from the classical "Hoare" loop invariants (invented by C.A.R. Hoare in 1969), as implemented in Eiffel, JML, ACSL or Spec#.

In all these languages, the loop invariant must be true when reaching a loop, each time the loop resumes, and at loop end. If we had adopted this style of loop invariants in SPARK 2014, the code above would have to be written:

```
function Get_Prime (Low, High : Positive)
                    return Natural is
  J : Positive := Low;
begin
  while J <= High loop
    pragma Loop_Invariant
      ((if Low <= High then J in Low .. High + 1)
        and
      (for all K in Low .. J - 1 => not Is_Prime (K)));
    if Is_Prime (J) then
      return J;
    end if;
    J := J + 1;
  end loop;
  return 0;
end Get_Prime;
```

You can see immediately that the loop invariant gets more complex, because:

- Low might be greater than High, hence the guard "if Low <= High" before stating the range of J, for the loop invariant to hold when reaching the loop
- J may end up being greater than High by 1, hence the range for J "Low .. High + 1", to account for the possible highest value at loop end
- J has been increased before resuming the loop, hence the range for J from Low to J - 1, due to the fact the loop invariant is checked at the beginning of an iteration.

Hence the decision in SPARK 2014 to allow loop invariants anywhere in the loop, so that the user can put it where it is most natural to express. The loop invariant thus needs not hold when reaching the loop, if the loop is never entered, nor does it need to hold when exiting the loop.

In case you wonder if the loop invariant given previously is useful, it allows you to prove the following contract automatically with GNATprove (expressed with contract cases):

```
function Get_Prime (Low, High : Positive) return
    Natural with Contract_Cases =>
  -- case 1: there is a prime between Low and High

  ((for some J in Low .. High => Is_Prime (J)) =>
    -- the smallest prime greater or equal to Low is
    -- returned
```

```
      Get_Prime'Result in Low .. High and
      Is_Prime (Get_Prime'Result)  and
      (for all J in Low .. Get_Prime'Result -1 => not
               Is_Prime (J)),
   -- case 2: there is no prime between Low and High

   (for all J in Low .. High => not Is_Prime (J)) =>
     -- zero is returned
     Get_Prime'Result = 0);
```

6 Loop Variants

Loop variants are the little-known cousins of the loop invariants, used for proving termination of subprograms. Although they may not look very useful at first, they can prove effective as I show with a simple binary search example. And we came up with both an elegant syntax and a slick refinement for loop variants in SPARK 2014, compared to similar constructs in other languages.

I presented previously loop invariants as one of the key annotations (with subprogram contracts) that users should provide for using a tool like GNATprove. What about loop variants? On the one side, they can be omitted, and on the other side, it's up to you to check termination if you do so...

I must confess I've never been a big supporter of loop variants, so I did not care much that they get included in SPARK 2014 or not. SPARK 2005 did not have them and users have never complained about it.

I did not care because:

- Many loops in critical embedded software are "for" loops, for which termination is not an issue (in Ada).
- For most other loops in such software ("while" loops and "plain" loops), termination can be easily checked manually.
- The remaining loops require usually a complex termination argument, that is unlikely to be proved automatically by a tool.

Loop variants made it nonetheless in SPARK 2014, with a rather elegant syntax, and a slick refinement compared to similar constructs in other languages. For example, here is the loop variant that expresses that the scalar quantity J always increases through the loop:

```
pragma Loop_Variant (Increases => J);
```

Because J is a scalar value (an integer or an enumeration), it is bounded by the value of its type, so it cannot increase forever without failing a range check. So, by proving that J always increases through the loop, and that no run-time error occurs in the loop, one can be sure that the loop terminates normally.

In all other languages, the loop variant must always be a decreasing positive integer. We can express it this way in SPARK 2014 too, for example the above is equivalent to:

```
pragma Loop_Variant (Decreases => Type_Of_J'Last
      - J + 1);
```

SPARK 2014 offers the possibility to choose which direction (increasing or decreasing) is most natural, and takes care of comparing with the right bounds. Additionally, more complex loop variants can be expressed with multiple components. A countdown in hours, minutes and seconds can use the following loop variant:

```
pragma Loop_Variant (Decreases => Hours,
                     Decreases => Minutes,
                     Decreases => Seconds);
```

Here, the first component (Hours) should decrease between two consecutive iterations of the loop, or else it stays the same and the second component (Minutes) decreases, or else this one also stays the same and the last component (Seconds) decreases. And one can mix decreasing and increasing directions of variations. Nicer than the alternative:

```
pragma Loop_Variant (Decreases => Hours * 3600 +
      Minutes * 60 + Seconds);
```

(plus in the above, you should also check that the expression does not fail a range check or an overflow check)

So, when is it useful? The typical example is an algorithm that iterates or traverses a collection (an array or a container), and whose termination is not obvious. I found just a few days ago the following test in our test suite where a loop variant was useful. GNATprove proved all checks and assertions on the initial (wrong) implementation of binary search:

```
function Search (A : Ar; I : Integer) return T with
  Pre  => (for all I1 in A'Range =>
             (for all I2 in I1 .. A'Last =>
                A (I1) <= A (I2))),
  Post => (if Search'Result in A'Range then A
     (Search'Result) = I
        else (for all Index in A'Range =>
                A (Index) /= I));

function Search (A : Ar; I : Integer) return T is
   Left  : U;
   Right : U;
   Med   : U;
begin
   Left  := Ar'First;
   Right := Ar'Last;

   if A (Left) > I or else A (Right) < I then
      return 0;
   end if;

   while Left < Right loop
      pragma Loop_Invariant
       ((for all Index in A'First .. Left => A (Index) <= I)
          and then
        (for all Index in Right .. A'Last => I <= A
                (Index)));
```

```
      Med := Left + (Right - Left) / 2;
      if A (Med) <= I then
        Left := Med;
      elsif A (Med) >= I then
        Right := Med;
      else
        return Med;
      end if;
    end loop;

    return 0;
  end Search;
```

Except that I added a very simple loop variant to show that the loop terminates:

```
pragma Loop_Variant (Decreases => Right - Left);
```

and GNATprove could not prove it!

```
binary_search.adb:20:10: warning: loop variant might
fail
```

for a good reason: the loop never terminates when the value searched is in the array! The update to Med is incorrect, and should be written:

```
if A (Med) < I then
   Left := Med + 1;
elsif A (Med) > I then
   Right := Med - 1;
else
   return Med;
end if;
```

which (with an updated loop invariant) leads to a fully proved implementation, including the loop variant!

For more details on loop variants, see the SPARK 2014 Reference Manual [4].

7 Mixing SPARK and Ada Code

The first step before any formal verification work with SPARK is to delimitate the part of the code that will be subject to formal verification (the code in SPARK) within the overall Ada application (which could also contain parts coded in C, in Java, in assembly, etc.). This post presents the solution we've come up with for SPARK 2014.

The possibility of easily linking Ada code with code in other programming languages (C in particular) has been one of the landmark features of Ada since the start, with an Annex of the Ada Reference Manual [3] dedicated to such interfacing. As in many programming languages for embedded applications, Ada also offers the capacity to call directly assembly instructions within the program.

None of these models is suitable for interfacing Ada code with SPARK code:

- SPARK being a subset of Ada, it would be overly restrictive to limit the interface to link-time combination of separate units written fully in Ada or fully in SPARK;
- SPARK being used for formal verification, it would be overly permissive to allow freely mixing of Ada and SPARK code, without clear boundaries.

The solution we've come up with for SPARK 2014 is to let the user define those parts of the code that are in SPARK, using a special aspect or pragma SPARK_Mode. The rest of the code is allowed to use Ada features that are not in SPARK. For example, assume I have a unit with a core service in SPARK, called Compute, and logging and display services in Ada. I can describe this as follows:

```
package Services is
  procedure Compute with SPARK_Mode;
  procedure Log;
  procedure Display;
end Services;
```

I can still call the SPARK and Ada procedures freely from each other, for example:

```
package body Services is
  procedure Compute with SPARK_Mode is
  begin
    -- do something
    Log;
  end Compute;

  procedure Log is ...

  procedure Display is
  begin
    Compute;
    -- display values
  end Display;
end Services;
```

Because procedures in SPARK and in Ada are clearly identified, formal verification can be applied to the first and usual verification based on testing to the second. Combining these results is possible by using subprogram contracts.

What is important to be able to formally analyze Compute above is that the procedure Log has a signature that is compatible with SPARK restrictions, and that it declares in a subprogram contract any constraint for calling it (the precondition) and any effect it has on its environment (the global annotation), although tool GNATprove automatically generates a safe approximation of the global annotation if the user does not give one.

If a unit is mostly in SPARK, it can be marked itself in SPARK, and individual subprograms in the unit can opt out of SPARK, for example:

```
package Services with SPARK_Mode is
  procedure Compute;
  procedure Log with SPARK_Mode => Off;
  procedure Display with SPARK_Mode => Off;
end Services;
```

A spec (subprogram or package) can be in SPARK and not its body, which it typical of features that will be called from SPARK code, but which are not themselves implemented in SPARK. Likewise, a package public part can be in SPARK, but not its private part, which is expressed as follows:

```
package Services with SPARK_Mode is
  -- SPARK interface
private
  pragma SPARK_Mode (Off);
  -- implementation in Ada
end Services;
```

Finally, entities that are neither marked in SPARK or not in SPARK may be used in SPARK code, as far as their declaration does not violate SPARK rules. This greatly facilitates using other units from SPARK code, as the code used needs not be marked with SPARK_Mode aspect or pragma. For example, this is the case for the Ada standard library: many subprograms of the standard library have a declaration compatible with SPARK, but they are not currently marked in SPARK; they can nonetheless be called from SPARK code (for example, to do I/O).

If you want to know more, a brief overview of SPARK_Mode is given in the SPARK 2014 Toolset User's Guide [5].

8 Global State

Global variables are a common source of programming errors: they may fail to be initialized properly, they can be modified in unexpected ways, sequences of modifications may be illegal, etc. SPARK 2014 provides a way to define abstractly the global state of a unit, so that it can be referred to in subprogram specifications. The associated toolset checks correct access to global variables in the implementation.

Global variables can be easily subverted to cater for poor design and quick-and-dirty workarounds, to a point that they are considered as evil in some professional environments. Their extended scope and lifetime is a source of programming errors: as they are sometimes initialized far from their definition, it's easy to forget to initialize them completely; as they are accessible from many points in the program, they can be used to implement conflicting needs; as they may be modified through different subprograms, their correct use may require specific sequences of calls, etc.

SPARK 2014 provides a way to define abstractly the global state of a unit, so that it can be referred to in subprogram specifications. For example, a unit describing an HTML page might have a global variable to hold the content, and another one to hold the CSS style sheet:

```
package HTML_Page with
  Abstract_State => (Content, Style_Sheet)
is ...
```

Then, operations over this HTML page can specify whether they read or write the global state of the page.

```
procedure Initialize_Content with
  Global => (Output => Content);

procedure Update_Content (New_Item : Item) with
  Global => (In_Out => Content);

procedure Display_Text with
  Global => (Input => (Content, Style_Sheet));
```

The above states that:

- Initialize_Content should initialize the value of the Content global state, but not read it, and neither read not write the value of the Style_Sheet global state.
- Update_Content may update the value of the Content global state, but neither read nor write the value of the Style_Sheet global state.
- Similarly, Display_Text may read both parts of the global state, but it should not write any.

The benefit of describing global state abstractly thus appears already at the level of the unit spec, as a way to clearly describe interactions between subprograms and global state.

But the real benefit appears when checking that the body of the unit correctly implements its spec. First, each abstract state is refined into a list of concrete variables:

```
package body HTML_Page with
  Refined_State =>
       (Content     => (Header, Content_Body, Footer),
        Style_Sheet => (Background, Fonts,
                                    Title_Styles))
is
   Header       : HTML_Section;
   Content_Body : HTML_Section;
   Footer       : HTML_Section;
   Background   : Color;
   Fonts        : List_Of_Fonts;
   Title_Styles : List_Of_Styles;
   ...
```

Then, each Global contract of subprogram is expressed with respect to concrete variables, in a Refined_Global contract:

```
procedure Initialize_Content with
  Refined_Global => (Output => (Header,
                                Content_Body, Footer)) is
...

procedure Update_Content (New_Item : Item) with
  Refined_Global => (In_Out => Content_Body) is
...

procedure Display_Text with
  Refined_Global => (Input => (Content_Body, Fonts,
                               Title_Styles)) is
...
```

When the tool GNATprove is applied to this program, it checks that the refined contracts correctly refine the abstract ones (this is the case above), and that the implementation implements the refined contracts. For example, if Update_Content reads Header, contrary to what is specified in its Refined_Global contract, GNATprove issues an error:

```
html_page.ads:8:14: "Header" must be listed in the
Global aspect of "Update_Content"
```

Or if Display_Text does not read the value of Fonts, contrary to what is specified in its Refined_Global contract, GNATprove issues a warning:

```
html_page.adb:26:49: warning: unused initial value of
"Fonts" [unused_initial_value]
```

And that's not all! Thanks to contracts on subprograms, GNATprove can detect any possible read of uninitialized global variables. For example, if Initialize_Content attempts to read the value of Header before initializing it, GNATprove issues an error:

```
html_page.adb:22:20: "Header" is not initialized
[uninitialized]
```

The same is true for client units of HTML_Page, which may also read and write its global state through calls to its API. If a client program calls Update_Content before Initialize_Content, GNATprove issues the error:

```
client.adb:8:04: "Content" is not initialized [uninitialized]
```

Finally, a special contract Initializes can be used to specify that a package initializes some state at elaboration, for example:

```
package HTML_Page with
      Abstract_State => (Content, Style_Sheet),
      Initializes    => Content
is
```

Again, GNATprove will check correct initialization of the concrete variable which refine global state Content here.

In summary, SPARK 2014 allows users to specify correct access to global variables, and the associated tool GNATprove checks that all accesses to global variables are indeed according to the specification.

References

[1] G. T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D. Cok, P. Müller, J. Kiniry, P. Chalin, and D. M. Zimmerman (2011), *JML Reference Manual* (DRAFT).

[2] P. Baudin, P. Cuo, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, V. Prevosto (2013), *ACSL: ANSI/ISO C Specification Language v1.7*.

[3] ISO/IEC 8652:2012(E) (2012), *Ada 2012 Reference Manual.*

[4] AdaCore and Altran UK Ltd (2013), *SPARK 2014 Reference Manual.*

[5] AdaCore and Altran UK Ltd (2013), *SPARK 2014 Toolset User's Guide*.

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 #149: Asserting the truth, but (possibly) not the whole truth

Yannick Moy, AdaCore

Abstract. In Ada 2012, assertions that state desired properties of programs are not limited to pragma Assert. This Gem presents how pragma Assertion_Policy can be used to control which of these assertions should be executed at run time.

Let's get started…

In the beginning was created Ada. It did not have any assertions. Then came GNAT, which introduced pragma Assert. The ARG saw that it was good, and adopted it in Ada 2005. Then came GNAT again, which introduced pragma Precondition and pragma Postcondition. The ARG saw that they were good too, and adopted them as aspects in Ada 2012. The ARG even tried to beat GNAT at this game, and introduced at the same time aspects for type predicates (see Gems #146 and #147) and type invariants (see Gem #148[5]), which are other forms of assertions. Then came GNAT again, introducing pragmas Assume, Assert_And_Cut, and Loop_Invariant, and aspect Contract_Cases, yet other forms of assertions.

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?

GNAT provides a switch -gnata that enables all assertions: pragma Assert of course, but also all the newer forms of assertions presented above. So each unit can be independently compiled with or without assertions. But that's not always adequate.

Let's take the example of writing a library. We want to use preconditions to prevent the library from being called in an invalid context (defensive programming), and postconditions plus type predicates to help with debugging and maintenance of the library (assertion-based verification). Here is the code:

```
package Library is
   type Status is (None, Acquired, Released);

   type Resource is record
      Id   : Integer;
      Stat : Status;
   end record
    with Dynamic_Predicate =>
      (if Resource.Id = 0 then Resource.Stat = None
       else Resource.Stat /= None);

   No_Resource : constant Resource :=
                              Resource'(0, None);

   procedure Get (R : in out Resource; Id : Integer)
  with
      Pre  => R.Stat = None,
      Post => R.Stat = Acquired;

   procedure Free (R : in out Resource) with
      Post => (if R.Stat'Old = Acquired
               then R.Stat = Released);
end Library;

package body Library is
   procedure Get (R : in out Resource; Id : Integer) is
   begin
      R.Stat := Acquired;
      R.Id   := Id;
   end Get;

   procedure Free (R : in out Resource) is
   begin
      if R.Stat /= Acquired then
         return;
      end if;
      R.Stat := Released;
   end Free;
end Library;
```

When this code is compiled with the switch -gnata, each call to Get incurs four run-time assertions (and calls to Free have three):

- a precondition check on subprogram entry
- a postcondition check on subprogram exit
- a predicate check for parameter R on subprogram entry
- a predicate check for parameter R on subprogram exit

That's fine during testing and debugging (when we use -gnata), but we'd like the production code to only contain run-time assertions for the preconditions, to catch misuse of the library in the actual product, while avoiding the overhead of the other checks.

Ada 2012 provides pragma Assertion_Policy for that purpose. This pragma can take the name of an assertion aspect/pragma as first argument, and the desired policy for

[5] Gems #146, #147 and #148 were published in the June 2013 issue of AUJ.

that aspect as second argument. To enforce checking of preconditions even when -gnata is not used, one only has to include the following line at the start of library.ads:

```
pragma Assertion_Policy (Pre => Check);
```

Now, any misuse of the library by client code will be detected, no matter how the library is compiled. Take for example a program that fails to release the resource between two calls to Get:

```
with Library; use Library;
procedure Client is
   R : Resource := No_Resource;
begin
   Get (R, 1);
   Get (R, 2);  -- incorrect
end Client;
```

This code (and the library code) can now be compiled without -gnata:

```
$ gnatmake client.adb
gcc -c client.adb
gcc -c library.adb
gnatbind -x client.ali
gnatlink client.ali
```

And it still raises an error at run time:

```
$ ./client
raised SYSTEM.ASSERTIONS.ASSERT_FAILURE :
failed precondition from library.ads:16
```

For more information on pragma Assertion_Policy, or the new assertion pragmas/aspects supported by GNAT, see the GNAT Pro Reference Manual.

And as Tony Hoare puts it: "Assert early and assert often!.

Gem#151 Specifying Mathematical Properties of Programs

Yannick Moy, AdaCore

Abstract. With the addition of many new kinds of assertions in Ada 2012, it is tempting to state properties of your data that "forget" about the possibility of overflows. GNAT has defined a compilation switch and a pragma that make it possible.

Let's get started…

Integer overflows are exotic and dangerous beasts, that most programmers do not encounter very often, and tend to forget about. An integer overflow occurs when the result of an arithmetic computation does not fit in the machine integer type that needs to hold the result. Of course, Ada requires run-time checks to protect against integer overflows, which are enabled by the switch -gnato in GNAT. But it is common to compile without this switch for production binaries, in which case an integer overflow will result in what the Ada Reference Manual calls "erroneous behavior", which means that anything could happen (see Gems #132 to #135) [6].

[6] Ada Gems #132 and #135 were published in the March 1013 issue of AUJ.

Let's consider a function Max_Payload computing the maximum payload less than a capacity Capacity that can be constructed with two items It1 and It2:

```
package Pack is

   type Payload is new Natural;

   function Max_Payload
     (It1, It2 : Payload;
      Capacity : Payload) return Payload;

end Pack;
```

The implementation of Max_Payload tries to fit the biggest payload first, and then the smallest one:

```
package body Pack is

   function Max_Payload
     (It1, It2 : Payload;
      Capacity : Payload) return Payload
   is
      Result : Payload := 0;
      Small : Payload := Payload'Min (It1, It2);
      Big : Payload := Payload'Max (It1, It2);
   begin
      if Big <= Capacity then
         Result := Big;
      end if;

      if Small <= Capacity - Result then
         Result := Result + Small;
      end if;

      return Result;
   end Max_Payload;

end Pack;
```

Note that the test:

```
if Small <= Capacity - Result then
```

is written this way to avoid integer overflows, while the more natural way of writing this test:

```
if Small + Result <= Capacity then  -- incorrect
```

is vulnerable to integer overflows, if Small + Result is larger than the maximum integer.

While it is expected to write such unnatural expressions in code in order to avoid integer overflows, we would like to write specifications (like subprogram contracts) in a more mathematical way. For example, a natural way to express the postcondition for the function Max_Payload is:

```
function Max_Payload
  (It1, It2 : Payload;
   Capacity : Payload) return Payload;
  with Post =>
    Max_Payload'Result =
      (if It1 + It2 <= Capacity then It1 + It2
       elsif It1 <= Capacity and
             (It1 >= It2 or It2 > Capacity) then It1
```

```
      elsif It2 <= Capacity then It2
      else 0);
```

As contracts are executable in Ada, one can compile them as run-time assertions when passing the switch -gnata to GNAT. (For finer-grain control over execution of assertions, see Gem #149.)

Let's test the above implementation:

```
with Pack; use Pack;

procedure Test_Pack is
begin
  pragma Assert (Max_Payload
                   (1, Payload'Last, 10) = 1);
end Test_Pack;
```

Compiling and running leads to a run-time error, because It1 + It2 does not fit in an integer:

```
$ gnatmake -gnata -gnato test_pack.adb
$ ./test_pack

raised CONSTRAINT_ERROR : pack.ads:10 overflow
check failed
```

Does that mean we cannot write specifications in the most natural way? With GNAT, the answer is no, by using an alternative overflow-checking mechanism for assertions (including subprogram contracts, pragma Assert, etc.)

The idea is to use 64-bit integers (Long_Long_Integer) for arithmetic computations in assertions, to eliminate the possibility of overflow in most cases. This can be achieved either by compiling with the switch -gnato12 or by adding the following pragma in pack.adb or in a configuration file:

```
pragma Overflow_Mode (General => Strict,
                      Assertions => Minimized);
```

Compiling and running now results in no errors:

```
$ gnatmake -gnata -gnato12 -s test_pack.adb
$ ./test_pack
```

Note that GNAT uses 64-bit integers only when they are needed, based on the knowledge of static type bounds. Another mode (Eliminated, also triggered with switch -gnato13) directs the compiler to completely remove the possibility of overflows by using a run-time library of infinite-precision integers. Finally, the alternative overflow modes can also be used for code, as well as assertions, if the user wishes. For more details on overflow modes see the GNAT User's Guide.

PS: Still not sure that the body of Max_Payload implements its contract? As the code above is in SPARK 2014, just use the tool GNATprove to prove it! That's what I did.

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