ADA USER JOURNAL

Volume 39
Number 3
September 2018

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

I would like to start this Editorial by welcoming Kristoffer Nyborg Gregertsen, from SINTEF, Norway, to the Ada User Journal Editorial Team. Kristoffer, who has been a recurrent contributor to the Journal with his work on Ada clocks and timers, is now also contributing in the role of Assistant Editor. The editorial team is thus reinforced, which I am sure will provide our readers with an even better Journal, both in contents as well as in the production process.

Concerning the contents of the September issue, the first article is a report on the work of the Ada Rapporteur Group (ARG), in preparation for the forthcoming revision of the language. This report is, as its first instalment published in the January issue of the Journal, written by Jeff Cousins, now former chair of the ARG.

Afterwards, the issue continues the publication of contributions which originate from the Ada-Europe 2018 conference. First, the reader will find two papers derived from Industrial Presentations of the conference: the first, by Maurizio Martignano from Spazio IT, Italy, describes the pitfalls of tailoring C Guidelines and proposes an approach to properly manage compliance to MISRA and CERT guidelines; the second, from authors from White Elephant GmbH, Switzerland, present the authors' experience on using Ada packages within existing non-Ada embedded systems.

The conference contributions also include two papers derived from technical presentations. In the first, Alejandro Mosteo from the Instituto de Investigación en Ingeniería de Aragón and the Centro Universitario de la Defensa de Zaragoza, Spain, proposes the Alire tool, a library repository manager for Ada open source packages. The second paper, by authors of the US Air Force Academy, USA, provides the final report of the IRONSIDES project, which constructed a provably secure DNS server, using Ada and SPARK.

The final contribution in this issue is an article by Bo Sandén, from the Colorado Technical University, USA, presenting two approaches to design a control application in a multiprocessor context.

The reader will also find in this issue the usual News Digest, Calendar and Forthcoming Events sections, provided by the respective editors. In particular, the forthcoming events section provides information about the program of the upcoming ACM SIGAda High Integrity Language Technology workshop, which will take place 5-6 November, in Boston, USA, co-located with the ACM SPLASH conference, and the call for contributions for Ada-Europe 2019: the 24th International Conference on Reliable Software Technologies, which will take place 10-14 June 2019, in Warsaw, Poland.

Luís Miguel Pinho
Porto
September 2018
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

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

Contents

Ada-related Events

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

Ada-Belgium Spring Event

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Sun, 27 May 2018 18:03:03 -0000
Subject: Ada-Belgium Spring 2018 Event, Sun 10 June 2018
Newsgroups: comp.lang.ada, fr.comp.lang.ada, be.comp.programming

---

Ada-Belgium Spring 2018 Event

Sunday, June 10, 2018, 12:00-19:00

Leuven, Belgium

including at 15:00

2018 Ada-Belgium General Assembly

and at 16:00

Ada Round-Table Discussion

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

---

Announcement

The next Ada-Belgium event will take place on Sunday, June 10, 2018 in Leuven.

For the 11th year in a row, Ada-Belgium organizes their "Spring Event", which starts at noon, runs until 7pm, and includes an informal lunch, the 25th(!) General Assembly of the organization, and a round-table discussion on Ada-related topics the participants would like to bring up.

Schedule

- 12:00 welcome and getting started (please be there!)
- 12:15 informal lunch
- 15:00 Ada-Belgium General Assembly
- 16:00 Ada round-table + informal discussions
- 19:00 end

Participation

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

For practical reasons registration is required. If you would like to attend, please send an email before Wednesday, June 6, 21:00, to Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> with the subject "Ada-Belgium Spring 2018 Event", so you can get precise directions to the place of the meeting. Even if you already responded to the preliminary announcement, please reconfirm your participation ASAP.

If you are interested to join Ada-Belgium, please register by filling out the 2018 membership application form[1] and by paying the appropriate fee before the General Assembly. After payment you will receive a receipt from our treasurer and you are considered a member of the organization for the year 2018 with all member benefits[2]. Early enrolment ensures you receive the full Ada-Belgium membership benefits (including the Ada-Europe indirect membership benefits package).

As mentioned at earlier occasions, we have a limited stock of documentation sets and Ada related CD-ROMs that were distributed at previous events, as well as some back issues of the Ada User Journal[3]. These will be available on a first-come first-serve basis at the General Assembly for current and new members. (Please indicate in the above-mentioned registration e-mail that you're interested, so we can bring enough copies.)

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium/forms/member-form18.html

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

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

Informal lunch

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

General Assembly

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

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

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/18/180610-abga-conv.html

Ada Round-Table Discussion

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

Directions

To permit this more interactive and social format, the event takes place at private premises in Leuven. As instructed above, please inform us by e-mail if you would like to attend, and we'll provide you precise directions to the place of the meeting. Obviously, the number of participants we can accommodate is not unlimited, so don't delay...

Looking forward to meet many of you!

Ada-Europe 2018 in Lisbon

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Tue, 12 Jun 2018 05:43:47 -0000
Subject: Press Release - Reliable Software Technologies, Ada-Europe 2018
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

FINAL Call for Participation

*** UPDATED Program Summary ***

23rd International Conference on Reliable Software Technologies - Ada-Europe 2018

18-22 June 2018, Lisbon, Portugal

http://www.ada-europe.org/conference2018

** Check out tutorials and workshops! **

** Full Program available on conference web site **

*Online proceedings available at event *

*** Register now! ***

Press release:

23rd Ada-Europe Conference on Reliable Software Technologies

International experts meet in Lisbon

Lisbon, Portugal (12 June 2018) - The University Lisboa and Ada-Europe organize from 18 to 22 June 2018 the "23rd International Conference on Reliable Software Technologies - Ada-Europe 2018" in Lisbon, Portugal. The event is organized in cooperation with the Ada Resource Association (ARA), and with ACM's Special Interest Groups on Ada (SIGAda), on Embedded Systems (SIGBED) and on Programming Languages (SIGPLAN).

The Ada-Europe series of conferences has over the years become a leading international forum for providers, practitioners and researchers in reliable software technologies. These events highlight the increased relevance of Ada in general and in safety- and security-critical systems in particular, and provide a unique opportunity for interaction and collaboration between academics and industrial practitioners.

This year's conference offers two days of parallel tutorials and workshops, three keynotes, a full technical program of refereed papers and industrial presentations, an industrial exhibition and vendor presentations, and a social program.

Eight excellent tutorials on Monday and Friday cover a broad range of topics: Recent Developments in SPARK 2014; Scheduling analysis of AADL architecture models; Access types and memory management in Ada 2012; Numerics for the Non-Numerical Analyst; Writing Contracts in Ada; Introduction to Libadalang; Unit-testing with Ahven; Frama-C, a Framework for Analysing C Code.

In addition, on Monday the conference hosts the new workshop on "Runtime Verification and Monitoring Technologies for Embedded Systems" (RUME 2018), and on Friday for the 5th consecutive year the workshop on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering" (DeCPS 2018).

Three eminent keynote speakers have been invited to open each day of the core conference program. Paulo Esteves-Veríssimo (University of Luxembourg, Luxembourg), on "Security and Dependability Challenges of Information Technology (IT) and Operational Technology (OT) Integration". Carl Brandon (Vermont Technical College, USA), on "From Physicist to Rocket Scientist, and how to make a CubeSat that works". Erhard Plödereder (University of Stuttgart, Germany), on "Vulnerabilities in Safety, Security, and Privacy".

The technical program presents 10 refereed and carefully selected technical papers and 4 presentations on the latest research, new tools, applications and industrial practice and experience, a collection of 12 industrial presentations reflecting current practice and challenges, and vendor presentations. Springer Verlag publishes all peer-reviewed papers in the proceedings of the conference, as LNCS Vol. 10873. The remainder of the proceedings will be published in the Ada User Journal, the quarterly magazine of Ada-Europe.

The industrial exhibition opens Tuesday morning in the networking area and runs until the end of Thursday afternoon. Exhibitors include AdaCore, PTC Developer Tools, Rapita Systems, and Ada-Europe.

The social program includes on Tuesday evening a Welcome Reception on board of modern catamaran, to see Lisbon from a different perspective and watch the sunset from the Tagus river. On Wednesday evening will be the traditional Ada-Europe Conference Banquet, held at the restaurant "A Casa do Bacalhau", which means "The House of the Codfish", located in the old stables of the Duke of Lafões palace. Each day, coffee breaks in the exhibition area and sit-down lunches offer ample time for interaction and networking.

The Best Paper Award will be presented during the Conference Banquet, the Best Presentation Award during the Closing session.

The conference is hosted by Univ. Lisboa at the VIP Executive Art's Hotel, strategically located in the Parque das Nações area, Lisbon's modern business centre, close to the Tagus river and the Vasco da Gama bridge, and can easily be accessed by metro.

The full program is available on the conference web site.

Online registration is still possible.

Latest updates:

The 16-page "Final Program" is available at <http://www.ada-europe.org/conference2018/AdaEurope2018%20Final%20Program.pdf>

Check out the 8 tutorials in the PDF program, or in the schedule at <http://www.ada-europe.org/conference2018/tutorials.html>.

Registration fees are very reasonable and the registration process is done on-line. Don't delay! For all details, select "Registration" at <http://www.ada-europe.org/conference2018> or go directly to <http://ae2018.di.fc.ul.pt/registration.html>.

The proceedings, published by Springer Verlag as Lecture Notes in Computer Science Vol. 10873, are already available online. See <https://link.springer.com/book/10.1007/978-3-319-92432-8>. A printed copy is included in every full conference registration.

Help promote the conference by advertising for it! <http://www.ada-europe.org/conference2018/promotion.html>. Put up the poster at <http://www.ada-europe.org/conference2018/picts/AE2018_poster.png>.

Recommended Twitter hashtags: #AdaEurope and/or #AdaEurope2018.

For the latest information consult the conference web site <http://www.ada-europe.org/conference2018>.

[See also “Ada-Europe 2018 in Lisbon”, AUJ 39-2, p. 62. —sparre]

Ada-Europe 2019 in Warsaw

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Fri, 22 Jun 2018 12:05:48 -0000
Subject: Ada-Europe 24th Int'l Conf. on Reliable Software Technologies
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

As announced this week at the Ada-Europe 2018 conference in Lisbon: Ada-Europe 2019 will be in Warsaw, Poland, in the week of 10-14 June.

Info on http://www.ada-europe.org/conference2019 will be expanded shortly, including the Preliminary Call for Contributions. Start planning!

ACM HILT 2018 in Boston

From: S. Tucker Taft, AdaCore
Date: Fri, 29 Jun 2018 10:25:30 -0700
Subject: CFP: ACM HILT 2018 Workshop on Languages/Tools for Cyber-Resilience at SPLASH in Boston, Nov 5&6
Newsgroups: comp.lang.ada

Here is a chance to show how Ada and SPARK can be used to address cyber-security challenges:

https://2018.splashcon.org/track/hilt-2018-papers

HILT 2018: Workshop on Languages and Tools for Ensuring Cyber-Resilience in Critical Software-Intensive Systems, as part of SPLASH 2018, November 5 & 6, 2018, Boston, MA, USA, Sponsored by ACM SIGAda.

The High Integrity Language Technology (HILT) 2018 Workshop is focused on the cyber-resilience needs of critical software systems, where such a system must be trusted to maintain a continual delivery of services, as well as ensuring safety in its operations. Such needs have common goals and shared strategies, tools, and techniques, recognizing the multiple interactions between security and safety.

We encourage papers and extended abstracts relating to:

- Language features that can be used to build security and/or safety into software-intensive systems; Approaches to apply effectively the emerging technologies of AI and Machine Learning in critical software systems;
- Mechanisms that can be used to understand, certify, and manage systems that are “data driven,” relying on “soft code,” where control flow and algorithms are expressed using data rather than “hard code” expressed directly in programming languages;
- Extending contract-based programming to specifying security resistance and resilience properties as well as safety and/or correctness properties;
- Strategies to minimize risk when applying complex software requirements to cyber-physical systems;
- Modeling and/or programming language features and analysis techniques that aid in code analysis and verification and that increase the level of abstraction and expressiveness;
- Language features that support continuous requirements maturation to support evolving needs, particularly in cyber-physical systems, while ensuring that security and safety properties are preserved.

This workshop is designed as a forum for communities of researchers and practitioners from academic, industrial, and governmental settings, to come together, share experiences, and forge partnerships focused on integrating and deploying tool and language combinations to address the challenges of building cyber-resilient software-intensive systems. The workshop will be a combination of presentations and panel discussions, with one or more invited speakers.

Attendees wishing to present at the workshop should prepare full papers (approx. 6-8 pages), or extended abstracts (approx. 2-4 pages) for their proposed presentations, and the workshop program committee will select presentations and organize them into sessions. Other interested participants are welcome to register for the HILT 2018 Workshop as part of their SPLASH 2018 registration.

- Aug 1: Papers or Extended abstracts due;
- Sep 1: Notification of submissions accepted for presentation
- Oct 1: Final submissions due
- Nov 5&6: Workshop as part of SPLASH 2018

Please submit papers and extended abstracts, by Aug 1, 2018, on HotCRP: https://hilt18.hotcrp.com/

Workshop Co-Chairs

- Bill Bail, MITRE
- Tucker Taft, AdaCore, Inc

Organizing Committee

- Dirk Craeynest, ACM SIGAda International Representative, KU Leuven
- Drew Hamilton, Chair, ACM SIGAda, Mississippi State University, CCI
- Clyde Roby, Secretary-Treasurer, ACM SIGAda, Institute for Defence Analyses
- Alok Srivastava, Editor, ACM Ada Letters, Engility Corp.
- Ricky E. Sward, Past Chair, ACM SIGAda, MITRE

URLs:

- SPLASH 2018: http://www.splashcon.org
- HILT 2018 Information: http://sigada.org/conf/hilt2018
- HILT 2018 Submissions: https://hilt18.hotcrp.com/
- ACM SIGAda: http://sigada.org

Ada-related Resources

The Ada-wide Search Engine

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 May 2018 19:53:12 -0500
Subject: Re: Ada in polluted WWW searches
Newsgroups: comp.lang.ada

> [problems searching for Ada]

This problem was well known decades ago. Tom Moran and I built the Ada-wide search engine to "solve" this problem -- it indexes all known Ada sites. It doesn't work as well as it used to because a lot of sites have switched to HTTPS which we can't index yet. (I need to find some time to fix that; I have *soooo* much spare time ;-).

Anyway, you can find it on AdaIC or directly at <http://www.ada-auth.org/wide-search.html>.

Note that the blurb for this on the ACAA home page says "Search many Ada-related web sites without getting results for dentists and dairymen."

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue, 22 May 2018 08:08:52 +0200
Subject: Re: Ada in polluted WWW searches
Newsgroups: comp.lang.ada

> [...]

Is the source for the indexing engine available? Maybe somebody could be tempted to submit a patch, adding support for HTTPS.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 22 May 2018 16:31:38 -0500
Subject: Re: Ada in polluted WWW searches
Newsgroups: comp.lang.ada

> [...] Is the source for the indexing engine available? [...]

Unfortunately, it's not. We built it out of spare parts laying around, and I never found time to figure out the licensing on those. Another job for the copious spare time. :-)

I do plan to work on that project this summer after the Lisbon ARG meeting (it's tied into a long-running project to get the various servers off of Windows, to use more modern hardware and not cost $$$).

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat Jul 7 2018
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn:	2_712 members	[1]
- Reddit:	1_900 readers	[2]
- StackOverflow:	1_000 followers	[3]
- Google+:	771 members	[4]
- Freenode	87 participants	[5]
- Gitter:	57 people	[6]
- Twitter:	8 tweeters	[7]

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

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

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

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

[5] #Ada on irc.freenode.net

[6] https://gitter.im/ada-lang

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

[See also “Ada on Social Media”, AUJ 39-2, p. 63. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat Jul 7 2018

Subject: Repositories of Open Source software

GitHub: 2_123 repositories [1]
513 developers [2]
2_809 issues [3]
Rosetta Code: 645 examples [4]
33 developers [5]
1 issues [6]
Sourceforge: 265 projects [7]
BlackDuck OpenHUB: 206 projects [8]
Bitbucket: 82 repositories [9]
Codelabs: 45 repositories [10]
AdaForge: 8 repositories [11]

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

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

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

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

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

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

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

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

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

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

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

[See also "Repositories of Open Source Software", AUJ 39-2, p. 63. —sparre]

Ada-related Tools

DragonEgg

From: Simon Clubley <clubley@eisner.decus.org>
Date: Mon, 21 May 2018 21:37:12 -0000
Subject: DragonEgg has been revived
Newsgroups: comp.lang.ada

[I apologise if this has already been covered in the walls of text which have been flying around recently, but I have not seen this mentioned yet.]

I've just discovered that DragonEgg has been revived for GCC 8.x and LLVM 6.x, although there is nothing in the announcement about Ada:

http://lists.llvm.org/pipermail/llvm-dev/2017-August/116705.html

[See also <https://github.com/xiangzhai/dragonegg>. —sparre]

State of the Compiler Market

From: Simon Clubley <clubley@eisner.decus.org>
Date: Tue, 22 May 2018 12:29:09 -0000
Subject: Re: DragonEgg has been revived
Newsgroups: comp.lang.ada

> [...]

What I would like to see is an Ada compiler that can generate code for a wide range of targets without any GPL restrictions on the generated code.

I'm not really bothered how that happens but LLVM seems like an interesting option.

The real question however is will this Ada compiler still work with the versions of the toolchains available 2-5 years from now or will it fall into disuse just like DragonEgg did ?

There's a confidence problem here. I can write C and C++ code in 2018 for some random embedded target knowing there's a very very good chance I will still be able to compile that code on the freely available toolchains which will exist 5 years from now.

I don't currently have that confidence with the Ada compilers which are available in 2018.

As I have said before, the language is _really_ good, but the compiler situation is lousy.

From: Simon Wright <simon@pushface.org>
Date: Wed, 23 May 2018 08:26:46 +0100
Subject: Re: DragonEgg has been revived
Newsgroups: comp.lang.ada

> What I would like to see is an Ada compiler that can generate code for a wide range of targets without any GPL restrictions on the generated code.

Pretty sure that's called GCC.

People are perfectly happy to use GCC for C-based commercial projects in spite of the fact that libgcc, and GNU libstdc++, if you're that way inclined, have *exactly* the same runtime exception as FSF GNAT's RTS and the GNAT Pro RTS.

The formal position is that the GCC compiler itself doesn't assert any licensing restrictions over target code generated by it beyond that derived from the original source code.

I can see that people, especially commercial lawyers, might be confused about this, especially if they read all the hot air that's been blasting over this newsgroup lately. It's a good thing that that's unlikely.

Seems to me that one could in theory get over the licensing issue by writing an independent BSD-licensed RTS. Not that this is a small task; deliberately omitting finalization, exception propagation, full tasking, and multiprocessors would make it just about feasible for a small team, I think. But I may be seeing through rose-tinted specs because of having based Cortex GNAT RTS on FSF GCC.

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 23 May 2018 09:11:55 +0100
Subject: Re: DragonEgg has been revived
Newsgroups: comp.lang.ada

> People are perfectly happy to use GCC for C-based commercial projects [...]

It's not just the licensing stupidity with Ada libs. On the C or C++ front, they can easily go to Clang and get more targets, I.e. iOS, Ada programmers can't.

From: Simon Clubley <clubley@eisner.decus.org>
Date: Fri, 25 May 2018 13:16:54 -0000
Subject: Re: DragonEgg has been revived
Newsgroups: comp.lang.ada

[...]

I don't care if the compiler itself is GPL.

I _do_ care if the RTS or anything else is GPL (or even LGPL) and as a result there are constraints imposed on my binaries or source code when I use the compiler. I want to be able to compile programs using the compiler without having to do anything else other than ship the binary for my program.

This is why I talk about generated code and not the compiler.

The FSF runtime exception is fine for me.

I would like to be able to use Ada in all the places I can currently use C and C++ code, including bare metal targets.

If I start using Ada in those places, I would like to be sure that I can still build Ada code for those targets in 2-5 years using the current toolchains of the day, just like I can with C and C++ code.

It would be nice if it was as easy to port a compiler toolchain to a new OS or architecture as it is to port a RTOS to a new target. This would be one answer to the lack of targets for an Ada compiler.

For those of you who have not done RTOS based development, an RTOS is typically very cleanly divided internally into libraries of generic code and target specific low level Board Support Packages (BSP) that implement the required functionality for a specific piece of hardware.

All that it typically takes to port to a new piece of hardware is to write a new BSP and the interface from the BSP to the rest of the RTOS is typically very clean and well documented.

It would be nice if an Ada compiler was also that clean internally and as well documented so that you could easily port it to a new OS or environment yourself if you needed.

[See also “State of the Compiler Market”, AUJ 38-2, p. 75. —sparre]

Qt5Ada

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Sat, 26 May 2018 20:25:41 -0700
Subject: Announce : Qt5Ada version 5.11.0 (548 packages) release 26/05/2018 free edition
Newsgroups: comp.lang.ada

Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.11.0 final)

Qt5ada version 5.11.0 open source and qt5c.dll,libqt5c.so(x64) built with Microsoft Visual Studio 2015 in Windows, gcc x86-64 in Linux.

Package tested with gnat gpl 2012 ada compiler in Windows 32bit and 64bit , Linux x86-64 Debian 9.2

It supports GUI, SQL, Multimedia, Web, Network, Touch devices, Sensors,Bluetooth, Navigation and many others thinks.

Changes for new Qt5Ada release :

Added new packages: Qt.QStringView,Qt.QGraphicsCustomItem,Qt.QGLContext

My configuration script to build Qt 5.11.0 is: configure -opensource -release -nomake tests -opengl dynamic -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked OPENSSL_LIBS="-lssleay32 -llibeay32" -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -icu -prefix "e:/Qt/5.11"

As a role ADA is used in embedded systems, but with QTADA(+VTKADA) you can build any desktop applications with

powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, Internet browsing , Modbus control and many others thinks.

Qt5Ada and VTKAda for Windows, Linux (Unix) is available from

http://hybrid-web.global.blackspider.com/urlwrap/?q=AXicFc47DoJAEIDhOYKnsHMBIb4SotHYqRFiLOxgd4VJZnfI8goXtLbwIB5B7P8v-ScHeH0Avm8AR0MU5qJ2nTAZkmTbOCYh2UAfJWvpW-kH0WK5AmKLSigmk9kdajk2AgnKpqnqjecph50WBXNB-s-9J5PSrkPdb1HFn6etI8T69mA19xckpRW4TlQt_R-nLZ1Fddl5tAWAFCMWz_OvTcj&Z (google drive. It can be mounted as virtual drive or directory or viewed with Web Browser)

The full list of released classes is in "Qt5 classes to Qt5Ada packages relation table.docx"

VTKAda version 8.1.0 is based on VTK 8.1.0 (OpenGL2) is fully compatible with Qt5Ada 5.11.0

I hope Qt5Ada and VTKAda will be useful for students, engineers, scientists and enthusiasts

With Qt5Ada you can build any applications and solve any problems easy and quickly.

[See also “Qt5Ada”, AUJ 39-1, p. 10. —sparre]

TCP/IP in SPARK

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Wed, 30 May 2018 17:40:02 -0700
Subject: SPARK TCP-IP
Newsgroups: comp.lang.ada

Does anyone have a checkout/archive of the SPARK TCP/IP implementation that used to be hosted here: [...]

From: Matti Oinas <matti.oinas@gmail.com>
Date: Wed, 30 May 2018 20:41:03 -0700
Subject: Re: SPARK TCP-IP
Newsgroups: comp.lang.ada

Are you looking this one?

https://github.com/AdaCore/spark2014/tree/master/testsuite/gnatprove/tests/ipstack

Simple Components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 2 Jun 2018 16:34:43 +0200
Subject: ANN: Simple components for Ada v4.29
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, directed graphs, sets, maps, B-trees, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support, streams, multiple connections server/client designing tools and protocols implementations. The library is kept conform to the Ada 95, Ada 2005, Ada 2012 language standards.

http://www.dmitry-kazakov.de/ada/components.htm

Changes to the previous version:

- Handling faulty devices added to the package GNAT.Sockets.Connection_State_Machine.ELV_MAX_Cube_Client;
- Asynchronous execution of remote calls is supported;
- Bug fix in the function To_HTML from GNAT.Sockets.Connection_State_Machine.HTTP_Server;
- Bug fix GNAT.Sockets.Server.Secure.Anonymous, misspelled Initialize.

[See also “Simple Components”, AUJ 39-2, p. 66. —sparre]

Cortex GNAT RTS

From: Simon Wright <simon@pushface.org>
Date: Thu, 07 Jun 2018 17:39:01 +0100
Subject: ANN: Cortex GNAT RTS 20180607
Newsgroups: comp.lang.ada

This release doesn't change any RTS functionality; instead, it reorganises the structure of the source code so as to accommodate the changes made to keep in synchrony with compiler releases without using git branches. [I'd like to shout out a big THANKS! to the person here who reminded me about using variant-specific subdirectories for the changed files. I was getting quite overwhelmed with running 6 parallel branches!]

This affects how you build the RTS: you must specify which release of the compiler you're building for, e.g.

make RELEASE=gnat-gpl-2017

and the RTS must be installed, either with the compiler or locally (you can't any longer use the RTS directly in its build location).

Find at

https://github.com/simonjwright/cortex-gnat-rts/releases/tag/r20180607

[See also “Cortex GNAT RTS”, AUJ 39-2, p. 66. —sparre]

From: Simon Wright <simon@pushface.org>
Date: Sat, 09 Jun 2018 11:57:51 +0100
Subject: Re: ANN: Cortex GNAT RTS 20180607
Newsgroups: comp.lang.ada

> [...]

To build using the new GNAT CE 2018 ARM-ELF compiler, use:

make RELEASE=gcc8

From: Simon Wright <simon@pushface.org>
Date: Thu, 14 Jun 2018 21:47:06 +0100
Subject: ANN: Cortex GNAT RTS 2018-06-14
Newsgroups: comp.lang.ada

This release[1] adds support for the task aspect Secondary_Stack_Size. Writeup at [2].

It turns out that it also supports GNAT CE 2018 (use RELEASE=gcc8).

[1] https://github.com/simonjwright/cortex-gnat-rts/releases/tag/r20180614

[2] https://forward-in-code.blogspot.com/2018/06/secondary-stack-in-cortex-gnat-rts.html

GNAT Community Edition

From: Simon Wright <simon@pushface.org>
Date: Fri, 08 Jun 2018 18:58:05 +0100
Subject: GNAT Community Edition 2018
Newsgroups: comp.lang.ada

GNAT Community Edition 2018 is available. The macOS arm-elf cross compiler is there too! (phew)

[See also "GNAT Community Edition", AUJ 39-1, p. 9. —sparre]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 8 Jun 2018 21:27:18 +0200
Subject: Re: GNAT Community Edition 2018
Newsgroups: comp.lang.ada

> [...]

Great news! 64-bit Windows and 64-bit Gtk.

VisualAda

From: alby.gamper@gmail.com
Date: Sun, 24 Jun 2018 01:10:08 -0700
Subject: ANN: VisualAda (Ada Integration for Visual Studio 2017)
Newsgroups: comp.lang.ada

I am pleased to announce the initial release of VisualAda. This is a extension (aka plugin) for Visual Studio 2017 and is made freely available. Some or the features of VisualAda are:

1) Full edit, build and debug integration

2) GIT, and TFS source control integration

3) Limited Intellisense (For now)

4) Project templates targeting both Desktop and UWP based applications (Note for UWP applications you will need the Ada/Winrt bindings available on GitHub)

Please feel free to download the plug-in from the following URL

https://marketplace.visualstudio.com/items?itemName=AlexGamper.VisualAda

Generic Image Decoder

From: SourceForge
Date: Thu, 28 Jun 2018
Subject: Generic Image Decoder
URL: https://sourceforge.net/projects/gen-img-dec/

https://sourceforge.net/projects/gen-img-dec/files/gid_008.zip/download

The Generic Image Decoder is a package for decoding a broad variety of image formats, from any data stream, to any kind of medium. Unconditionally portable code: OS-, CPU-, compiler- independent code.

Features:

- Supported formats: BMP, GIF, JPEG, PNG, PNM, TGA
- Use of generics and inlining at multiple nesting levels for fast execution
- Standalone (no external dependency)
- Task safe
- Endian-neutral
- Unconditionally portable
- Pure Ada 95 (nothing compiler/system specific), can be used in projects in Ada 95, Ada 2005, Ada 2012 and later language versions

[See also "Generic Image Decoder", AUJ 36-2, p. 65. —sparre]

Zip-Ada

From: SourceForge
Date: Thu, 28 Jun 2018
Subject: Zip-Ada
URL: http://unzip-ada.sf.net

https://sourceforge.net/projects/unzip-ada/files/zipada54.zip/download

Zip-Ada is a library for .zip archives. Full sources are in Ada and are unconditionally portable. Input and output can be any stream (file, buffer,...) for archive creation as well as data extraction. Task safe and endian-neutral.

Features:

- Files and streams supported, for archives and entries, for compression and decompression
- Unconditionally portable
- Task safe
- Endian-neutral
- Standalone
- Zip methods supported for compression: Reduce, Shrink, Deflate, LZMA.
- Zip methods supported for decompression: the above methods, plus: Implode, Deflate64, BZip2
- Pure Ada 95 (nothing compiler/system specific), can be used in projects in Ada 95, Ada 2005, Ada 2012 and later language versions

[See also "Zip-Ada", AUJ 38-4, p. 178. —sparre]

Bar Codes

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Thu, 5 Jul 2018 02:49:32 -0700
Subject: Ann: Ada Bar Codes v.002
Newsgroups: comp.lang.ada

Ada Bar Codes is free and fully Ada open-source.

- Supported bar code formats in v.002: Code 128 and QR Code (new)
- Ready-to-use output formats:
  - o PDF, SVG (vector graphics)
  - o PBM (raster graphics)

http://ada-bar-codes.sf.net/

The project Ada Bar Codes provides a package for generating various types of bar codes (1D, or 2D like QR codes) on different output formats, such as PDF or SVG.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Thu, 5 Jul 2018 17:25:15 +0200
Subject: Re: Ann: Ada Bar Codes v.002
Newsgroups: comp.lang.ada

> [...]

You can use it to display bar codes with Gnoga using Gnoga_Bar_Codes:

https://github.com/jrcarter/Gnoga_Bar_Codes

[See also "Gnoga", AUJ 39-2, p. 65. —sparre]

Ada-related Products

GNAT Pro for BlackBerry QNX

From: AdaCore Press Center
Date: Tue, 15 May 2018
Subject: AdaCore's GNAT Pro Ada Toolchain Released for BlackBerry QNX - AdaCore
URL: https://www.adacore.com/press/gnat-pro-ada-toolchain-for-blackberry-qnx

AdaCore and BlackBerry partnering to support development of critical applications

NEW YORK and PARIS, May 15, 2018– AdaCore today announced a new partnership with BlackBerry to support the company's industry-leading QNX operating system across AdaCore's family of GNAT Pro software tools, including GNAT Pro Assurance, GNAT Pro Enterprise and GNAT Pro Developer. The support for QNX within the GNAT Pro product line will further expand the broad range of embedded platforms available to Ada users and also offer C developers on QNX an easy migration path to the Ada or SPARK languages.

GNAT Pro for QNX is initially targeted for the ARM Cortex A family with plans to support all architectures in the future.

GNAT Pro for QNX comes with a full Ada run-time library supporting all versions of the language from Ada 83 through Ada 2012, together with an early implementation of features that are

expected to be in the next Ada standard. The product includes the GNAT Programming Studio (GPS) IDE and the Eclipse plugin GNATbench, several basic static analysis tools for metrics computation and coding standard verification, the Ada unit testing tool GNATtest, and the SPARK Discovery toolset that can be used to gain experience with formal methods in general and the SPARK language in particular.

“BlackBerry’s QNX operating system is the foundational software in automotive, industrial automation, medical, defense, railway and many other mission-critical systems that require reliability, safety, and security,” said Grant Courville, Head of Product Management at BlackBerry QNX. “We are pleased to partner with AdaCore to support the integration of the GNAT Pro Ada software as this will enable customers who require Ada language support to leverage both companies’ expertise and technology to build reliable, safe and secure QNX-based products.”

"In recent years we've seen increasing interest in Ada from domains outside of the language's traditional aerospace and defense niche," said Quentin Ochem, lead of Business Development at AdaCore. "We're thrilled to combine our forces with one of the leading players in the mission-critical embedded software domain, BlackBerry QNX. Our joint solution will help developers design systems at the highest levels of reliability, safety and security."

[...]

RapiTest

From: Rapita Systems
Date: Wed, 16 May 2018
Subject: (Re)-introducing RapiTest | Rapita Systems
URL: https://www.rapitasystems.com/news/re-introducing-rapitest

Bowing to popular demand, we've renamed our (formerly) RapiTestFramework product to RapiTest. Future versions of the tool will be released with the new name, while existing versions will continue to work as normal.

Why did we make the change? One reason among many is that the former name was somewhat difficult on the tongue and many people, including our own engineers, were already calling the product RapiTest.

RapiTest will continue to reduce the cost of critical software verification by helping engineers efficiently write, run and analyze results from requirements-based tests.

AdaCore Extends Support for Wind River VxWorks Portfolio

From: AdaCore Press Center
Date: Tue, 19 Jun 2018
Subject: AdaCore Extends Support for Wind River VxWorks Portfolio - AdaCore
URL: https://www.adacore.com/press/adacore-extends-support-for-wind-river-vxworks-portfolio

GNAT Pro Ada and VxWorks offer 32-bit and 64-bit support on the latest ARM, Intel, and Power multi-core processors

PARIS & NEW YORK & MUNICH, Germany, June 19, 2018 – Avionics Electronics Europe Conference – AdaCore, a trusted provider of software development and verification tools, today announced the availability of its flagship GNAT Pro Ada Development Environment for the Wind River®VxWorks®7 real-time operating system (RTOS), on leading multi-core hardware platforms. GNAT Pro 18 now supports VxWorks 7 on the ARM (64-bit), Power (64-bit) and Intel (32-bit) architectures, under both Linux and Windows development environments. These releases extend GNAT Pro’s coverage of the Wind River VxWorks platforms, being added to the existing support for VxWorks 7 targets (ARM 32-bit, Power 32-bit, and Intel 64-bit), VxWorks 6 and VxWorks 653, reinforcing the companies’ longstanding strategic alliance.

“Adding support for the new VxWorks platforms continues a long history of Ada on VxWorks,” said Jamie Ayre, Commercial Director at AdaCore. “Hundreds of projects in various domains have benefitted from the close relationship between Wind River and AdaCore. Our software development and verification tools combined with the power of VxWorks allow our customers in the aerospace community – both commercial and military – to develop reliable, safe and secure applications that need to meet the most demanding standards.”

“GNAT Pro Ada’s capability to support Wind River VxWorks 7 on ARM, Intel, or Power hardware platforms will drive down both program cost and risk,” stated Chip Downing, senior director of aerospace and defense at Wind River. “Opening up 64-bit capabilities on multi-core processors will enable a vast new range of applications for our joint customers.”

GNAT Pro customers on VxWorks can choose from several specialized Ada run-time libraries based on project requirements:

- The ZFP (Zero Footprint Profile) with minimal run-time code.
- The Cert profile, which extends the ZFP with features including support for ARINC-653 APEX processes (on VxWorks 653) in Ada or mixed-language applications. The Cert profile is amenable to analysis for inclusion in systems requiring certification under standards such as DO-178B or DO-178C.
- The Ravenscar-Cert profile, which extends the Cert profile with the Ravenscar tasking subset, likewise appropriate for systems needing certification
- Full Ada, for maximal expressibility when certification is not required.

In addition to using one of the certifiable run-time libraries on VxWorks, customers can reduce certification costs by adopting one of AdaCore’s qualifiable verification tools. These include the CodePeer advanced static analysis tool for Ada, the GNATcheck coding standard checker, and the GNATcoverage code coverage analyzer. Certification material for the Cert and Ravenscar-Cert libraries, and qualification material for the qualifiable tools, are available as an option to customers with a subscription to the GNAT Pro Assurance edition.

One of the most promising developments in the avionics community is the Future Airborne Capability Environment (FACE™) initiative, which can help reduce system costs through portable components. VxWorks 653 is the first Commercial-Off-The-Shelf (COTS) product to be certified as conformant to the FACE Technical Standard’s Operating System Segment (OSS) Safety Base Profile. GNAT Pro for VxWorks 653 can thus offer users the benefits of Ada’s high reliability together with the safety-critical support and ease of rapid component integration that come from VxWorks 653 and its FACE conformance.

[...]

Ada and Operating Systems

MacOS: GCC for ARM-EABI

From: Simon Wright <simon@pushface.org>
Date: Sun, 20 May 2018 11:48:04 +0100
Subject: ANN: GCC 8.1.0 for arm-eabi
Newsgroups: comp.lang.ada

This is GCC 8.1.0, rebuilt as a cross-compiler from macOS to ARM-EABI (tested with the Cortex-M3 as found on the Arduino Due[1] and the Cortex-M4 as found on the STMicroelectronics[2] STM32F4 Discovery and STM32F429I Discovery boards; but note that GCC has implemented multilib support for other ARM chips).

Find at

https://sf.net/projects/gnuada/files/GNAT_GCC%20Mac%20OS%20X/8.1.0/arm-eabi/

GNAT GDB 2017 (rebuilt for arm-eabi) is included.

The compiler comes with no Ada Runtime System (RTS). See the Cortex GNAT Run Time Systems project[3] for candidates.

NOTE 1: the compiler/RTS interface has changed; for the time being, you will need to check out the [gcc8] branch.

NOTE 2: for the same reason, this compiler can't presently be used with AdaCore's bb-runtimes repository at Github[4].

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

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

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

[3] https://github.com/simonjwright/cortex-gnat-rts/tree/gcc8

[4] https://github.com/AdaCore/bb-runtimes

[See also “Mac OS X: GCC for ARM-EABI”, AUJ 38-2, p. 77. —sparre]

Windows: 64 bit Bindings?

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Mon, 21 May 2018 12:30:27 -0700
Subject: win64ada or win_32_64_ada ?
Newsgroups: comp.lang.ada

Are there win64ada bindings around, or win_32_64_ada (they would adapt depending on the compilation target, like GNATCOM & GWindows do), or do the current win32ada just work with the 64 bit address types?

Debian: Yearly Migration

From: Nicolas Boulenguez <nicolas.boulenguez@free.fr>
Date: Sat, 26 May 2018 15:11:22 +0200
Subject: yearly migration
Newsgroups: gmane.linux.debian.packages.ada

Next default C compiler on Debian will be gcc-8. Most Ada packages already build with trivial changes [1].

Every library will need to rename its -dev and so package. Such renamings take time because they imply a manual review in the NEW queue.

Please consider updating your packages and uploading them to experimental, where gnat already depends on gnat-8. Once the dust has settled there, we will reupload a consistent set to unstable and hopefully see it quickly migrate to testing.

This is an opportunity to apply unrelated pending changes requiring package renamings, like packaging a new upstream version [2].

[1] As usual, adacontrol fails because of incompatibilities between gnat and ASIS. Updating ASIS [2] will hopefully fix this.

[2] Adacore usually updates its GPL packages in may.

By the way...

Until now, Debian has supported the coexistence of different versions for the default Ada and C compilers. The motivation is to allow a time window or a release where $adaversion < $cversion, so that GCC does not need to wait for all Ada packages to update its version.

More and more packages rely on the gprbuild tool. Currently, gprbuild assumes that all (default) compilers produce the same ABI for all languages. Any effort by successive maintainers to do better has been wasted for 8 years.

- If the user must configure a compiler version manually, they will complain.

- If the tool selects $cversion, it will find no Ada compiler.

- If the tool selects $adaversion, people writing pure C will want the default C ABI.

I suggest that we stop promising that we support the divergence. As far as I know, it has been some years since GCC has not been waiting just for Ada.

Any idea?

Debian: Package Maintenance

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Sun, 10 Jun 2018 15:25:24 +0200
Subject: Asking for best practices to maintain a debian Ada package
Newsgroups: gmane.linux.debian.packages.ada

In the past, I've made several Debian packages for various Ada libraries and tools I'm writing. I've read carefully the Debian Ada policy and I think I followed what was explained. At the end, I was able to provide Debian packages for Ubuntu trusty (14.04) and raring (13.04). You may have a look at the list:

https://blog.vacs.fr/vacs/debian/ubuntu-trusty/index.html

I would like to resurrect these packages and get a better way on how to manage the debian package files. I have a couple of questions and need your advices on how to maintain such files on different Debian versions.

How do you maintain your debian/* files for different versions of Debian and different versions of gnat?

Do you have recommendations on package naming to take into account the library version and the gnat compiler version?

How can I ship a version A of a package for gnat-7 and a version A of the same package for gnat-8?

I'm aware of the Ada-France monotone server. I was able to get the package files for several packages (libaws, libxmlada) but it looks like only gnat-8 is used now.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 20 Jun 2018 21:52:54 +0200
Subject: Re: Asking for best practices to maintain a debian Ada package
Newsgroups: gmane.linux.debian.packages.ada

> [...] How can I ship a version A of a package for gnat-7 and a version A of the same package for gnat-8?

The "aliversions" must be different; the "aliversion" is part of the package *name*. The numerical versions are separate from the name and they can be the same.

For example:

- libfoo1-dev (=8.2.4-4) for gnat-7

- libfoo2-dev (=8.2.4-4) for gnat-8

You would maintain these two as branches in your version control system.

> [...] it looks like only gnat-8 is used now.

Yes, as part of the policy we choose a single version of the compiler and build everything with it. This makes all Ada packages compatible with one another. We don't have the manpower (aka courage, aka time) to maintain several versions of gnat together. Also we don't want a situation where one Ada library is available for one compiler but not the other.

Debian, Ubuntu and Fedora: FSF GNAT

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 11 Jun 2018 16:03:04 +0200
Subject: Re: GNAT Community Edition 2018
Newsgroups: comp.lang.ada

> Where is GCC 8 already available?

Debian (buster), Ubuntu, Fedora, all have GNAT 8, works just fine.

Linux: Docker Images

From: Tomek Wałkuski <tomek.walkuski@gmail.com>
Date: Fri, 29 Jun 2018 02:26:50 -0700
Subject: ANN: Ada / GNAT Docker image
Newsgroups: comp.lang.ada

Hi, for all you Docker folks: Ubuntu with GNAT installed:

https://hub.docker.com/r/tomekw/ada-gnat/

Project repository:
https://github.com/tomekw/ada-gnat

From: Maxim Reznik <reznikmm@gmail.com>
Date: Fri, 29 Jun 2018 08:09:07 -0700
Subject: Re: ANN: Ada / GNAT Docker image
Newsgroups: comp.lang.ada

In case if somebody wants Docker image for GNAT Community 2018, I have one:

https://hub.docker.com/r/reznik/gnat/

Project repository:
https://bitbucket.org/reznikmm/gnat

It contains just compiler (no any extra library such as GNATCOLL, xmlada, bareboard RTL).

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sat, 07 Jul 2018 16:55:12 +0200
Subject: Re: ANN: Ada / GNAT Docker image
Newsgroups: comp.lang.ada

There is also a Docker image with GNAT on Debian by Samuel Tardieu:

https://hub.docker.com/r/rfc1149/gnat/

That's the one I use for testing my Ada libraries, when they are pushed to Bitbucket.

[See also "Getting Started with AVR-Ada", AUJ 38-4, p. 180. —sparre]

References to Publications

Concurrent Programming

From: Mehdi Saada <00120260a@gmail.com>
Date: Sun, 20 May 2018 03:53:18 -0700
Subject: tutorial for concurrent programming techniques (in Ada).
Newsgroups: comp.lang.ada

Is there somewhere such tutorial online?

Something with the basic techniques, and the use of standard libraries?

I'm at my first scheduling program, and I'm always telling myself "I would like to do that and that, I saw it somewhere but I've no clue how to do it". Things that are complicated per se, but beginners would gain to see gathered.

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Sun, 20 May 2018 10:21:26 -0400
Subject: Re: tutorial for concurrent programming techniques (in Ada).
Newsgroups: comp.lang.ada

> Is there somewhere such tutorial online?

Well...

- https://en.wikibooks.org/wiki/Ada_Style_Guide/Concurrency

- https://en.wikibooks.org/wiki/Ada_Programming/Tasking

Though in truth, one should have some familiarity with general tasking concepts (semaphores, mutex, critical section) -- which used to be part of any course in operating systems.

Mostly concerned with hard real-time:

- https://www.amazon.com/Concurrent-Real-Time-Programming-Alan-Burns/dp/0521866979

- https://www.amazon.com/Analysable-Real-Time-Systems-Programmed-Ada/dp/1530265509

- https://www.amazon.com/Real-Time-Systems-Programming-Languages/dp/0201729881

More general...

- https://www.adacore.com/papers/a-comparison-of-the-concurrency-and-real-time-features-of-ada-95-and-java

- https://www.amazon.com/Communicating-Sequential-Processes-International-Computing/dp/0131532715

- http://greenteapress.com/wp/semaphores/ (semaphores can be modeled in Ada using protected objects; though I seem to recall textbooks pre-Ada95 showing how to do them via rendezvous and tasks)

> Something with the basic techniques, and the use of standard libraries?

Ada's tasking model is built into the language itself, and is not a library (except as affected by the degree of the run-time system and operating system features -- bare-board development requires one to provide a run-time that supports tasking natively, whereas Linux/Windows development punts from the run-time to the operating system).

From: Olivier Henley <olivier.henley@gmail.com>
Date: Tue, 22 May 2018 07:25:48 -0700
Subject: Re: tutorial for concurrent programming techniques (in Ada).
Newsgroups: comp.lang.ada

> [...]

Not a tutorial per se, but this is an !AWESOME! read: "Concurrent and Real-Time Programming in Ada" [see link above —sparre]

"Building Parallel, Embedded, and Real-Time Applications with Ada" (Covers PolyORB, for distributed programming): http://www.cambridge.org/core_title/gb/395592

Definitely the first one. You will greatly cover tasks, protected objects and real-time scheduling. All along it exemplify how to program different systems like worker pool, futures etc. Worth every penny.

From: Olivier Henley <olivier.henley@gmail.com>
Date: Wed, 23 May 2018 10:30:12 -0700
Subject: Re: tutorial for concurrent programming techniques (in Ada).
Newsgroups: comp.lang.ada

> [...]

The first book goes deeper about concurrency and scheduling. ~300p on concurrency and ~150p on scheduling. (Alan Burns and Andy Wellings)

The second book is ~100p to present the Ada itself (type model, oop etc), ~100p for concurrency, ~100p for distributed computing (PolyORB, Corba, etc) and finally ~100p for real-time and scheduling. (John W. McCormick, Frank Singhoff, Jerome Hugues)

Note: It looks to me they are not the same people.

Best is to buy both. They are very neat book and inspire to do everything using Ada. :)

Simple Blockchain

From: Tomek Wałkuski <tomek.walkuski@gmail.com>
Date: Wed, 20 Jun 2018
Subject: Simple blockchain in Ada
URL: https://tomekw.com/simple-blockchain-in-ada/

I consider myself a late adopter. Everyone talks about blockchain these days. Everyone tries to apply the technology everywhere, even when it doesn't make sense. So let's learn by doing and try to implement the simple blockchain from scratch. And let's do this in Ada!

Wikipedia defines blockchain as:

"A blockchain, originally block chain, is a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block typically contains a cryptographic hash of the previous block, a timestamp, and transaction data. By design, a blockchain is resistant to modification of the data."

Twitter has a different opinion:

"High-latency, low-throughput, append-only database with very expensive transaction commit protocols just doesn't have the same ring to it as "Blockchain" does it?"

So it looks like we have to model:

- blocks: timestamped records that are able to store some kind of payload

- a chain of blocks, a.k.a., the blockchain

- a way to prove the validity of the whole blockchain

- a proof of work to implement the commit protocol.

Block can be implemented as a new Ada type, Block.Object:

```
[...]
package Simple_Blockchain.Block is
   type Object is private;
   [...]
private
   type Object is record
      Cryptographic_Hash_Current_Block :
         String (1 .. 64);
      Cryptographic_Hash_Previous_Block
         : String (1 .. 64);
      Timestamp : Time;
      Transaction_Data_Payload :
         Unbounded_String;
   end record;
end Simple_Blockchain.Block;
```

The complete, working code can be found at tomekw/simple_blockchain Github repository: https://github.com/tomekw/simple_blockchain

[...]

Book on Cyber Security

From: AdaCore Press Center
Date: Wed, 27 Jun 2018
Subject: AdaCore Shows How to Address the Cyber Security Challenge - AdaCore
URL: https://www.adacore.com/press/adacore-shows-how-to-address-the-cyber-security-challenge

Free book offers guidance for achieving secure and reliable software

PARIS & NEW YORK & GAITHERSBURG, Maryland, June 27, 2018 – Workshop on Sound Static Analysis for Security - AdaCore, a trusted provider of software development and verification tools, today announced the availability of AdaCore Technologies for Cyber Security, the latest volume in its series of free publications on high-assurance software. Authored by internationally known experts Dr. Roderick Chapman and Dr. Yannick Moy, the book explains why making a cyber system secure is so difficult and shows how using appropriate programming languages and tools can contribute to a solution. Languages such as Ada and SPARK, and verification based on formal methods or other sound static analysis techniques, can prevent vulnerabilities from being introduced in the first place. They can also detect latent issues in legacy codebases, including many of the weaknesses in the MITRE Corporation's Common Weakness Enumeration (CWE).

"Many of the nasty security-related incidents that we've seen over the past few years stemmed from entirely preventable software errors," said co-author Yannick Moy, a senior software engineer at AdaCore. "By following the guidance presented in our new book, software developers can learn from history and avoid repeating it."

"Developing software that is robust in the face of malicious attack is a huge engineering challenge," said co-author Roderick Chapman. "It requires a world-class combination of languages, technologies, disciplines and skills. Ada, SPARK, and AdaCore's tools can provide some key pieces of that puzzle. In particular, AdaCore has pioneered the development of static verification techniques that are both formal and sound, and so offer real assurance. I hope the book will inform, entertain, and challenge readers' preconceptions about how high-assurance software can be developed and verified."

AdaCore Technologies for Cyber Security contains four principal chapters.

- "The Challenge of Secure Software" identifies the various factors that make security so hard to achieve, ranging from the interconnected nature of modern systems to the limits of testing as a verification mechanism. The chapter concludes with "A Manifesto for Secure Software" that outlines the basic principles for high-integrity software engineering.
- "Languages, Tools and Technologies Overview" summarizes the Ada and SPARK languages, as well as AdaCore's tools and technologies, and highlights their contributions to system security.
- "Security Vulnerabilities and Their Mitigation" considers a number of specific high-profile software vulnerabilities, inspired by the CWE/SANS "Top 25 Most Dangerous Software Errors", and discusses how each can be prevented or mitigated using Ada, SPARK, and AdaCore's tools.
- "Industrial Scenario Examples" presents a number of security-related scenarios that may arise in real-world projects. Each opens with a description of the context and the security issue, and then shows how either Ada or SPARK, in conjunction with the relevant AdaCore tools, can contribute. Each scenario is illustrated with one or more examples drawn from experience with customers and industrial projects.

Complementing the discussion in these chapters, additional details and examples are provided in two appendices. One appendix focuses on the MITRE Corporation's Common Weakness Enumeration (CWE) and shows how the use of Ada and/or SPARK, as well as AdaCore's tools, can address specific CWEs. The second appendix shows how contract-based programming in SPARK or Ada, verified by the corresponding static or dynamic analysis, can help avoid the "SQL Injection" vulnerability.

"AdaCore Technologies for Cyber Security" is available at

<http://adacore.com/cyber-security-book>; to request a printed copy, please contact info@adacore.com.

[...]

Introduction to Ada Programming

From: Andrew Shvets <andrew.shvets@gmail.com>
Date: Fri, 29 Jun 2018 20:23:21 -0700
Subject: Introduction to Ada Programming, 2nd Edition
Newsgroups: comp.lang.ada

In 2016, I published an introductory Ada book as an e-book. This time around, I have created a second version of this book, located here: https://amzn.to/2KC3Zic

This is what the second edition has:

- It's in print. You can buy a printed version. I've read programming e-books before, but mostly it has been a question if my screen/monitor was large enough to correctly display the source-code (it often was, but was impossible on an iPhone).
- I added a chapter on networking. Networking is something that I'm working on improving my understanding. Only after I sufficiently understood how to implement examples in Ada did I add this to my next book.
- There is a chapter on how to build libraries. Packages and methods are great for encapsulating code and making it more reusable. However, not having to re-compile the code is simply phenomenal.
- There is a chapter on proofs. In my previous book, this was a part of a chapter. In this book, it is a chapter. Such an important topic earned its own chapter.
- I had a professional editor looked over the book. After my first attempt, I realized that there is plenty of room for improvement in subsequent versions. As a result, I hired a professional to give me invaluable input on specific things that I can do better.

I'm open to sending PDFs as review copies, please send your requests to: introductory.ada@gmail.com

Thank you for those who have helped me better understand Ada!

[See also "Introductory Ada Programming Book", AUJ 38-1, p. 10. —sparre]

Méthodes de Génie Logiciel avec Ada 95

From: Olivier Henley <olivier.henley@gmail.com>
Date: Wed, 4 Jul 2018 08:21:55 -0700
Subject: Looking to buy Méthodes de Génie Logiciel avec Ada 95 (paper version)
Newsgroups: comp.lang.ada

I am looking to buy the paper version of:

Méthodes de Génie Logiciel avec Ada 95, J-P. Rosen, ISBN 2 7296 0569 X

Anyone has a copy and would to sell it to me? olivier.henley@ 'google_email_service'.com

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 5 Jul 2018 16:42:36 +0200
Subject: Re: Looking to buy Méthodes de Génie Logiciel avec Ada 95 (paper version)
Newsgroups: comp.lang.ada

[...]

It has been issued as a WikiBook (thanks to Pascal Pignard):

https://fr.wikibooks.org/wiki/ Méthodes_de_génie_logiciel_avec_Ada

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 6 Jul 2018 11:38:25 +0200
Subject: Re: Looking to buy Méthodes de Génie Logiciel avec Ada 95 (paper version)
Newsgroups: comp.lang.ada

[...] I made it available on Adalog's web site. The link is at the bottom of the page at:

http://www.adalog.fr/fr/livrejpr.html

Ada Inside

CLARREO Pathfinder

From: AdaCore Press Center
Date: Tue, 22 May 2018
Subject: University of Colorado's Laboratory for Atmospheric and Space Physics adopts Ada and GNAT Pro for NASA project
URL: https://www.adacore.com/press/university-of-colorados-laboratory-for-atmospheric-and-space-physics-adopts-ada-and-gnat-pro-for-nasa-project

Ada selected over C to run on a Cortex M1 core

PARIS & NEW YORK,May 22, 2018 – AdaCore today announced that the University of Colorado's Laboratory for Atmospheric and Space Physics (LASP) has selected the Ada language and the GNAT Pro for the ARM Cortex product for NASA's Climate Absolute Radiance and Refractivity Observatory (CLARREO) Pathfinder mission. CLARREO Pathfinder will deploy a Reflected Solar spectrometer on the International Space Station (ISS) starting in 2021 that will detect the complete spectrum of radiation from the Sun reflected by Earth.

LASP has selected the Ada language over C, to develop the orchestration and interface portions of the CLARREO Pathfinder flight software, which is responsible for controlling the instruments and interfacing with the ISS. The application will run on an ARM Cortex M1 FPGA board, using a bare metal configuration together with the Ravenscar micro-kernel provided by the GNAT Pro toolchain.

"We selected Ada and the Ravenscar micro-kernel for several reasons: it is as efficient as C, allows object-oriented design, will increase reliability, and provides a tasking system without introducing a great deal of complexity like many of the other options we considered," said Mathew Merkow, CLARREO Pathfinder flight software lead at LASP."Ada provided an extremely robust and efficient foundation for our framework, Adamant. We partnered with AdaCore to port Ravenscar to the Cortex M1; they have been a great partner, and we are excited to continue our relationship with them on this and future projects."

"The CLARREO Pathfinder project represents a new generation of applications developed with Ada, in areas where C has been the traditional choice," said Quentin Ochem, lead of business development at AdaCore. "We are excited to support the usage of our technology to meet the ever-increasing reliability requirements and challenges of space missions."

[...]

Jobs

From: eduardsapotski@gmail.com
Date: Sun, 3 Jun 2018 23:15:27 -0700
Subject: How to find remote job as Ada-developer?
Newsgroups: comp.lang.ada

I have more than ten years of experience in programming. Mainly programmed on C# and Java. Very long ago I programmed Atmel-microcontrollers on C-language. In recent times I'm sick of Java and C#. A year ago I met Ada-language. I like everything! Ada is the most correct programming language I have ever met! While using Ada-language only in their small projects.

I have no experience of industrial development in Ada-language. How to get it? There are no vacancies for Ada-developers in my region.

I'm not really interested in money. I am willing to work for a nominal fee, only to gain experience.

Who can advise?

From: Luke A. Guest <laguest@archeia.com>
Date: Sun, 3 Jun 2018 23:28:55 -0700
Subject: Re: How to find remote job as Ada-developer?
Newsgroups: comp.lang.ada

> [...]

You're basically stuck with military, aerospace, some car company's and train company's, I think there's some medical equipment out there using Ada.

Ada is just not advertised enough.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon, 04 Jun 2018 08:49:10 +0200
Subject: Re: How to find remote job as Ada-developer?
Newsgroups: comp.lang.ada

> I have no experience of industrial development in Ada-language. How to get it?

Check out the list of companies AdaCore advertises as their customers.

See if you would be willing to move to any of their locations.

Check if they have open positions and apply - or send them unsolicited applications.

I doubt that you will get a remote position without documented experience in Ada. Even with documented Ada experience, it is hard. And for big projects, you should expect 6-24 months on-site just to get to know the project well enough to be able to work independently.

Ada in Context

Anonymous Allocators

From: Mehdi Saada <00120260a@gmail.com>
Date: Wed, 16 May 2018 01:23:19 -0700
Subject: Re: little precision about anonymous access types
Newsgroups: comp.lang.ada

I may add, that the craziest thing was to allow the very possibly of using allocators with non-discriminant/non-parameter anonymous access (though I have no idea, and it's not easy to find, where and for how long goes things like values like my_func(new something', ...) types. Their existence, that forbid and uselessness aside, it wouldn't be that much of a loophole. At least provide a way to bind the objects' life time to something, dudes...

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 17 May 2018 16:20:14 -0500
Subject: Re: little precision about anonymous access types
Newsgroups: comp.lang.ada

The better solution here is restriction No_Anonymous_Allocators (see H.4(8.1/3)). Using that restriction helps because it forces all allocation to named access types (for which you can do deallocation in all of the normal ways). This is just to note that Ada does have ways to mitigate this problem (I noted another one, pragma Default_Storage_Pool, in a previous message). The annoyance is that these things aren't the default.

ARM ELF Development with GNAT GPL

From: Adam Jensen <hanzer@riseup.net>
Date: Wed, 23 May 2018 06:37:44 -0000
Subject: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

I would like to tinker with Ada and SPARK embedded real-time software development. [...]

I have installed AdaCore's GNAT GPL and SPARK Discovery on Ubuntu 18.04 LTS [...]

And I've installed gnat-gpl-2017-arm-elf-linux-bin.tar.gz [...]

I am following this tutorial: http://www.inspirel.com/articles/Ada_On_Cortex.html

[...]

Given:

```
$ export PATH="$HOME/.local/gnat-arm/bin:$PATH"
```

The tutorial suggests that maybe this (below) might produce a binary suitable to be loaded onto the MCU:

```
$ arm-eabi-gcc -c -mcpu=cortex-m4 -mthumb program.adb
$ arm-eabi-ld -T flash.ld -o program.elf program.o
$ arm-eabi-objcopy -O binary program.elf program.bin
```

This is what actually happens:

```
$ arm-eabi-gcc -c -mcpu=cortex-m4 -mthumb program.adb
fatal error, run-time library not installed correctly
cannot locate file system.ads
compilation abandoned
```

I guess that the LD_LIBRARY_PATH and GPR_PROJECT_PATH environment variables should be set but I don't yet understand enough to make reasonable guesses.

Any advice on how to proceed would be very much appreciated!

From: Simon Wright <simon@pushface.org>
Date: Wed, 23 May 2018 09:07:51 +0100
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> Any advice on how to proceed would be very much appreciated!

I think that the reason why the tutorial works and your attempt doesn't is that the tutorial was developed on a Raspberry Pi, which is already an ARM-based machine, so the native compiler actually has a runtime (i.e. system.ads etc) visible to it. Yours doesn't, and gcc-for-ada must see an RTS.

I managed to get a compilation using this over-the-top incantation:

```
$ gprbuild -c -u -f program.adb --target=arm-eabi --RTS=zfp-stm32f4
```

but a simpler (more memorable!) procedure might be to construct your own:

1. Create directories adainclude/, adalib/
2. Copy $prefix/arm-eabi/lib/gnat/zfp-stm32f4/gnat/system.ads to your adainclude/ ($prefix is the root of your compiler installation, I think ~/.local/gnat-arm)

The ZFP (zero footprint) runtime is the closest to what you need, and the fact that the -stm32f4 part isn't quite right shouldn't matter; I suspect that system.ads is the same for all the zfp runtimes.

Now,

```
$ arm-eabi-gcc --RTS=. -c program.adb
```

From: Adam Jensen <hanzer@riseup.net>
Date: Thu, 24 May 2018 07:35:46 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> [...]

```
>  $ gprbuild -c -u -f program.adb --target=arm-eabi --RTS=zfp-stm32f4
```

Many thanks, this works! I do not yet know why it works - what it is doing - but the hint is a valuable. It occurred to me that I should be looking for AdaCore documentation. I have yet to find a "getting started" tutorial for embedded development aimed at scientists, engineers and other technically mature people (ideally, such a tutorial would be comprehensive, to the point, and regularly tested), but I did find:

GPRbuild and GPR Companion Tools User's Guide <https://docs.adacore.com/gprbuild-docs/html/gprbuild_ug.html>

and

GNAT User's Guide Supplement for Cross Platforms <http://docs.adacore.com/live/wave/gnat_ugx/html/gnat_ugx/gnat_ugx.html>

By mining these two documents it might be possible to extract a basic explanation for these very first steps of the embedded development process.

It is curious that the Ada technology's utilization of the system engineering approach has not [yet] been extended into the pedagogical component. After all, software development is a very human-centric process.

But I digress. Thanks again for the guidance! After extracting the basics of project management from the core documentation, I hope to find a build process that enables the use of a Ravenscar profile. And after that I hope to configure the development environment to include SPARK process components in the real-time embedded system. Does this seem reasonable?

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Thu, 24 May 2018 12:12:26 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> [...]

The error you saw "Cannot find system.ads" and Simon's answer arise because, targeting small embedded CPUs, you are looking below the full Ada environment (supplied by the runtime system (RTS) on the host machine, to targets which may require unique runtimes supplying the facilities you need and nothing more (thanks to potential space limitations).

As such I would suggest a ZFP RTS as a good short-term study, for several reasons:

- it can be a pretty small codebase, but revealing in terms of how things are done and how to adapt them.
- there are a plethora of targets out there, un- or semi-supported by Ada, from the AVR and MSP430 to ARM cores from ST, NXP, TI and others. Starting with the STM4 as you are is good, but you may want to port to other platforms for cost, power, security or other reasons ... the TI Hercules which runs dual ARM cores in lockstep for safety, has obvious attractions as an Ada target, for example.

Nice price too ... https://store.ti.com/LAUNCHXL-TMS57004.aspx

And porting to these builds on understanding the RTS, starting with the simplest - ZFP - as in Simon's suggestion - or AVR-Ada or my MSP430-Ada adaptation. I finally got round to machining the case and bezel, so I am wearing a wristwatch running Ada, telling the time 1970's style, in under 1 kilobyte including RTS.

(the current version still has 200 bytes of C startup code which the linker insists on inserting by default; one TODO is to persuade the linker to let me replace that with pure Ada and strip out the unnecessary stuff)

You suggest going in 2 more interesting directions:

- Building up to a Ravenscar profile: I believe Simon's work so far builds on FreeRTOS, but a "native" Ravenscar RTOS would be nice too...
- SPARK qualification would be excellent ... again, especially for the Hercules. And again, a SPARK proven ZFP RTS

would be a good base to build on, and a relatively simple place to start.

From: Adam Jensen <hanzer@riseup.net>
Date: Fri, 25 May 2018 04:45:16 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> [...]

AdaCore's GNAT GPL seems to include a full Ravenscar RTS for Xilinx's zynq7000 ARM/FPGA SoC: <https://www.xilinx.com/products/boards-and-kits/device-family/nav-zynq-7000.html>

One of those development kits might be my next target platform, but successfully configuring tools from two different vendors for hardware/software co-design - simulation, emulation, and cross-compilation - on a third-party OS (Ubuntu or RHEL) seems like a long way off. Right now, configuring a Ravenscar/SPARK development environment that can produce a binary for the Nucleo-144 board that will flash an LED is the [surprisingly challenging] goal :) [...]

Is it common for developers to create their own run-time system for embedded platforms? My inclination would be to look for hardware based on 1) RTS availability/quality and 2) toolkit complexity/completeness (completeness implies useful documentation). Given that, which seems like an obvious thing to do, I am surprised that AdaCore does not have more apparent associations with hardware vendors where dev-kits and SBC products are promoted. I bought the Nucleo-144 board because I thought there was a BSP, RTS, and a tool-chain configuration tutorial. That turned out to be a bit of a mistake and generally a poor choice. If AdaCore, or some other enterprising entrepreneur, offered well-developed BSP, RTS, tool-chain configuration and programming tutorials for several MCU dev-kits and SBC (single board computer) products, that would make the choice easy and actually enable people to get started with the technology in a reasonable way. It seems so bizarre to me that this isn't a front-page item for AdaCore. I guess there are hidden obstacles in their business model and the way the incentives are arranged in their social organization. I suppose it could have something to do with European culture. In France, does pedagogy have the demeanour of a wood-chipper (e.g., is it based in punishment, toil, and obscurity)? <smirk>

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Fri, 25 May 2018 10:50:07 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> Is it common for developers to create their own run-time system for embedded platforms? [...]

Not common, no. But RTS availability (esp. SPARK RTS) has to start somewhere, and for the MSP430 I didn't really develop one, just adapt from AVR-Ada.

With remarkably little feedback on that project, I admit I've put remarkably little effort into pushing it further. But I want it for my own purposes, the watch is just a pretty by-product.

>[...] I am surprised that AdaCore does not have more apparent associations with hardware vendors where dev-kits and SBC products are promoted. [...]

Not AdaCore ... there isn't much hobbyist money for them, given their business model. They do publicise occasional hobby-level projects like LEGO Mindstorms and Certyflie, but I don't see them making money off it.

Meanwhile we have to support each other, perhaps your work on Nucleo can feed back into Simon's RTS and expand its supported platforms.

From: Adam Jensen <hanzer@riseup.net>
Date: Sat, 26 May 2018 05:06:40 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> [...]

> Not AdaCore ... there isn't much hobbyist money for them, given their business model. They do publicise occasional hobby-level projects like LEGO Mindstorms and Certyflie, but I don't see them making money off it.

They have the "Make with Ada" competition: <https://www.makewithada.org/>

And the AdaCore University: <http://university.adacore.com/>

If there is not a large vibrant community of people who understand and use the technology it will fade and collapse. It seems like maybe they recognize this but it doesn't seem like they know what to do. (Only an idiot would have advertisers involved in technical communication). C'est la vie.

> Meanwhile we have to support each other, perhaps your work on Nucleo can feed back into Simon's RTS and expand its supported platforms.

The Nucleo-144 board was selected as a gentle starter kit to develop some confidence and familiarity with the tool-chain and the work-flow. It was a total failure in this role. However, I have been keeping notes and at some point I might create a tutorial for Ada/SPARK development on Ubuntu x86_64 targeting the ARM MCU on a Nucleo-144 board. After that, I will probably move to a platform with more resources. Eventually, I need a processor coupled with an FPGA - the FPGA is where most of the hard real-time activity (traction with physics) should take place, IMO.

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Sat, 26 May 2018 23:58:17 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

> Super cool. Are your project's documents posted/hosted anywhere for others to view and use?

https://sourceforge.net/projects/msp430ada/

It's somewhat stuck in the past, using Peter Bigot's rather nice MSP430 backend, because gcc's own MSP430 backend built into newer versions have a considerably poorer code generator (last time I looked a couple of years ago).

Revisiting it is on my to-do list, hopefully it has improved.

> [“Make with Ada” and AdaCore University]

Both good forms of publicity, though I wonder to what extent they manage to bring in new people as opposed to reaching to the converted.

[...]

From: Adam Jensen <hanzer@riseup.net>
Date: Fri, 25 May 2018 03:29:59 -0000
Subject: Re: How to configure GNAT GPL on x86-64 Linux for ARM ELF development
Newsgroups: comp.lang.ada

[...]

> I think that the reason why the tutorial works and your attempt doesn't is that the tutorial was developed on a Raspberry Pi, which is already an ARM-based machine, so the native compiler actually has a runtime (i.e. system.ads etc etc) visible to it.

This is so very relevant yet the tutorial seems rather vague [to me] on this point. It should be explicit, in bold, in a highlighted box on the front page, IMO. Thanks again for pointing that out. Even on a second reading, it isn't clear [to me] that using a Raspberry Pi as a software development platform is the environment of the tutorial.

<http://www.inspirel.com/articles/Ada_On_Cortex_Documentation_And_Tools.html>

Multiple Iterators for a Type

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 25 May 2018 09:49:46 -0700
Subject: Multiple iterators for a type
Newsgroups: comp.lang.ada

I want to have a type which is an array of 8 bit values, I want the default iterator to be the normal array loop.

But then I want to add more iterators which return different types but constructed from the array, i.e. a 32-bit value and a sub-array.

1. Can this be done on the base type or do I need to create new types from the base type?
2. if 1. can be done, do these iterators all need to be one package or can I put them in child packages?

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Fri, 25 May 2018 21:50:08 +0200
Subject: Re: Multiple iterators for a type
Newsgroups: comp.lang.ada

> [...] Can this be done on the base type [...]?

It can be done on the base type.

> [...] or can I put them in child packages?

They can be in child packages. Or even locally defined in the subprogram where you use them.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 25 May 2018 16:50:07 -0500
Subject: Re: Multiple iterators for a type
Newsgroups: comp.lang.ada

> [...]

There can only be one "of" iterator, and it's built-in for array types. To replace the "of" iterator you need different private types (which means of course that they can't directly be used as arrays, either, although you can emulate that). Why you'd want to go through that escapes me.

You can explicitly use alternate iterators using the "in" syntax. After all, any iterator object can be iterated (duh!), and you can create as many different ones of those as you want/need.

The "of" iterator is just a convenience, and I think the language would have been just fine without it. Ignore its existence and you'll be just fine and can have all of the iterators you ever could need.

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 26 May 2018 04:57:45 +0100
Subject: Re: Multiple iterators for a type
Newsgroups: comp.lang.ada

> [...]

I'm attempting to implement a Unicode string using UTF-8, so I want the basic iterator over octets, then the next will iterate over the octets and generate code points, then another will be graphème clusters.

From: Jeremiah Breeden <jhb.chat@gmail.com>
Date: Fri, 25 May 2018 21:44:16 -0700
Subject: Re: Multiple iterators for a type
Newsgroups: comp.lang.ada

> [...]

When I did multiple iterators I ended up making a package for it, which I later adapted just for fun to provide iteration of types in generics. The steps I ended up doing were:

1. Create my Cursor type and Has Element
2. Create a set of functions returning reference types, but use a package to do it so I could pass them into another generic
3. Instantiate Iterator_Interfaces for my cursor
4. Implement my iterator
5. Pass it into a package that created an iterable wrapper

Technically if you just want "in" iteration, you stop at #4, but I like the "of" version so that is why I made a package for step 5.

In the end I was able to get something like:

```
for E of Iterable (Container_Object) loop
   E.Do_Things;
end loop;
```

where Iterable is a function from my generic package that returns an iterable version of Container object. It was handy for generics and having multiple iterators, though it comes at a performance cost since it uses a layer on top of the container.

Package for making reference types:
https://github.com/jeremiahbreeden/bullfrog/blob/master/src/bullfrog-access_types-references.ads

Package for making "of" iterable wrappers:
https://github.com/jeremiahbreeden/bullfrog/blob/master/src/bullfrog-containers-iterable_wrappers.ads

Package with some predefined wrappers of the standard containers to my iterable wrappers:
https://github.com/jeremiahbreeden/bullfrog/blob/master/src/bullfrog-containers-predefined_iterable_wrappers.ads

Some testing of the predefined wrappers that I made:
https://github.com/jeremiahbreeden/bullfrog/blob/master/src/tests/test_generic_iteration.adb

Literals for Private Types

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 30 May 2018 14:46:08 -0500
Subject: Re: Strings with discriminated records
Newsgroups: comp.lang.ada

> [...]

See AI12-0249-1. This hasn't been discussed at a meeting yet, so it probably will change some, but Tucker suggests aspects "Integer_Literal", "Real_Literal", "Null_Literal", and "String_Literal". These represent functions that can be specified:

```
type Message (discriminants or not) is ...
   end record
  with String_Literal  => Make_Message;
```

where Make_Message is something like:

```
function Make_Message (Lit : in
      Wide_Wide_String) return Message;
```

with the obvious semantics when a string literal is encountered.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 30 May 2018 14:48:43 -0500
Subject: Re: Strings with discriminated records
Newsgroups: comp.lang.ada

BTW, the reason that I said "it might change" is that there are some issues with what to do with private types where the full type "naturally" has the same kind of literal. I don't think that is handled quite right yet, and it's relatively important to get right (user-defined aggregates have a similar, even worse problem).

[See also <http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai12s/ai12-0249-1.txt?rev=1.2&raw=N>. —sparre]

Use Clauses

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 30 May 2018 15:09:54 -0500
Subject: Re: Strings with discriminated records
Newsgroups: comp.lang.ada

> [...] What's the problem?

(A) The person I was responding to was upset that they had to write the fully qualified name in this case, and practically, they're right. (Tucker has complained about this multiple times.)

(B) But package use clauses make things visible that are not overloadable (like objects), so they tend to be a substantial maintenance hazard -- adding something to a package can make client code using use clauses illegal because of conflicts. Adding unrelated stuff should *never* make any client illegal (changes, obviously are different).

If Ada had made objects/exceptions overloadable, this problem would be much less severe. But it didn't, and changing that now would be difficult to do without lots of subtle incompatibilities.

Use type/use all type (and better still, prefix calls) mostly avoid this problem by only operating on overloadable things. You still can get conflicts, but only when there are design issues (having multiple operations with the same name and profile

means either there is unnecessary duplication [two routines doing the same thing] or that there is routines doing different things with the same name (yikes!).

In a best case world, a rename conflicting with the original routine or another rename of it would be ignored in all use clauses, along with overloading of objects/exceptions. That would reduce conflicts to a handful of cases. (If overloading of generics could be figured out, that would be even better.) But that has to wait for Ada++.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 31 May 2018 06:19:28 +0200
Subject: Re: Strings with discriminated records
Newsgroups: comp.lang.ada

> [...] package use clauses make things visible that are not overloadable [...] tend to be a substantial maintenance hazard [...]

Here I don't agree. OF COURSE, changing a specification can make client code illegal, with or without use clauses. And I would not call making code illegal a "maintenance hazard"; on the contrary, a maintenance hazard is when a change does not make code illegal, but acts differently. We know how hard Tuck fought in Ada 95 to eliminate the Beaujolais effect...

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 31 May 2018 17:18:37 -0500
Subject: Re: Strings with discriminated records
Newsgroups: comp.lang.ada

> [...]

Whenever large amounts of code depend on some package, causing unusual illegalities in working code just because of the addition of a new object/exception is a major problem. Consider something like Claw: we have to avoid making changes to the specs -- even ones that are clearly good ideas -- in order to avoid breaking user code. Similarly, I have to document *every single* change in a language-defined package as an incompatibility -- even though only people overusing use clauses have a possibility of such an incompatibility. And this is a real problem; it bites me often in Janus/Ada (which itself overuses use clauses) -- the main reason that I hardly ever use them in new code.

Ada's "solution" of making things illegal is better than silent changes (although those can happen, too, especially in child units), but the best situation is one where adding new subprograms/objects/exceptions don't have any effect at all on existing code (in the absence of dubious design - that is multiple different things with the same name and profile). Anything else makes it hard to enhance libraries cleanly (you end up with unnecessary child packages - like "Ada.Directories.Hierarchical_File_Names" - to avoid the incompatibilities - and that itself is just another kind of pain.

Execute External Program

From: John Smith <yoursurrogategod@gmail.com>
Date: Sun, 3 Jun 2018 20:17:31 -0700
Subject: Trying to execute a command from inside of Ada
Newsgroups: comp.lang.ada

I found the following example: http://rosettacode.org/wiki/Execute_a_system_command#Ada

And this is how I tried to adapt it to Linux:

```
with Interfaces.C;

with Ada.Text_IO; use Ada.Text_IO;
with GNAT.OS_Lib; use GNAT.OS_Lib;

procedure Sys_Command is
   Result : Integer;
   Arguments : Argument_List :=
               (1 => new String'("bash"),
                2 => new String'("ls -l ~"));
begin
   Spawn (Program_Name  => "bash",
          Args  => Arguments,
          Output_File_Descriptor => Standout,
          Return_Code  => Result);
end Sys_Command;
```

The problem is that 'ls -l ~' is not executed correctly. I don't see any output at all. What am I doing wrong?

From: Yuta Tomino <aghia05@gmail.com>
Date: Sun, 3 Jun 2018 21:42:09 -0700
Subject: Re: Trying to execute a command from inside of Ada
Newsgroups: comp.lang.ada

> [...]

"-c" switch is needed for bash to pass the subcommand. Try to compare them in your interactive shell.

$ bash "ls -l ~"

$ bash -c "ls -l ~"

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Mon, 04 Jun 2018 08:44:10 +0200
Subject: Re: Trying to execute a command from inside of Ada
Newsgroups: comp.lang.ada

> [...]

Have you tried to run this on your command line?

```
bash "ls -l ~"
```

That's basically what you ask your program to do.

I would:

1) Avoid involving Bash in this.

2) Remember to pass each argument separately.

And I might additionally:

3) Expand "~" myself, as "ls" doesn't know how to do that (but "system()" or "/bin/sh" might).

Generic Formal Type with 'Image Attribute

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Wed, 6 Jun 2018 15:03:22 +0200
Subject: Generic formal type with 'Image
Newsgroups: comp.lang.ada

I'm pretty sure the answer is "no", but just in case:

Is there a formal for a generic that serves for any type that has a predefined 'Image?

The purpose is to avoid:

```
generic
   type Printable is ...
    -- What should go here?
   with function Image (P : Printable) return
String is <>;
  package
```

and then have to pass the 'Image attribute as the Image function in all instantiations.

The closest thing I can think of is (<>) but that won't do for floating point types.

I understand this is an unusually narrow case (I need a generic for many numeric types, both discrete and floating, and this would save me some typing -- that I have already spent here anyway.)

With these issues I feel a kind of overlap/missed connection between attributes and interfaces.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 6 Jun 2018 15:34:02 -0500
Subject: Re: Generic formal type with 'Image
Newsgroups: comp.lang.ada

[...]

Ada 2020 is likely to allow 'Image on all types, and also to allow redefining it similarly to the way one redefines streaming. I say likely because that's still being worked on, and with the deadlines rapidly approaching, I can't guarantee anything will get finished and thus included unless it is already finished.

Parsing JSON with GNATCOLL

From: eduardsapotski@gmail.com
Date: Thu, 7 Jun 2018 22:52:54 -0700
Subject: GNATCOLL JSON Parsing
Newsgroups: comp.lang.ada

I try understand parsing JSON in Ada.

For example:

Have web-api that gives simple JSON: http://api.exmo.com/v1/trades/?pair=BTC_USD&limit=10

I need to save this data to database.

Created type:

```
   type Money is delta 0.00000001 range 0.0
.. 9_999_999_999.9;
   type UTC_Date is range 1_500_000_000
.. 3_000_000_000;

   type Trade is record
      Trade_Id   : Integer;
      Pair       : Unbounded_String;
      Trade_Type : Unbounded_String;
      Price      : Money;
      Quantity   : Money;
      Amount     : Money;
      Date       : UTC_Date;
      Saved      : Boolean;
   end record;
```

Created collection:

```
   package Vector_Trades is new
Ada.Containers.Vectors(Natural, Trade);

   Trades : Vector_Trades.Vector;
```

Receive data:

```
   JSON : Unbounded_String;

   JSON := To_Unbounded_String(
             AWS.Response.Message_Body
             (AWS.Client.Get (URL =>
             "http://api.exmo.com/
             v1/trades/?pair=BTC_USD
             &limit=10")));
```

What to do next? How to get list of objects from the JSON-text?

How to save data to database already understood.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 8 Jun 2018 11:35:19 +0200
Subject: Re: GNATCOLL JSON Parsing
Newsgroups: comp.lang.ada

> [...]

Something like (not tested):

```
   declare
      use GNATCOLL.JSON;
      Current_Item, Reply : JSON_Value :=
             Create;
      BTC_Array  : JSON_Array :=
             Empty_Array;
   begin
      Reply := Read (Strm =>
             AWS.Response.Message_Body
             (AWS_Reply),
                Filename => "");

      if Reply.Has_Field ("BTC_USD") then
         BTC_Array := Reply.Get("BTC_USD");
         if Length (BTC_Array) > 0 then
            for I in 1 .. Length (BTC_Array) loop
               Current_Item := Get (BTC_Array, I);
               declare
                  Trade_ID : Integer := 0;
               begin
                  if Current_Item.Has_Field
                  ("tradeid") then
                     Trade_ID := Current_Item.Get
                  ("tradeid");
                  end if;
                  ...
               end;
            end loop;
         end if;
      end if;
   end;
```

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Fri, 8 Jun 2018 05:00:27 -0700
Subject: Re: GNATCOLL JSON Parsing
Newsgroups: comp.lang.ada

> [...]

You might find the library <https://github.com/persan/gnatcoll-json> useful. It contains JSON supPort for most Ada.Containers.* packages and some examples.

Forcing Explicit Initialisation

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Thu, 14 Jun 2018 17:37:26 +0200
Subject: Unknown constraints and type composition
Newsgroups: comp.lang.ada

I think I have read somewhere that types with unknown constraints are a good way of ensuring you (or your users) don't end with uninitialized values:

```
   types Whatever (<>) is [limited] private;
   function Create return Whatever;
```

This seems nice at first sight but when these types have any likelihood of ending as members of another type you will hit the "unconstrained member" problem.

A workaround then is to use a Indefinite_Holder, but that's an imposition on your clients (ugly). If your type is furthermore limited, then you must use pointers and consider providing controlledness and deallocation in the enclosing type (uglier).

Right now I'm on the point of a new design where I have many interrelated types that require initialization calls (it's a C binding). And, as always, I'm unsure of the way to go, or if I'm missing another technique without shortcomings. Your thoughts if you have any on this issue are much appreciated.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 14 Jun 2018 18:19:57 +0200
Subject: Re: Unknown constraints and type composition
Newsgroups: comp.lang.ada

> [...]

As well as problems with publicly derived types.

[...]

In large projects instead of holder I use a reference-counted controlled handle. The target's type declaration goes into private packages. The handles go to the public interface packages. It is tedious, but it the only working method if you want to enforce construction and hide implementation.

From: Simon Belmont <sbelmont700@gmail.com>
Date: Thu, 14 Jun 2018 09:58:12 -0700
Subject: Re: Unknown constraints and type composition
Newsgroups: comp.lang.ada

One approach would be to use coextensions, assuming you are aware of all that entails, e.g.:

```
   type Inner (<>) is private;
   function Create_Inner return Inner;
...
   type Outer (<>) is private;
   function Create_Outer return Outer;

private
   type Outer (x : access Inner) is null
record;
...
   function Create_Outer return Outer is
   begin
      return Outer (x => new
            Inner'(Create_Inner));
   end Create;
```

And, assuming the compiler follows the advice (and is bug free :/) everything should work itself out safely and neatly. Though a portable way to do this would be nice.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Thu, 14 Jun 2018 19:53:07 +0200
Subject: Re: Unknown constraints and type composition
Newsgroups: comp.lang.ada

> [...] types with unknown constraints are a good way of ensuring you (or your users) don't end with uninitialized values [...]

It's one way. There are others that might be better. Unknown discriminants are more for generic formal types, to show that the generic accepts indefinite actual types.

One way to deal with this is to make the full type a record with reasonable defaults for all the components. This works for all versions of the language.

Another is to make the type a descendant of [Limited_}Controlled and override Initialize. This works for Ada 95 and later.

Another way is to have

```
   function Initialized (Thing : Whatever)
         return Boolean;
```

that returns True if its parameter has been initialized. Have a Dynamic_Predicate on Whatever that Initialized returns True. Have a postcondition on your New_Whatever function that its return value is Initialized. Make the full type a record with an Initialized component

default initialized to False. This only works for Ada 2012.

Another is to leave off the predicate, and instead give all operations on the type the precondition that the value is Initialized. This only works for Ada 2012, but it can be emulated in any version of Ada with manual checks of the precondition.

> [...]

After having made an effort to make the type indefinite, you should not be surprised that it's an indefinite type.

Part of design is to try to anticipate all reasonable uses for a type, and choose an approach that works for them all. If this is a reasonable use of the type, then unknown discriminants is not a suitable approach.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 15 Jun 2018 07:13:38 +0200
Subject: Re: Unknown constraints and type composition
Newsgroups: comp.lang.ada

> [...] One way to deal with this is to make the full type a record with reasonable defaults for all the components. [...]

Even better, the default can be a raise expression (or a function that raises an exception for pre-2012), so no uninitialized object can be created. This is a run-time check, but a decent compiler would warn you at compile time.

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Sun, 08 Jul 2018 15:53:44 +0200
Subject: Re: Unknown constraints and type composition
Newsgroups: comp.lang.ada

> Even better, the default can be a raise expression [...]

This does not give a warning at compile time with FSF GNAT 8 (Debian/sid), nor with GNAT CE 2018, but you get the correct run-time error with both compilers:

```
private with Ada.Strings.Unbounded;

package Initialisation_Required is
   type Instance is private;
   function Create (Name : in String)
         return Instance;
private
   type Instance isrecord
      Name : Ada.Strings.Unbounded.
         Unbounded_String := raise
         Constraint_Error
          with "Uninitialised object.";
   end record;
end Initialisation_Required;

package body Initialisation_Required is
   function Create (Name : in String)
return Instance is
   begin
      return R : Instance do
         R.Name := Ada.Strings.Unbounded.
            To_Unbounded_String (Name);
      end return;
   end Create;

end Initialisation_Required;

with Initialisation_Required;
procedure Demo is
   O : Initialisation_Required.Instance :=
         Initialisation_Required.
         Create (Name => "Hello");
begin
   O := Initialisation_Required.Create
         (Name => "Hello");
end Demo;
```

Another problem with this is that you can't wait until you leave your internal constructor function, before the initialisation has to happen.

With the example above, you get your exception already at "return R : Instance do", which isn't what we want. (Easy to work around, but something you should be aware of.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 14 Jun 2018 16:28:40 -0500
Subject: Re: Unknown constraints and type composition
Newsgroups: comp.lang.ada

> [...]

If there was a technique without shortcomings, we wouldn't need any other options!!

The other option, of course, is to ensure that the default initialized object is "meaningful" in some sense. Most of my objects default initialize to a state that causes most operations on them to raise an exception ("Not_Valid_Error" in Claw). I don't think it is reasonable to try to make objects valid 100% of the time, that forces all of your operations into the straitjacket of Ada scoping.

That of course has the downside of making the checks dynamic. In the case of Claw, there's no choice anyway (a Window object can disappear due to an action that comes from the program's user [clicking on the close button], not the program's code, so nothing really can be determined statically). I suspect that is somewhat true of most real systems.

Subpools

From: Simon Belmont <sbelmont700@gmail.com>
Date: Thu, 14 Jun 2018 06:48:26 -0700
Subject: Comprehending subpools
Newsgroups: comp.lang.ada

I've been trying for what I guess is six years now to figure out subpools, and I just can't seem to make heads or tails of it. Yes, I understand it inasmuch as it means you can deallocate multiple objects at once with proper finalization, but it seems like a hell of a lot of work went into it, with multiple AI's, hundreds of comments, and what has to be thousands of man-hours for a feature that seems niche, at best. The AI's are filled with big talk and grand plans (the phrase "portable garbage collecting pool" was uttered), and even the "Controlled" pragma was marked as being supplanted by subpools, but what made it into the language seems, well, not much better than what was there already. But it wouldn't be the first Ada feature that was too complex for programmers to understand, so please point out where I have gone off the rails.

The rationale says "this is far safer and often cheaper than trying to associate lifetimes with individual objects". But is it really?

Deallocating subpools is still just as unchecked as deallocating an individual objects, and it's not like you get partial credit for dangling references. Deallocating a subpool with a reference into it still hanging around is just as unsafe as regular Unchecked_Deallocation, so you still have the same old problem of either limiting the scope, or reference counting everything. And practically, if you are going to reference count it, it's going to be some generic, reusable package that either works right for everything or nothing at all, which is exactly as safe/unsafe as it was before.

Moreover, reference counting in a world of subpools is even harder, because not only do you have to worry about the objects in the subpool, but the subpool handle itself. So it becomes a *more* complex task of ensuring that a) all the references into the subpool are gone and B) all references to the subpool itself are gone. The reference counted values would need some mechanism to know which subpool they came from, which probably means the subpool handle, which now also has to be reference counted, so you have shared pointers inside of shared pointers, which i would classify as "at best equally complex and error prone" instead of "far safer". You could argue you might save some bytes by only having a single total reference count instead of many, but that is probably offset by needing to save the subpool reference anyway. And of course this means reference counted subpool values would be different than normal reference counted values, which means a THIRD type of incompatible pointer running around; i.e. you still can't have a set of references that can hold 'regular' (accessibility-checked) access values, 'standard' reference counted values, and subpool counted values.

Alright, so maybe this is intended for the case where you can control the entire scope, and not have to worry about passing them around. But in those cases,

can't you just declare a new access type and use 'Storage_Size (or a normal storage pool) to give you exactly that? The only reason to use a subpool is when the type has to be declared at a higher scope, which is presumably so you can pass that type around (and save it off somewhere), which demands some sort of reference counting.

So then perhaps it's "often cheaper"? It will certainly be faster to deallocate a big chunk than a bunch of little chunks, but all premature optimization platitudes aside, I can honestly not think of a time when the speed of deallocations concerned me in the least, as either programmer or user. Normally this happens when the program (or significant portion of it) is over, at which point who cares how long some thread runs in the background cleaning up storage? Even the given example of a retail shopping web site seems forced; who worries about how long a web server takes to deallocate old shopping cart data after the user has closed his browser? If anything that example demonstrates the need for some form of automated garbage collection, not optimizing the manual method.

Moreover, even in a supposed use-case where speed DOES matter, the whole point of this change was to do it in a way that allows for finalization of the objects; otherwise the pre-existing Mark/Release pool was sufficient. But if you need a subpool because all these objects are going to have complex finalization, isn't that almost certainly going to be the bottleneck? The runtime still has to walk through the list of objects, one-by-one, and run their big, slow, complex Finalize procedures. So the idea that you can just 'adjust a pointer' and be done with it really doesn't hold water either.

So at the end of the day, I can't see how much of any of this is better than what was already there, and certainly not to the extent it was worth the apparently enormous amount of time spent, especially when there are other things begging for improvement. It seems like it would have been much more efficient to just make the mark/release pool a standard library package instead of an example, and add a rule that saying that the implementation needs to finalize the objects within.

I have to assume I just haven't had that "a-ha!" moment that readjusts my old way of thinking, so does someone actually have a real, serious, concrete example of something using subpools that warrants the time and expense that this took? I'm all for revising the memory management facilities in Ada (I long for the day when you can use raw access values from a reference-counting pool, a mark/sweep pool, and those generated from 'Access interchangeably), but this just doesn't seem to be it.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 14 Jun 2018 16:21:59 -0500
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

> [...]

The use case is situations when you have separable data structures that it only makes sense to treat as a whole. Think of an expression tree in a compiler. There are a lot of inter-structure links, so a reference counting scheme for every pointer doesn't work. Rather, you can use a subpool and only reference count the references to the entire tree. When that goes to zero, you use the subpool to clobber the entire structure. Alternatively, you might have weak references to the tree, that automatically get nulled when the tree is clobbered.

You can't use separate access types in cases like this, since there's lots of shared code that needs to take pointers to these trees. And you want to clobber the whole structure at once, as that reduces the possibility of dangling pointers.

Tucker originally had various weak and strong references with the subpool proposal, but those were massively complex and can easily be constructed out of existing Ada concepts. So the subpool is a tool, but it's expected to be used with programmer constructed strong and weak references - by itself, it only really provides one thing: the ability to finalize a group of objects together. (The one thing that you can't do without it.)

It *is* a niche need; personally, I think using tree containers to represent an expression tree would be a better solution to the problem given above. Those too can have dangling pointers, but only if you insist on an implementation that puts performance above safety. (Unfortunately, many users do exactly that.) It's relatively easy to detect all dangling cursors for the unbounded containers (the requirement for the packages to be Pure prevents such detection for the bounded containers, although the usual implementation of a bounded container means that such cursors still point at *something*, it's just not what you expect).

In any case, subpools precedes the tree containers, so that wasn't an option then. For me, I'd try to avoid ever writing an access type and use the containers instead (you can get dangling detection for free, and many operations -- like iteration and lookup -- never need a cursor in the first place).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 15 Jun 2018 09:15:00 +0200
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

> [...]

I think the questions rather were:

1. What is so special about arena or a mark-and-release pool that it cannot be handled by a user-defined pool in Ada 95?

2. Arena is inherently unsafe whatever implementation used. So all talk about "safety" does not make much sense.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 15 Jun 2018 17:15:21 -0500
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

> 1. What is so special about arena or a mark-and-release pool that it cannot be handled by a user-defined pool in Ada 95?

No pool that does block deallocation (not using Unchecked_Deallocation) can work properly with controlled objects. For almost all implementations, attempting to do that with controlled objects would cause the program to instantly crash because of an attempt to follow pointers that no longer exist.

Since virtually all memory management schemes in Ada use controlled types or some language-defined equivalent, that essentially means that you would be so limited in what you can put into such a pool that it is close to useless.

> 2. Arena is inherently unsafe whatever implementation used. So all talk about "safety" does not make much sense.

"Safety" in this case is related to properly handling controlled types. With that, one can construct properly working strong and weak references and other safe memory management structures that will work on essentially any Ada objects. Without it, you have no chance of any safe memory management.

The basic idea is that one manages the "strong" references to an arena (such as the reference to the root of a tree), and when they are all gone, one can safety destroy the arena. The weak references aren't managed for that purpose, but don't become erroneous when the arena is destroyed, but rather just get (effectively) nulled out.

One could implement the containers this way, such that when a container is destroyed, that the entire arena of nodes is immediately reclaimed. (There's no legal references into the container at that point.)

In any case, a subpool by itself doesn't provide any safety; it's just a building block to be used to provide such safety. All of the other things needed to build such abstractions already existed in Ada, the only thing missing was an ability to finalize all of the objects in an arena (subpool) at once.

Draw your own conclusions as to how valuable (or not) that is.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 16 Jun 2018 09:36:19 +0200
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

> [...]

Right, it is so special case that I join OP wondering why this was paid any attention at all. Normally the schema is reverse, the objects must go when the arena goes, if any safety could be added, then a linked list of objects to finalize [prematurely, BTW] as it is done in other cases.

From: Simon Belmont <sbelmont700@gmail.com>
Date: Mon, 18 Jun 2018 17:32:23 -0700
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

I guess it's just a case of me reading too much into things. The rationale declares subpools "a major new facility", but I just couldn't (and perhaps still can't) see a niche feature as being worth all the time and trouble. When people say "far safer" i think of code that doesn't have to be prefixed with Unchecked_* at all, not just "you have to call it less".

And sure, finalization is of course important, but subpools seem almost specifically engineered to solve one problem that one person had writing one type of program, and not a general-purpose building block (which happily most features in Ada are). Which is fine for small features that are relatively easy, but just judging from the AI text, subpools seems to be the biggest change to 2012 second only to contracts, and it mostly seems, well, wasted. It doesn't appear the default pool has to support them (?), so step one to using a subpool is to go and implement a pool-with-subpools and hardcode your program to use it, and that's a high barrier to entry even when it's warranted. And when there are so many other things developers on CLA are always clamoring for (<cough>constructors<cough>), it all seems like an odd way to focus energies. Not to be flippant, but my kingdom for a do loop...and 'do' is already a reserved word!

I'd rather pull all the nonsense of wrapping access values in controlled types out of the client in the first place and put it into the pool itself (a callback passed to allocate or something?), instead of just solving the problems piecemeal. Having to use controlled types for memory management is the problem IMHO. Let code work with access values directly and leave it to the pool they came from to decide how and when to clean it up.

I suppose I was just hoping for more. I would, however, be interested to hear examples of how other people have found them useful in their own code (outside of compiler ASTs) to help foster my imagination of what else can be done with them.

Thank you again for the responses and continued support.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 29 Jun 2018 14:57:08 -0500
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

> [...] Let code work with access values directly and leave it to the pool they came from to decide how and when to clean it up.

That was my original idea for what eventually became generalized references. The problem being that you have to lie about the specification of pools for that to work (the "System.Address" parameters become handles that you pass into a dereferencer). And magic was required for the call-back needed to tell the pool when the dereference wasn't needed anymore. Most readers couldn't wrap their heads around either part of that.

Using a totally different specification for a new kind of pool wasn't appealing, as it would mean having to support multiple ways of doing the same thing.

[...]

P.S. The original driving force for subpools came from Tucker Taft and Bob Duff, who had used a system like this when implementing the tool now known as CodePeer. The original proposal was ten times more complicated, containing strong and weak references, and automated deallocation mechanisms. All of these can easily be constructed with the building blocks available, and trying to support all of that would have taken a lot more effort.

I've never seen much value for arena memory management myself, but I prefer to hide access types as much as possible with almost no visible surface. In that case, all of the memory management belongs to the objects, and that tends to require separate management for each object.

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Fri, 29 Jun 2018 15:42:31 -0700
Subject: Re: Comprehending subpools
Newsgroups: comp.lang.ada

> [...]

Hm, maybe they could be useful for some sort of distributed system? I mean if you're considering a shared memory-space (like, say, IEEE 1394) then disconnection of a device and reclaiming the address-values ala arena management seem to be fairly analogous.

Generics

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Sun, 24 Jun 2018 10:19:32 +0100
Subject: Re: Ada Successor Language
Newsgroups: comp.lang.ada

> [...]

You really need to understand the rules: "assume the best" in generic specifications, "assume the worse" in generic bodies.

I.e. a generic spec is legal if there is at least one legal instantiation. This is because the user is assumed to see the specification, and therefore understand why some instantiation is rejected.

A generic body is illegal if there is at least one illegal instantiation. This is because the user is not assumed to see the body of a generic.

Hint: a generic can be made legal by moving some declarations from the body to the specification (even into the private part).

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.

2018

October 02-05 — 37th IEEE **International Symposium on Reliable Distributed Systems** (SRDS'2018), Salvador, Bahia, Brazil. Topics include: dependability, security and privacy of distributed systems; methods and tools for the design, implementation, verification, validation and benchmarking of dependable and secure applications, middleware and operating systems; etc.

October 04-05 — 8th **Workshop on Model-Based Design of Cyber Physical Systems** (CyPhy'2018), Torino, Italy. In conjunction with ESWEEK 2018.

☺ October 10-12 — 26th **International Conference on Real-Time Networks and Systems** (RTNS'2018), Poitiers, France. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, multimedia), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), hypervisors), formal specification and verification, real-time distributed systems (fault tolerance, task/messages allocation, ...), etc.

October 11-12 — 12th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2018), Oulu, Finland. Topics include: the strengths and weaknesses of software engineering technologies and methods from a strong empirical viewpoint, including quantitative, qualitative, and mixed studies; case studies, action research, and field studies; replication of empirical studies and families of studies; mining software engineering repositories; empirically-based decision making; assessing the benefits/costs associated with using certain development technologies; industrial experience, software project experience, and knowledge management; software technology transfer to industry; etc.

October 15-18 — 29th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2018), Memphis, Tennessee, USA. Topics include: innovative techniques and tools for assessing, predicting, and improving the reliability, safety, and security of software products; reliability, availability and safety of software systems; validation and verification; faults, errors, failures, defects, bugs; software quality and productivity; software security; dependability, survivability, fault tolerance and resilience of software systems; systems (hardware + software) reliability engineering; open source software reliability engineering; supporting tools and automation; industry best practices; virtualization and software reliability; empirical studies of any of the above topics; software standards; etc.

October 16-19 — 15th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2018), Stellenbosch, South Africa. Topics include: semantics of programming languages; theories of concurrency; theories of distributed computing; models of objects and components; timed, hybrid, embedded and cyber-physical systems; static analysis; software verification; software testing; model checking and automated theorem proving; verified software, formalized programming theory; etc.

☺ November 04-09 — ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2018), Boston, Massachusetts, USA. Topics include: all aspects of software construction, at the intersection of programming, languages, and software engineering. Events include:

ACM SIGAda's HILT workshop (High Integrity Language Technology for Cybersecurity in Real-Time and Safety-Critical Systems).

Nov 05-06 11th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2018). Topics include: areas ranging from theoretical and conceptual contributions, to tools, techniques, and frameworks in the domain of software language engineering; generic aspects of software languages development rather than aspects of engineering a specific language; software language design and implementation; software language validation; software language integration and composition; software language maintenance (software language reuse, language evolution, language families and variability); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.

November 04-09 12th **Joint European Meeting of the Software Engineering Conference** and the ACM SIGSOFT **Symposium on the Foundations of Software Engineering** (ESEC/FSE'2018), Orlando, Florida, USA. Topics include: architecture and design; components, services, and middleware; debugging; dependability, safety, and reliability; development tools and environments; distributed, parallel, and concurrent software; education; embedded and real-time software; empirical software engineering; formal methods, including languages, methods, and tools; model-driven software engineering; processes and workflows; program analysis; program comprehension and visualization; refactoring; reverse engineering; safety-critical systems; scientific computing; security and privacy; software economics and metrics; software evolution and maintenance; software modularity; software product lines; software reuse; testing; traceability; etc.

♦ ☺ Nov 05-06 ACM SIGAda's **High Integrity Language Technology International Workshop on Languages and Tools for Ensuring Cyber-Resilience in Critical Software-Intensive Systems** (HILT'2018), Boston, Massachusetts, USA. Co-located with SPLASH 2018. Organized by ACM SIGAda. Topics include: language features that can be used to build security and/or safety into software-intensive systems; extending contract-based programming to specifying security resistance and resilience properties as well as safety and/or correctness properties; modeling and/or programming language features and analysis techniques that aid in code analysis and verification and that increase the level of abstraction and expressiveness; language features that support continuous requirements maturation to support evolving needs, particularly in cyber-physical systems, while ensuring that security and safety properties are preserved; etc.

November 10-13 18th **International Conference on Runtime Verification** (RV'2018), Limassol, Cyprus. Topics include: monitoring and analysis of the runtime behaviour of software and hardware systems. Application areas include cyber-physical systems, safety/mission-critical systems, enterprise and systems software, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy.

November 26-30 21st **Brazilian Symposium on Formal Methods** (SBMF'2018), Salvador-BA, Brazil. Topics include: techniques and methodologies (such as model-driven engineering, development methodologies with formal foundations, software evolution based on formal methods, ...); specification and modeling languages (such as well-founded specification and design languages, formal aspects of popular languages, code generation, formal methods of programming paradigms (such as objects, aspects, and component), formal methods for real-time, hybrid, and safety-critical systems, ...); theoretical foundations (such as models of concurrency, ...); verification and validation (such as abstraction, modularization and refinement techniques, correctness by construction, model checking, static analysis, formal techniques for software testing, software certification, ...); experience reports regarding teaching formal methods; applications (such as experience reports on the use of formal methods, industrial case studies, tool support).

November 28-30 19th **International Conference on Product-Focused Software Process Improvement** (PROFES'2018), Wolfsburg, Germany. Topics include: experiences, ideas, innovations, as well as concerns related to professional software development and process improvement driven by product and service quality needs.

December 03-05 16th **Asian Symposium on Programming Languages and Systems** (APLAS'2018), Wellington, New Zealand. Topics include: foundational and practical issues broadly spanning the areas of programming

languages and systems, such as semantics, design of languages and type systems, domain-specific languages, compilers, interpreters, abstract machines, program analysis, verification, model-checking, software security, concurrency and parallelism, tools and environments for programming and implementation, future directions of programming, addressing rapid changes of underlying computing platforms, etc.

December 04-07 25th **Asia-Pacific Software Engineering Conference** (APSEC'2018), Nara, Japan. Topics include: agile methodologies, component-based software engineering, cyber-physical systems and Internet of Things, debugging and fault localization, embedded real-time systems, formal methods, middleware, model-driven and domain-specific engineering, open source development, parallel, distributed, and concurrent systems, programming languages and systems, refactoring, reverse engineering, security, reliability, and privacy, software architecture, modelling and design, software comprehension, software engineering education, software engineering tools and environments, software maintenance and evolution, software product-line engineering, software reuse, software repository mining, testing, verification, and validation, etc. Deadline for submissions: October 1, 2018 (posters).

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

☺ December 11-13 24th IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2018), Sentosa, Singapore. Topics include: parallel and distributed applications and algorithms, middleware, security and privacy, dependable and trustworthy computing and systems, cyber-physical systems, embedded systems, real-time systems, multi-core and multithreaded architectures, scheduling, etc.

☺ December 11-14 39th IEEE **Real-Time Systems Symposium** (RTSS'2018), Nashville, Tennesse, USA. Topics include: all aspects of real-time systems, including theory, design, analysis, implementation, evaluation, and experience.

December 12-14 23rd **International Conference on Engineering of Complex Computer Systems** (ICECCS'2018), Melbourne, Australia. Topics include: verification and validation, security and privacy of complex systems, model-driven development, reverse engineering and refactoring, software architecture, design by contract, agile methods, safety-critical & fault-tolerant architectures, real-time and embedded systems, cyber-physical systems and Internet of Things (IoT), tools and tool integration, industrial case studies, etc.

2019

January 08-11 31st **Conference on Software Engineering Education and Training** (CSEET'2019), Grand Wailea, Maui, USA. Topics include: curriculum development; empirical studies; personal or institutional experience; team development; software assurance, quality, and reliability education; methodological aspects of software engineering education; global and distributed software development; open source in education; cooperation between industry and academia; etc.

January 15-18 11th **Software Quality Days Conference** (SWQD'2019), Vienna, Austria. Topics include: improvement of software development methods and processes; testing and quality assurance of software and software-intensive systems; domain specific quality issues such as embedded, medical, automotive systems; novel trends in software quality; etc.

March 18-21 25th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2019), Utrecht, the Netherlands. Deadline for submissions: October 2, 2018 (papers).

March 25-28 14th **European Conference on Computer Systems** (EuroSys'2019), Dresden, Germany. Topics include: all areas of computer systems research; such as distributed systems; language support and runtime systems; systems security and privacy; dependable systems; parallelism, concurrency, and multicore systems; real-time, embedded, and cyber-physical systems; tracing, analysis, verification, and transformation of systems; etc. Deadline for submissions: October 1, 2018 (full papers), October 10, 2018 (workshops), January 23, 2019 (posters).

March 25-29 IEEE **International Conference on Software Architecture** (ICSA'2019), Hamburg, Germany. Topics include: model driven engineering for continuous architecting; component based software engineering and architecture design; re-factoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; preserving architecture quality throughout the system lifetime; software architecture for legacy systems and systems integration; architecting families of products; software architects roles and responsibilities; training, education, and

certification of software architects; industrial experiments and case studies; etc. Deadline for submissions: November 29, 2018 (abstracts Technical Track, New and Emerging Ideas, Software Architecture in Practice, Tool Demonstrations Track), December 03, 2018 (abstracts Early Career Researchers Forum), December 06, 2018 (papers Technical Track, New and Emerging Ideas, Software Architecture in Practice, Tool Demonstrations Track), December 10, 2018 (submissions Early Career Researchers Forum), January 17, 2019 (workshop papers), January 25, 2019 (tutorials). Deadline for early registration: February 28, 2019.

March 25-29 **Design, Automation and Test in Europe Conference** (DATE'2019), Firenze Fiera, Fortezza da Basso, Florence, Italy. Event includes: tracks on design methods & tools, application design, test and dependability, embedded and cyber-physical systems.

☺ April 01-04 **International Conference on the Art, Science, and Engineering of Programming** (Programming'2019), Genova, Italy. Topics include: programming practice and experience; general-purpose programming; distributed systems programming; parallel and multi-core programming; security programming; interpreters, virtual machines and compilers; modularity and separation of concerns; model-based development; testing and debugging; program verification; programming education; programming environments; etc.

April 07-11 10th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2019), Mumbai, India. Deadline for submissions: October 12, 2018 (workshops), October 13, 2018 (research and industrial/experience abstracts), October 15, 2018 (research and industrial/experience papers), December 14, 2018 (artifact registration), December 22, 2018 (artifacts), January 11, 2019 (work-in-progress/vision papers), January 14, 2019 (posters/demos, tutorials).

April 15-18 12th **Cyber-Physical Systems Week** (CPS Week'2019), Montreal, Canada.

☺ April 16-18 25th IEEE **Real-Time and Embedded Systems and Applications Symposium** (RTAS'2018). Topics include: research related to embedded systems or timing issues ranging from traditional hard real-time systems to embedded systems without explicit timing requirements, including latency-sensitive systems with informal or soft real-time requirements; original systems and applications, case studies, methodologies, and applied algorithms that contribute to the state of practice in the design, implementation, verification, and validation of embedded systems and time-sensitive systems (of any size); etc. Deadline for submissions: October 17, 2018 (papers), January 31, 2019 (brief presentations, demos).

April 16-18 10th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2019). Topics include: development of technologies, tools, and architectures for building CPS systems; design, implementation, and investigation of CPS applications; etc. Deadline for submissions: October 17, 2018 (full papers).

April 06-12 22nd **European Joint Conferences on Theory and Practice of Software** (ETAPS'2019), Prague, Czech Republic. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems).

April 08-12 34th ACM **Symposium on Applied Computing** (SAC'2019), Limassol, Cyprus.

☺ April 08-12 **Track on Programming Languages** (PL'2019). Topics include: technical ideas and experiences relating to implementation and application of programming languages, such as compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, etc.

April 08-12 **Track on Software Verification and Testing** (SVT'2019). 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, model checking, correct by construction development, model-based testing, software testing, static and dynamic analysis, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging,

verification and validation of large scale software systems, real world applications and case studies applying software testing and verification, etc.

April 08-12 14th **Track on Dependable, Adaptive, and Trustworthy Distributed Systems** (DADS'2019). Topics include: Dependable, Adaptive, and trustworthy Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.

April 08-12 **Track on Next Generation Programming Paradigms and Systems** (NGPS'2019). Topics include: runtime verification and monitoring; secure and dependable software; formal models and verification; design, implementation and optimization of high-level programming languages; middleware platforms; scenarios, case studies and experience reports on innovative applications; high-level parallel programming; distributed systems and concurrency; development tools; security, trust and privacy management; etc.

April 08-12 **Embedded Systems Track** (EMBS'2019). Topics include: verification, validation, testing, debugging, and performance analysis of embedded systems; cyber physical systems; multicore, SoC-based, and heterogeneous embedded systems and applications; multithreading in embedded systems design and development; compilation strategies, code transformation and parallelization for embedded systems; reliability in embedded computing and systems; security within embedded systems and embedded systems for security; safety-critical embedded systems; case studies; etc.

May 25-31 41st **International Conference on Software Engineering** (ICSE'2019), Montréal, Québec, Canada. Theme: "The next 50 years for Software Engineering". Deadline for submissions: October 1, 2018 (IEEE TCSE Harlan Mills Award nominations, Software Engineering in Practice, Software Engineering in Society, Software Engineering Education and Training, new ideas and emerging results, demonstrations), October 10, 2018 (workshops), November 19, 2018 (Doctoral Symposium), November 30, 2018 (Journal-First papers), January 7, 2019 (ACM Student Research Competition), February 1, 2019 (workshop papers), February 7, 2019 (student volunteers).

♦ June 10-14 Ada-Europe 24th **International Conference on Reliable Software Technologies** (Ada-Europe 2019), Warsaw, Poland. Sponsored by Ada-Europe, in cooperation (pending) with ACM SIGAda, SIGBED, SIGPLAN, and the Ada Resource Association (ARA). Deadline for submissions: February 14, 2019 (regular papers, industrial presentation outlines, tutorial and workshop proposals).

☺ July 33rd **European Conference on Object-Oriented Programming** (ECOOP'2019), London, England. Topics include: original and unpublished results on any Programming Languages topic. Deadline for submissions: January 11, 2019 (papers).

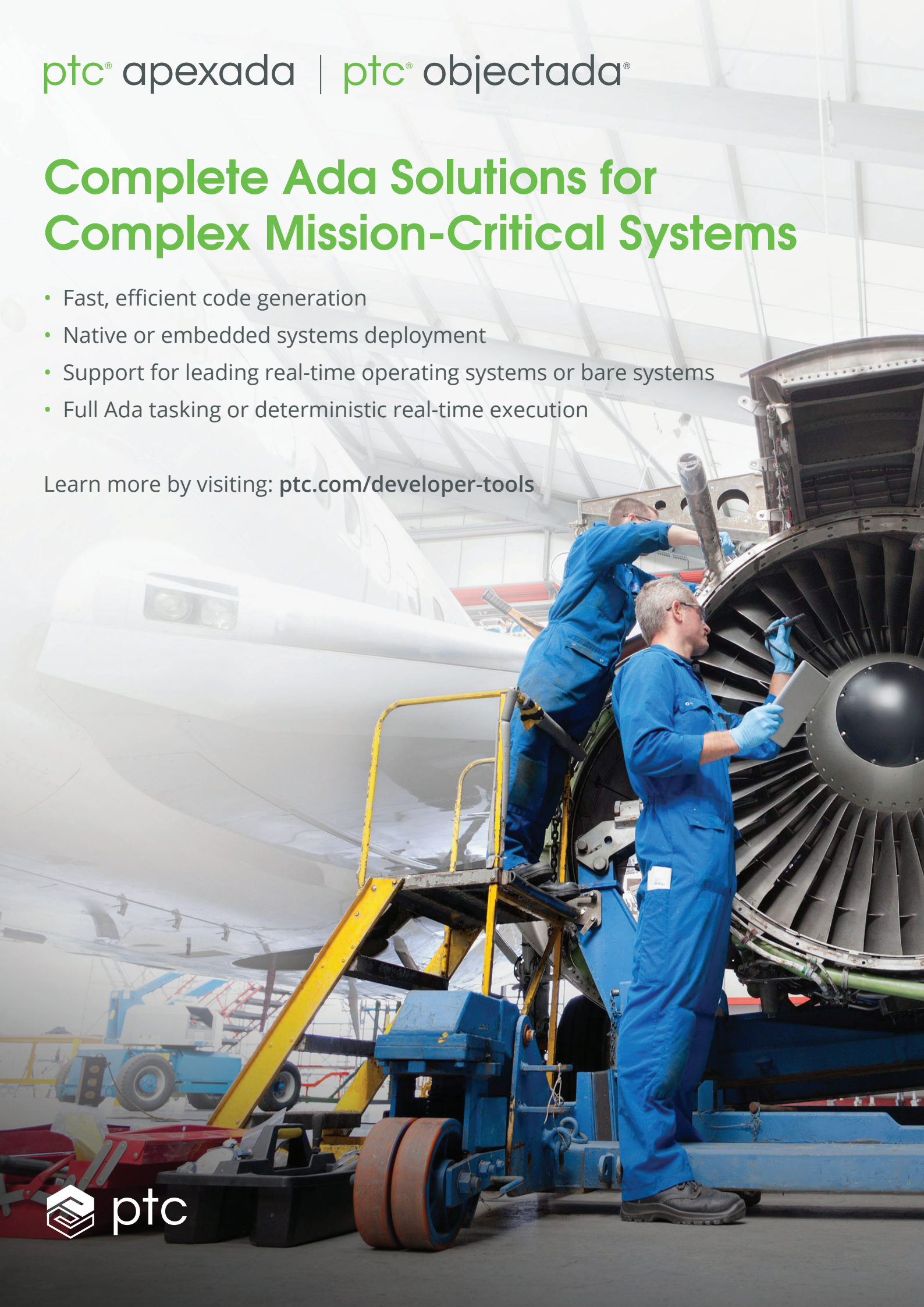
December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

ptc® apexada | ptc® objectada®

Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: **ptc.com/developer-tools**



ptc

HILT 2018 Workshop on *Languages and Tools for Ensuring Cyber-Resilience in Critical Software-Intensive Systems*

As part of SPLASH 2018, November 5 & 6, 2018, Boston, MA, USA

Sponsored by ACM SIGAda

This is the fifth in the HILT series of conferences and workshops focused on the use of *High Integrity Language Technology* to address challenging issues in the engineering of software-intensive critical systems. HILT 2018 will focus on addressing cybersecurity and cyber-resilience issues that arise in real-time, embedded, and/or safety-critical systems, where such a system must be trusted to maintain a continual delivery of services, as well as ensuring safety in its operations. Such needs have common goals and shared strategies, tools, and techniques, recognizing the multiple interactions between security and safety.

Keynote speakers:

Bob Martin, MITRE

Common Vulnerabilities Enumeration (CVE), Common Weakness Enumeration (CWE), and Common Quality Enumeration (CQE) – Attempting to systematically catalog the safety and security challenges for modern, networked, software-intensive systems.

Ray Richards, DARPA

DARPA's new Cyber-Assured Systems Engineering (CASE) Program – Motivations, Challenges, and Technical Approaches to addressing cyber-resilience in critical software-intensive systems

This workshop is designed as a forum for communities of researchers and practitioners from academic, industrial, and governmental settings, to come together, share experiences, and forge partnerships focused on integrating and deploying tool and language combinations to address the challenges of building cyber-resilient software-intensive systems. The workshop will be a combination of presentations and panel discussions, with a number of invited speakers.

Tentative HILT 2018 Schedule:

Time	Monday November 5th	Tuesday November 6th
8:30AM-9:00AM	Welcome to HILT 2018	ACM SIGAda Awards
9:00AM-10:00AM	Bob Martin, *MITRE,* CVE, CWE, CQE, and all that	Ray Richards, *DARPA,* DARPA CASE program, motivation and challenges
10:00AM-10:30AM	*Coffee Break*	*Coffee Break*
10:30AM-11:15AM	David Wheeler, *IDA*, Approaches to Cyber-Resilience	Lucas Wagner, *Rockwell Collins,* SpeAR – Using a formal specification language for security
11:15AM-12:00PM	Stephen Chong, *Harvard Univ.,* Language-Based Security	Sam Procter, *SEI,* Architecture-level Security concerns in a safety-critical system
12:00PM – 1:30PM	*Lunch Break*	*Lunch Break*
1:30PM – 2:15PM	Ina Schaefer, *TU Braunschweig,* Confidentiality-by-Construction	Jeff Foster, *Tufts Univ.,* Permission systems on Android and beyond
2:15PM – 3:00PM	Panel on *Language-Based Security (X10, SPARK, Rust, SpeAR, ...)*	<TBD> Hardware support for cyber resilience
3:00PM – 3:30PM	*Coffee Break*	*Coffee Break*
3:30PM – 4:00PM	*<submitted paper#1>*	Security Tools Showcase
4:00PM – 4:30PM	*<submitted paper#2>*	
4:30PM – 5:00PM	*<submitted paper#3>*	Workshop Wrapup
5:15PM – 6:15PM	SIGAda Annual Meeting	
6:30PM – 9:00PM	*HILT Banquet at Legal Sea Foods, 26 Park Plaza, Boston*	

Workshop Co-Chairs

- Bill Bail, MITRE
- Tucker Taft, AdaCore, Inc

Organizing Committee

- Dirk Craeynest, ACM SIGAda International Representative, KU Leuven
- Drew Hamilton, Chair, ACM SIGAda, Mississippi State University, CCI
- Clyde Roby, Secretary-Treasurer, ACM SIGAda, Institute for Defense Analyses
- Alok Srivastava, Editor, ACM Ada Letters, Engility Corp.
- Ricky E. Sward, Past Chair, ACM SIGAda, MITRE

Ada-Europe

24th International Conference on Reliable Software Technologies

10-14 June 2019, Warsaw, Poland

Conference & Program Chair

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

Educational Tutorial & Workshop Chair

Dene Brown
SysAda Ltd, UK
dene.brown@sysada.co.uk

Industrial Chair

Maurizio Martignano
Spazio IT, Italy
maurizio.martignano@spazioit.com

Exhibition & Sponsorship Chair

Ahlan Marriott
White Elephant GmbH, Switzerland
software@white-elephant.ch

Publicity Chair

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

Local Chair

Maciej Sobczak
GE Aviation – EDC Warsaw, Poland
maciej.sobczak@ge.com

General Information

Ada-Europe is pleased to announce that its 24th **International Conference on Reliable Software Technologies** (Ada-Europe 2019) will take place in Warsaw, Poland. The conference schedule at its fullest includes a three-day technical program and vendor exhibition from Tuesday to Thursday, and parallel tutorials and workshops on Monday and Friday.

Schedule

14 January 2019	Submission of papers, industrial presentation outlines, tutorial and workshop proposals
1 March 2019	Notification of acceptance to all authors
16 March 2019	Camera-ready version of papers required
30 April 2019	Industrial presentations, tutorial and workshop material required

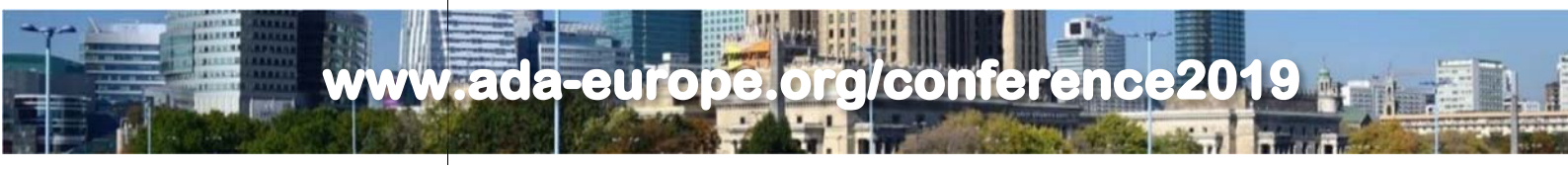
Topics

The conference is 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 from industry, academia and government organizations active in the promotion and development of reliable software technologies.

The topics of interest for the conference include but are not limited to:

- Design and Implementation of Real-Time and Embedded Systems,
- Design and Implementation of Mixed-Criticality Systems,
- Theory and Practice of High-Integrity Systems,
- Software Architectures for Reliable Systems,
- Methods and Techniques for Quality Software Development and Maintenance,
- Ada Language and Technologies,
- Mainstream and Emerging Applications with Reliability Requirements,
- Experience Reports on Reliable System Development,
- Experiences with Ada.

Refer to the conference website for the full list of topics.



Call for Regular Papers

The regular papers submitted to the conference must be original and shall undergo anonymous peer review. The corresponding authors shall submit their work by 14 January 2019. Such submissions shall be in PDF only and up to 16 LNCS-style pages in length. The authors shall use the EasyChair submission service at *https://easychair.org/conferences/?conf=adaeurope2019*.

The International Conference on Reliable Software Technologies is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the conference, both online and in print. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, strictly by 16 March 2019. For format and style guidelines, the authors should refer to *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register at least one author for the conference by that date will prevent the paper from appearing in the proceedings.

Call for Industrial Presentations

The conference seeks industrial presentations that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, of at least 1 page in length, by 14 January 2019, strictly in PDF, using the submission service at *https://easychair.org/conferences/?conf=adaeurope2019*.

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The authors of accepted contributions shall be requested to submit a 2-page abstract by 30 April 2019, for inclusion in the conference booklet, and be invited to deliver a 20-minute talk at the conference. These authors will also be required to expand their contributions into articles for publication in the *Ada User* Journal (*http://www.ada-europe.org/auj/*), as part of the proceedings of the Industrial Program of the Conference. For any further information, please contact the Industrial Chair directly.

Awards

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

Call for Educational Tutorials

The conference is seeking tutorials in the form of educational seminars including hands-on or practical demonstrations. Proposed tutorials can be from any part of the reliable software domain, they may be purely academic or from an industrial base making use of tools used in current software development environments. We are also interested in contemporary software topics, such as IoT and artificial intelligence and their application to reliability and safety.

Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), and the intended level of the tutorial (introductory, intermediate, or advanced). All proposals should be submitted by e-mail to the Educational Tutorial Chair.

The authors of accepted full-day tutorials will receive a complimentary conference registration. For half-day tutorials, this benefit is 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 Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the *Ada User Journal*.

Call for Exhibitors

The commercial exhibition will span the core 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.

Special Registration Fees

Contributors to the conference and all students will enjoy reduced registration fees.

Venue

The conference will take place in Warsaw, the capital of Poland, at the facilities of the Engineering Design Center (EDC), a partnership between General Electric (GE) Poland and the Institute of Aviation, one of the largest engineering institutions in Europe.

ARG Work in Progress II

Jeff Cousins CEng FIET

Member and former chair of the Ada Rapporteur Group; email: jeffrey.cousins@btinternet.com

Abstract

Work continues on developing the next edition of Ada. Adding support for parallelism is a priority, but there are many other small improvements, particularly in support of static analysis (mostly now approved) and in the Real-Time area (mostly relatively new proposals).

1 Introduction

This paper presents an update on the proposed changes for the next edition of Ada, provisionally called Ada 2020 in anticipation of publication in 2020. The previous paper was published in the Vol. 38, No. 1, March 2017 edition of the AUJ.

As before, Ada Issues (AIs) are first worked on and approved by the Ada Rapporteur Group (ARG). They are then passed to WG 9 (the ISO/IEC Working Group responsible for Ada) for consideration and approval before eventually being consolidated and sent to ISO for formal processing to create a revised international standard.

The deadline for public input into what new features should be incorporated into Ada 2020 passed in January 2018. The ARG held an internal ballot in May 2018 that split those AIs still outstanding approximately 50:50 between those for 2020 and those to 'hold' until a future edition. This allowed a draft Ada 2020 Scope document to be produced for WG 9, listing those Amendment AIs that the ARG hopes to include in Ada 2020 (including those already approved). The final version is imminent; but in the meantime, for progress see http://www.ada-auth.org/ai-files/grab_bag/2020-Amendments.html.

2 WG 9 approved

This section describes some of the more important changes to the language that have been approved by WG 9.

The issues listed in the previous paper as being in the pipeline, viz Add @ as an abbreviation for the LHS of an assignment (AI12-0125-3) and Update to the Fortran Annex (AI12-0058), were approved by WG 9.

Many of the changes since that paper was published are to allow more information to be specified for the support of static analysis tools, e.g.:

- Nonblocking subprograms (AI12-0064-2) and the related Specifying Nonblocking for Language Defined Units (AI12-0241);
- Max Queue Length aspect for protected types (AI12-0164);
- Stable properties of abstract data types (AI12-0187);
- Pre/Post for access-to-subprogram types (AI12-0220);
- Default_Initial_Condition for types (AI12-0265);
- Aspect No_Return for functions reprise (AI12-0269).

One of the issues is to tidy up an inconsistency in the language, i.e. Missing operations of static string types (AI12-0201).

2.1 Nonblocking subprograms (AI12-0064-2)

This adds the aspect Nonblocking and the attribute Nonblocking to Ada. These allow specifying and querying the blocking status of a subprogram. If a subprogram is declared to be nonblocking, the Ada compiler will verify that it does not execute any potentially blocking operations (other than deadlocking operations).

2.2 Static expression functions (AI12-0075)

The aspect Static is introduced. It can only be applied to an expression function, and requests that it be regarded as a static function.

If called in a context that requires the expression function to be static, such as in a static expression, then its actual parameters need to be static. For example, if we declare:

```
function If_Then_Else (Flag : Boolean;
                       X, Y : Integer) return Integer
is
  (if Flag then X else Y) with Static;
```

and then attempt to declare:

```
X : constant := If_Then_Else (True, 37, 1 / 0); -- Error.
```

we get an error at compile time since 1/0 is not a static expression.

2.3 Partial aggregate notation (AI12-0127)

A new syntactic form of aggregate, the delta_aggregate, is introduced. This allows one to update one or more fields of a composite object without having to specify every field. This will be particularly useful for postconditions, where one might want to check that only certain fields of a composite parameter had changed, for example:

```
procedure Twelfth (D : in out Date)
  with Post => D = (D'Old with delta Day => 12);
```

The values of the Year and Month components of the delta aggregate are the same as those of D'Old but the Day component is 12.

2.4 Max Queue Length aspect for protected types (AI12-0164)

The new aspect Max_Entry_Queue_Length for an entry declaration specifies the maximum number of callers allowed on that entry. This facilitates timing analysis and should be useful for new restricted tasking profiles besides Ravenscar.

Violation of this restriction results in the raising of Program_Error at the point of the call or requeue.

The value specified for the Max_Entry_Queue_Length aspect for an entry must be no higher than any specified for the corresponding type, and both must be no higher than the Max_Entry_Queue_Length partition-wide restriction. These are checked at compilation.

2.5 Stable properties of abstract data types (AI12-0187)

This adds a type and subprogram aspect Stable_Properties along with class-wide versions.

This aspect can be given on a partial view or on a full type with no partial view, and on primitive subprograms. (It is not allowed on formal types, since it is only meaningful for primitive subprograms.) The intent is that the subprogram version be used to override the type version when necessary; it is not very useful as a stand-alone aspect (it makes more sense to just modify the postcondition in that case).

The aspect determines a list of stable property functions for each primitive subprogram. The postcondition(s) of the subprogram are modified with an item that verifies that the property is unchanged for each parameter of the appropriate type, unless that property is already referenced in the explicit postcondition (or inherited postcondition, in the case of class-wide postconditions).

For example, suppose that we wish many subprograms to behave as if they have a postcondition as in:

```
procedure Put (File : in File_Type; Str : in String)
  with Pre  => Mode(File) /= In_File,
       Post => Mode(File) = Mode(File)'Old;
```

Then rather than having to repeat this postcondition for numerous subprograms, if Ada.Text_IO could be rewritten in the form:

```
package Ada.Text_IO is
  type File_Type is private
    with Stable_Properties => Is_Open, Mode;
…
```

then the declaration of Put could simply be:

```
  procedure Put (File : in File_Type; Item : in String)
  with Pre => Mode(File) /= In_File;
```

since we have stated that the Mode is a stable property.

(Sadly we cannot change Ada.Text_IO – this is just an illustrative example for the future!)

2.6 Missing operations of static string types (AI12-0201)

Relational operators and type conversions of static string types are now static.

Static membership tests for strings, e.g. S **in** "abc", were already allowed; static equality tests for strings, e.g. S = "abc", are now also allowed.

2.7 Pre/Post for access-to-subprogram types (AI12-0220)

This allows Pre and Post aspects for access-to-subprogram types, so that contract information is available when calling a subprogram indirectly via an access value, as well as (or even instead of) when called directly. For example, to check that a parameter is even:

```
type T1 is access procedure (X : Integer)
  with Pre => X mod 2 = 0;
procedure Foo (X : Integer) is ... end;
…
  Ptr1 : T1 := Foo'Access;
begin
  Ptr1.all (222); -- Precondition check performed
```

2.9 Specifying Nonblocking for Language Defined Units (AI12-0241)

Aspect Nonblocking is specified for language-defined units as needed to keep compatibility with Ada 2012. This follows on from AI12-0064-2 defining new aspect Nonblocking, this aspect now needs to be specified for many language-defined units and subprograms.

2.10 Bounded_Indefinite_Holders (AI12-0254)

A new container type, Bounded_Indefinite_Holder, allows the storage of a (single) class-wide object without the use of dynamic memory allocation, for use in safety critical environments. Rather than having a bounded indefinite variant of every container, it is envisaged that this holder container would be used as a building block, e.g. in a container of such holder containers.

Compared with the existing Indefinite_Holder, there is an additional generic parameter:

```
Max_Element_Size_in_Storage_Elements :
Storage_Count;
```

If this is exceeded, Program_Error is raised.

2.11 Default_Initial_Condition for types (AI12-0265)

A new contract aspect, Default_Initial_Condition, may be specified for a private type (or private extension). This is useful for checking that the default initialisation of an object has been performed as expected. After the successful default initialization of an object of the type, a default initial condition check is performed. In the case of a controlled type, the check is performed after the call to the type's Initialize procedure. For example:

```
package Sets is
  type Set is private
    with Default_Initial_Condition => Is_Empty (Set);
  function Is_Empty (S : Set) return Boolean;
  ...
end Sets;
```

2.12 Aspect No_Return for functions reprise (AI12-0269)

The aspect No_Return may now be specified for functions, not just procedures, but the reason that such a function never returns must be that it raises an exception (rather than containing an endless loop). The only return statements allowed in such functions are simple return statements with an expression that is a raise expression or a call of another non-returning function (or a parenthesized expression of one of these). As for procedures, there will be a check at run-time that it does not run into the final end.

3 In the pipeline

These have been approved by the ARG but have yet to be approved by WG 9. They include the first, of what is hoped to be several, AIs on parallelism. Again, one of the issues is to tidy up an inconsistency in the language, e.g. Generalize expressions that are objects (AI12-0226).

3.1 Parallel operations (AI12-0119)

Two parallel constructs are proposed, namely parallel blocks and parallel loops. A parallel block consists of a set of concurrent activities each specified by a handled sequence of statements, separated by the reserved word **and**, analogous to the syntax for a select statement where the alternatives are separated by the reserved word **or**. A parallel loop defines a loop body which is designed such that the various iterations of the loop can run concurrently. The implementation is expected to group the iterations into "chunks" to avoid creating an excessive number of physical threads of control, but each iteration is nevertheless considered for most purposes as its own separate logical thread of control.

Both constructs start with the new reserved word **parallel** to clearly indicate that these constructs are designed for parallel execution. The implementation might still not execute the constructs in parallel, but the intent is that if multiple processors are available, some or all of them should be allocated to the execution of the construct.

An example of using parallel blocks when searching a binary tree:

```
procedure Traverse (T : Expr_Ptr) is
begin
  -- Recurse down the binary tree
   if T /= null and then T.all in Binary_Operation'Class
then
     parallel do
       Traverse (T.Left);
     and
       Traverse (T.Right);
     and
          Ada.Text_IO.Put_Line ("Processing " &
                          Ada.Tags.Expanded_Name
                             (T'Tag));
     end do;
   end if;
end Traverse;
```

An example of using parallel blocks when searching a string for a particular character:

```
function Search (S : String;
                 Char : Character) return Boolean is
begin
  if S'Length <= 1000 then
    -- Sequential scan
    return (for some C of S => C = Char);
  else
    -- Parallel divide and conquer
    declare
      Mid : constant Positive := S'First + S'Length/2 – 1;
    begin
      parallel do
        for C of S(S'First .. Mid) loop
          if C = Char then
            return True; -- Terminates enclosing "do"
          end if;
        end loop;
      and
        for C of S(Mid + 1 .. S'Last) loop
          if C = Char then
            return True; -- Terminates enclosing "do"
          end if;
        end loop;
      end do;
      -- Not found
      return False;
    end;
  end if;
end Search;
```

An example of using a parallel loop when initialising an array:

```
parallel for I in Grid'Range(1) loop
   Grid(I, 1) := (for all J in Grid'Range(2) =>
                                    Grid(I,J) = True);
end loop;
```

3.2 Loop body as anonymous procedure (AI12-0189)

A loop body can be used to specify the implementation of a procedure to be passed as the actual for an access-to-subprogram parameter, when used in the context of a special kind of for-loop statement, whose iterator_specification is given by a procedure_iterator.

This can be used for iterating over Directories and Environment variables, or iterating through a map-like container over the keys. Dedicated mechanisms were proposed for these (AI12-0009 and AI12-0188, respectively), but it was considered more useful to add a more general mechanism. For example:

```
for (Name, Val) of Ada.Environment_Variables.Iterate
                                                (<>)
loop
-- "(<>)" is optional because it is the last parameter
   Put_Line (Name & " => " & Val);
end loop;
for (C : Cursor) of My_Map.Iterate loop
   Put_Line (My_Key_Type'Image (Key (C)) & " => " &
                  My_Element_Type'Image(Element (C)));
end loop;
```

3.3 Generalize expressions that are objects (AI12-0226)

A value conversion of an object is an object in order to be consistent with a qualified expression. If we have

```
Max : constant Natural := 10;
```

then the following are now all legal

```
-- Legal
Ren1 : Natural renames Max;
-- Qualified expression, legal
Ren2 : Natural renames Natural'(Max);
-- Value conversion, was illegal, now legal
Ren3 : Natural renames Natural(Max);
```

3.4 Contracts for generic formal parameters (AI12-0272)

Pre and Post are allowed on generic formal (nonabstract) subprograms. For example:

```
generic
   type Foo is ...
   with function Reduce (Obj : Foo) return Integer
      with Post => Reduce'Result in –9 .. 9;
package Gen is
   ...
end Gen;
```

In addition the new aspect Default_Initial_Condition (AI12-0265 – see above) is allowed on generic formal private types

3.5 Make subtype_mark optional in object renames (AI12-0275)

This makes the subtype_mark optional in object renaming declarations. The expression will be correctly typed as long as the right hand object_name can be resolved to only one specific type.

It has long been an irritant that writing a renaming declaration required looking up the subtype of the object, and that giving the subtype can be misleading as only the type is checked anyway.

4 The Future

4.1 What happened to the previous 'The Future'?

The proposals to add new aspects Nonblocking (AI12-0064) and Stable_Properties (AI12-0187) to aid analysis tools were successful – see above.

The proposal to add the basic syntax and semantics to support parallelism (AI12-0119 – see above) has been approved by the ARG, but the related AIs are still work in progress (though at higher priority than most):

- Global-in and global-out annotations to specify which global objects a subprogram may access, and in which mode (AI12-0079);
- Reduction Expressions (AI12-0242);
- Explicit chunk definition for parallel loops (AI12-0251-1);
- Parallel loop chunking libraries (AI12-0251-2);
- Map/Reduce Attribute (AI12-0262);
- Parallel Container Iterators (AI12-0266);
- Data race and non-blocking checks for explicit parallelism (AI12-0267).

A new container type, Bounded_Indefinite_Holder, is added – see above.

Investigations into Lambda functions (AI12-0190), Generators/co-routines (AI12-0197), and Declare expressions (AI12-0236) continue, albeit with less enthusiasm for Generators/co-routines.

The suggestions to add a new pragma Loop_Invariant and to add Function Decorators were never formally raised.

4.2 Support for Static Analysis

Ghost code (AI12-0239). "Ghost code" is code that is added to support specification and verification, typically functions (marked by the aspect Ghost) that are defined in specifications and called from preconditions and postconditions, but which do not generate any code in the final executable.

4.3 Real-Time

A number of AIs were discussed by, or arose from, the 19th International Real-Time Ada Workshop, as reported on in the Vol. 39, No. 2, March 2018 edition of the AUJ:

- Thread-safe Ada libraries (AI12-0139);
- Deadline Floor Protocol (AI12-0230);
- Compare-and-swap for atomic objects (AI12-0234);
- Admission Policy Defined for Acquiring a Protected Object Resource (AI12-0276);
- Dispatching Needs More Dispatching Points (AI12-0279);
- CPU Affinity for Protected Objects (AI12-0281);
- Atomic and Volatile generic formal types (AI12-0282).

IRTAW also proposed an extended version of the Ravenscar profile, provisionally named Jorvik, though the AIs have yet to be raised for this.

4.4 Others

The attribute 'Image is added for all types (AI12-0020). This should be a boon for debugging. A developer will be able to insert a line of Text_IO.Put for Any_Object'Image, without having to laboriously write a subprogram to output the object field by field if it is a composite object.

Defaults for generic formal parameters (AI12-0205). This would provide easier and more natural generic instantiation. These use the reserved words **or use**. For example:

```
generic
  type Item_Type is private;
  type Item_Count is range <> or use Natural;
                          -- New syntax using use
  with function "=" (L, R : in Item_Type)
                                        return Boolean;
package Lists is
  ...
end Lists;
```

This allows the instantiator to be able to provide a type for the Item_Count, but it can simply be omitted in ordinary circumstances (in which case Natural would be used).

Predefined big numbers support (AI12-0208). This defines a package Big_Numbers and various child packages to support arbitrary precision arithmetic.

Access value aliasing and parameter aliasing (AI12-0240). This is an ambitious proposal which introduces the concept of access type "ownership" such that no more than one access value can point to a given object on the heap, thus allowing automatic storage management.

User-defined literals (AI12-0249). An aspect, or aspects, will be defined to allow a type to use numeric, string, character, and/or null literals. For example:

```
type Unbounded_String is private
  with String_Literal => To_Unbounded_String;
function To_Unbounded_String
                              (Source : Wide_Wide_String)
  return Unbounded_String;

X : constant Unbounded_String := "This is a test"
```

The above declaration of X is equivalent to:

```
X : constant Unbounded_String :=
      To_Unbounded_String
            (Wide_Wide_String'("This is a test"));
```

And finally we have Iterator Filters (AI12-0250). When iterating through a container, it is often required to filter the results to only return those values that meet some condition. This proposal makes use of the keyword **when**, for example:

```
S : constant Set :=
                    (for E of C when E mod 2 = 1 => E);
```

to obtain all the odd elements of Container C.

5 Conclusions

The successful delivery of so many projects using earlier editions of Ada has demonstrated that it is already a very capable language. Ada 2020 will be a relatively modest update compared with Ada 2005 and Ada 2012, but the new features should maintain its power, in particular the support for parallelism to make use of the ever growing numbers of cores in a processor.

Revolutionize your software verification

Efficiency, Automation, Reliability

RAPITA Systems Ltd
A DANLAW Company

Unit testing · System testing · Coverage analysis · Timing analysis
V&V services · Multicore timing services · DO-178C training
Ada · C · C++

www.rapitasystems.com

C Guidelines Compliance and Deviations (the MISRA and CERT Cases)

Maurizio Martignano

Spazio IT – Soluzioni Informatiche, Via Manzoni 40, 46030 San Giorgio di Mantova, Italy; Tel: +39 0376 1434259; email: Maurizio.Martignano@spazioit.com

Abstract

C Guidelines such as MISRA and CERT define a set of rules and directives designed to help developers in writing quality code. Unfortunately, more often than not these guidelines are heavily tailored and customized before being applied to actual projects. The paper will first show some of the potential dangers that can be caused by such customizations. Then it will describe MISRA and CERT efforts in the attempt to rationalize, standardize the customization process. Finally, the paper will propose a viable approach to properly manage compliance and deviations by respecting the new MISRA and CERT indications while retaining the necessary flexibility at project level.

Keywords: C Language, MISRA, CERT, Standard, Guideline, Compliance, Conformance, Deviation.

1 Introduction

MISRA C:2012 ("guidelines for the use of the C language in critical systems") and SEI CERT C ("SEI CERT C Coding Standard") provide a set of guidelines (rules and directives – MISRA or rules and recommendations - CERT) designed to help developers in writing quality code, i.e. code that is safer, more secure, understandable and maintainable.

Obviously, it is not always possible to adhere to all these guidelines and this why in several software development projects deviations and compliance levels are established in the context of so called customization or tailoring activities.

The customization activity can be performed very formally, adhering once again to specific and additional standards, or rather informally on a project by project base, according to the specific needs and actual limitations of the project itself. The customization strategies adopted by actual projects range from not allowing any deviation to accepting every documented deviation.

Both MISRA and CERT have tried to improve this situation. MISRA in April 2016 has published a relatively new guidance, MISRA Compliance:2016 ("Achieving compliance with MISRA Coding Guidelines). CERT C Coding Standard (as well as the C++ and Java ones) in the Introduction, section 1.8 Conformance, provides a set of indications about the rules and recommendations conformance, defines a set of levels of conformance and describes a proper deviation procedure.

The paper will start by describing some examples of customizations, implemented by actual projects, and in particular will try to show that:

1. excessive tailoring of the guidelines could be a problem;
2. it is not always true that the knowledge of the language available inside projects is better than the one contained, implied by the guidelines;
3. guidelines can be used to improve the understanding of the language among unexperienced developers.

Secondly, the paper will describe in detail both MISRA and CERT standardization efforts in the areas of compliance and deviations.

Thirdly the paper will concentrate on the analysis tools and on their importance in verifying, endorsing and encouraging compliance to the guidelines.

Finally, the paper will propose a viable approach to properly manage compliance and deviations by respecting the new MISRA and CERT indications while retaining the necessary flexibility at project level.

2 Customizations Examples

Note 1: all customizations examples in this section have been "anonymized" so that the original projects cannot be identified.

Note 2: due to copyright restrictions, MISRA directives and rules descriptions have been altered. The original descriptions can be found in reference document [1].

Integral types size and sign

Involved guideline

MISRA C 2012 – Directive 4.6 (Advisory) – "*typedefs* indicating size and sign should be used instead of the basic integral types".

Description

This MISRA directive recommends to use the "new" C99 integral types contained in "stdint.h" instead of the "old" integral numerical types, e.g. *int8_t* instead of *char*, *uint8_t* instead of *unsigned char*, *int32_t* instead of int (or *long* depending on the hardware architecture), etc… This header file provides a set of *typedefs* specifying exact-width

integral types, together with their minimum and maximum values. "stdint.h" is particularly useful for embedded programming, especially in portions of code interfacing with hardware devices requiring integer data of fixed width. With the "basic" types the only guarantee about the sizes common among the various compilers is that *char* <= *int* <= *long* <= *long long*; but the actual sizes are implementation dependent.

In some projects (say Type A) this directive has been completely ignored and only the basic integral types have been used, leaving the actual sizes implicitly defined by the compiler implementation. In some other projects (say Type B) the directive has been followed to the letter. Finally, in yet some other projects (say Type C) MISRA has been customized and the directive has been adopted only "when relevant", only when "it made sense", that is only in portions of code interfacing with the hardware. In Type A and Type C projects coding "was not for everyone", coding required additional knowledge: either on the data representation implicitly used by the compiler or on the software architecture (to know when it was necessary to use the "new" C99 integral types). On the other hand, in Type B projects, coding "was for everybody", given that only the "new" integral types were allowed and there was no need to distinguish a piece of code from the others. Being knowledgeable on a given project, on a given piece of code is not just a matter of experience, but also a matter of "frequentation". A person that today knows everything on a given software system or some of its components, may lose all her knowledge in a matter of months, if not weeks when working on other assignments.

(Not NULL) Pointer Function Parameters

Involved guidelines

MISRA C 2012 – Directive 4.11 (Required) – "The validity of parameters passed to library functions shall be checked".

CERT – EXP34-C – "Do not dereference null pointers"

CERT – MEM10-C – "Define and use a pointer validation function"

Description

The "common sense" rule is that function parameters should be checked for their validity. Between the "caller" and the "callee" it is the first one which is more knowledgeable, has more information about the validity of the passed parameter. In the case of pointer function parameters, a first and simple check consists in verifying they are not NULL; both the "caller" and the "callee" have enough knowledge to perform this check. Though this check could be placed in the "caller" before the function invocation, inserting it in the "callee" simplifies the verification that all parameters have been checked. This check is usually performed via an "*if*" if the pointer parameter can only be known at execution time or via an "*assert-like*" construct when the pointer is known at design/compilation time.

Once again, in some projects (say type A) these guidelines have been ignored, and pointer function parameters are never checked. In some other projects (say Type B) the guidelines have been followed and always via an "*if*" check inside the "callee". In yet some other projects (say Type C) the difference between "*assert-like*" checks and "*if*" checks has been retained and in production/flight the "*assert-like*" checks have been removed (either via conditional compilation or actual deletion from the code). Quite often the reasoning behind non-complying with these guidelines has been performance, even if these checks do not (and cannot) consume a lot of resources. In Type B projects there is no particular decision to be taken: i.e. pointer functions parameters have to be checked, always, by the "callee", via an "*if*"-check. In Type C projects distinguishing between "*if*" and "*assert-like*" checks and deciding which checks can be safely removed in the final code could be rather difficult and require "knowledgeable" developers.

Forbidden Conversions (and then again)

Involved Guidelines

MISRA C 2012 – Rules 10.x – "Essential type model" and Rules 11.x – "Pointer type conversions"

Description

All these rules about conversions between pointer types and/or between essential types belonging to different categories or with different sizes are sound and make sense in generic, application level pieces of code. Wherever a violation is found something suspicious is taking place and most of the times it is an error.

Unfortunately, when coding low level portions of embedded systems and trying to access hardware IOs, registers, non-volatile memory, etc… it is impossible not to break one or more of these rules. E.g. the following very simple piece of code:

```
1: typedef unsigned char uint8_t;
2:
3: int main(void) {
4:
5:         uint8_t * ui8_ptr;
6:         ui8_ptr = (uint8_t *) 0x12;
7:
8:         return 0;
9: }
```

at line 6 violates Rule 11.4 "conversion between a pointer and integer type" and Rule 11.6 "cast from integer to pointer".

Because of this situation in some projects all the 11.x Rules have been completely ignored. In some other projects the Rules have been kept, accepted, and their violations have been confined (as much as possible) in isolated modules, isolated components. The advantages of the second approach are obvious: whenever deviations have to be adopted, it makes sense to confine such deviations to a portion, a subset of the system, so that the general

recommendations and guidelines are still valid and applicable to the other portions.

3 MISRA and CERT Standardization Efforts

Both MISRA and CERT have tried to harmonize, standardize the way in which deviations can be established.

MISRA

MISRA in April 2016 has published a guidance, MISRA Compliance:2016 ("Achieving compliance with MISRA Coding Guidelines" – reference document [3]).

MISRA guidelines are divided in:

1. directives: guidelines which are not defined with reference to the source code alone, but which also refer to, or impose requirements on processes and documentation;
2. rules: guidelines which impose requirements on the source code and the source code only.

Rules are then divided into:

1. decidable: rules that can always be assessed, verified by a (properly configured) analysis tool;
2. undecidable: rules that cannot be assessed, verified by an analysis tool in every situation.

Finally, guidelines are categorized into:

1. mandatory: guidelines for which violation is never permitted;
2. required: guidelines for which violations are permitted if justified by documented deviations
3. advisory: guidelines that can be violated without the need of a corresponding deviation.

All this framework gives a set of indications on how to establish deviations. Mandatory guidelines cannot have deviations. Among the required guidelines decidable rules are the less expensive to verify while undecidable rules and directives are more expensive. Advisory guidelines can be followed till it make sense, till it is reasonably practical to do so. It is important that all deviations are properly documented at the beginning of the projects and the guidance together with Appendix I of reference document [1] provide a detailed table of contents for the deviation record. On this specific point it must be noted that both the guidelines with their explanatory texts and examples as well as the deviations justifications provide a valuable training material that can be used to increase the "knowledge" and "awareness" of the less "experienced" team members.

The guidance provides also special provisions for "adopted source code", i.e. code deriving from other projects, for which further deviations may be needed in addition to the ones applied to the project code, and compiler standard libraries, for which in general MISRA compliance is not required.

CERT

The CERT compliance framework is similar but somehow simpler than the MISRA one (see reference document [2] chapter 1, and especially sections 1.8 "Conformance" and 1.13.4 "Risk Assessment").

CERT guidelines are divided into:

1. recommendations: guidelines that are likely to improve the quality of the system;
2. rules: guidelines whose violations are likely to introduce defects which may adversely affect the safety, reliability, or security of the system.

Each guideline contains a risk assessment based on its severity, likelihood and remediation cost; all these attributes are expressed as a number ranging from 1 to 3.

- severity: 1 – low, 2 – medium, 3 – high.
- likelihood: 1 – unlikely, 2 – probable, 3 – likely.
- remediation cost: 1 – high, 2 – medium, 3 – low.

The risk associated to a guideline is the product of these three attributes, which is called priority. Though the product ranges from 1 to 27, only the following 10 distinct values are present in the document: 1, 2, 3, 4, 6, 8, 9, 12, 18, and 27.

This scheme allows the definition of levels:

- L1: guidelines with priority from 12 to 27
- L2. guidelines with priority from 6 to 9
- L3: guidelines with priority from 1 to 4

Violations to L1 guidelines are high severity, likely and inexpensive to fix; violations to L2 guidelines are medium severity, probable and medium cost to fix; violations to L3 guidelines are low severity, unlikely and expensive to fix. A system is said to be L1 compliant if no L1 guideline is violated, L1-L2 complaint if no L1 and L2 guideline is violated and fully compliant if all guidelines are satisfied.

Though CERT Coding Standard realizes that strict adherence to all rules is unlikely and therefore deviations are needed, contrary to MISRA [1, Appendix I] [3], it does not provide detailed information on the deviation procedure nor on how to document deviations.

4 Analysis Tools

The web version of the CERT C Coding Standard (reference document [4]) for each guideline lists which tools can verify it (if any). The list is quite detailed providing the name of the tool, its version, the checker involved and a description with some additional information. Several years of Independent Software Verification and Validation activities have allowed the author to gather quite some experience on the various analysis tools. This experience can be summarised in the following heuristics.

There is no perfect tool. No tool is able to verify all MISRA (or all CERT) guidelines in all conditions. Analysis tools, in particular conditions, may generate both

"*false positives*" (i.e. the tool reports a flaw when one does not exist) and "*false negatives*" (i.e. the tool does not report anything when a flaw actually does exist). "*False positives*" can be simply filtered out, either during the analysis or when reporting the results. On the contrary there is no remedy against the "*false negatives*", apart from using different tools and hoping that what is missed by one tool is detected by another one.

Analysis tools can be divided into three broad categories:

1. "Shallow Analysers": based on patterns matching, e.g. "PC-Lint", "splint", "Understand", "Cppcheck", …
2. "Deep Analysers": based on techniques like bounded model checking, semantic analysis and abstract execution, e.g. "CBMC", "Frama-C", "Polyspace", …
3. "Compiler Based Analysers": based on the analyses performed by the compilers themselves, e.g. "Clang Static Analyzer", "Facebook Infer", …

"Shallow Analyser" are usually able to verify/assess the compliance of the majority of MISRA and CERT guidelines. In some cases, for some specific guidelines, "*Deep Analysers*" or manual intervention may be required.

The problem with "*Deep Analysers*" is that they require a lot of computing resources to perform their analyses and this might compromise their applicability to large codebases.

A recent, very interesting trend in static analysis is the adoption of "*Compiler Based Analysers*"; they offer several advantages, among which the most important are:

1. they are the "real thing", the very same tools used to build the software under analysis;
2. they are fast and can easily analyse large codebases;
3. they are easy to use, especially by developers and testers who are already accustomed to the compilation toolsets.

5 A Viable Approach to Manage Compliance

Compliance management can be seen as a process, divided in the following steps:

1. guidelines assessment and selection
2. system partitioning
3. tools configuration
4. (continuous) analysis execution
5. results exploitation

Guidelines Selection

Ideally in every project all guidelines should be adopted. Having to decide which deviations must be put in place it is very helpful to follow MISRA and CERT guidance in terms of priority, looking at which guidelines offer the "best value for money".

For MISRA:

1. decidable rules
2. undecidable rules
3. directives

It should be noted that in total there are 10 mandatory guidelines, i.e. guidelines that cannot be deviated; all of them are rules, 4 are decidable and 6 undecidable.

For CERT:

1. L1 guidelines
2. L2 guidelines
3. L3 guidelines

Deviations should be documented and MISRA offers a good guidance in terms of which information should be included in their justification.

System partitioning

Nowadays trends in software design and architecture, see for instance reference document [5], concentrate on simplifying and facilitating both the development and maintenance of software systems. This is why principles like the "dependency inversion" or design patterns like the "plugin architecture" are widely spread and adopted. On top of these criteria it would make sense to partition a system in modules, components based on their compliance to MISRA and/or CERT guidelines. For instance, a system could be partitioned into layers:

1. High-level, application level, layer – containing all modules where full compliance can be achieved
2. Low-level, base software, layer – containing all modules where deviations must be introduced (e.g. components interfacing the hardware, legacy libraries, compiler/language libraries, etc…)

Even in embedded, real-time, critical systems, quite likely, the majority of the components would belong to the high-level layer.

Tools Configuration

The configuration of tools is a rather expensive activity whose effort, based on the size and complexity of the project, may vary from few days to few weeks. Its purpose is to identify, define details like:

- selection of the analysis tools, to reduce as much as possible the "*false negatives*" while maintaining the costs inside a reasonable envelop;
- how to call the analysis tools, with which options, on which files;
- how to prepare the code base under analysis, so that the tools can process it (if needed);
- how to filter out "*false positives*" and when, if during the analysis itself or when presenting the results.

The configuration costs can be shared among projects with the same typology, e.g.: same quality/compliance requirements, same type of software, same type of platform, same compilation toolset, etc…

It must be noted that "Compiler Based Analysers" require very little or even no configuration at all (in fact the

compilation toolset has already been configured to build the software).

Results exploitation

It would be very limiting and economically inconvenient to consider the MISRA/CERT compliance analysis just as a tool, a sort of "*necessary evil*", required to obtain a given quality mark, an official certificate of the quality of a system.

MISRA and CERT guidelines are precious instruments to identify in the codebase "*hot spots*", "*critical areas*", portions of code that require special attention (during both the development and the maintenance).

Analysis tools, quite often, produce their results as text files or spreadsheets. It is much more useful to display the violations from within the code itself.

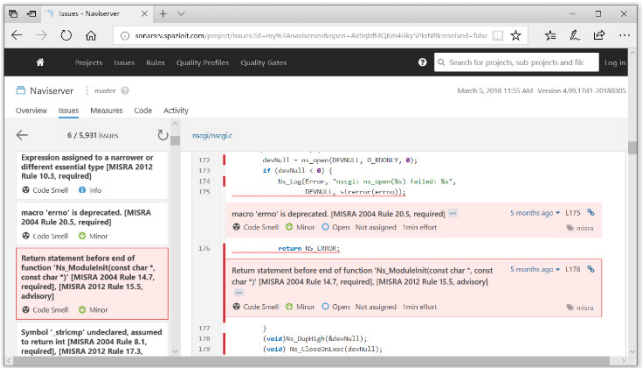


Figure 1- MISRA Violations in SonarQube

Figure 1 shows how valuable is to display the violations from within the code (see reference document [7]). For instance, looking at the code it is much easier to decide if a violation is valid or if it is a "*false positive*". On top of that, the information contained in the guidelines together with portions of code violating them become a powerful and effective "educational tool", a way of passing knowledge to less experienced developers or testers. There are quality platforms, e.g. SonarQube – see reference document [6], that are able to aggregate the results of various analysis tools and present them from within the code. On top of that they are also able to "follow" the historical evolution of the project and show, for instance, how violations have been addressed and solved.

6 Conclusions

The paper has shown the potential drawbacks of excessive customization during the tailoring of the MISRA (or CERT) guidelines to be applied to a given project. Examples have been provided in usually critical areas like data handling, data representation, defensive programming and (pointer) type conversions.

Secondly the paper has presented the efforts of both MISRA and CERT in the attempt of standardizing the process of customizing the guidelines. Though MISRA and CERT have produced two similar conceptual frameworks, the CERT one seems simpler.

Thirdly the paper has presented the current "*status of the art*" of the analysis tools.

Finally, the paper has presented the approach the author is using and recommending to manage compliance. The approach is based on concentrating more on actual feasibility and economic considerations rather than following a strict, a priori, adherence to regulations and standards.

Proper emphasis has been put on the educational value of MISRA and CERT guidelines.

References

[1] The Motor Industry Software Reliability Association (2012), *MISRA C:2012, "Guidelines for The Use of The C Language in Vehicle Based Software*", March 2013, ISBN 978-1-906400-11-8.

[2] Software Engineering Institute, Carnegie Mellon University (2016), *SEI CERT C Coding Standard - Rules for Developing Safe, Reliable, and Secure Systems*.

[3] The Motor Industry Software Reliability Association (2016), *MISRA COMPLIANCE:2016, "Achieving compliance with MISRA Coding Guidelines"*, ISBN 978-1-906400-13-2.

[4] SEI CERT C, *SEI CERT C Coding Standard* – Web Version: https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard

[5] R. C. Martin (2017), *Clean Architecture: A Craftsman's Guide to Software Structure and Design,* Prentice Hall, Pearson Education , ISBN 978-0-13-449416-6.

[6] SonarQube, http://www.sonarqube.org.

[7] http://sonarsrv.spazioit.com/projects.

Using Ada in Non-Ada Systems

A Marriott, U Maurer

White Elephant GmbH, Beckengässchen 1, 8200 Schaffhausen, Switzerland; email: software@white-elephant.ch

Abstract

This article is based on the industrial presentation "Using Ada in non-Ada systems" which was given at the 2018 Ada-Europe conference in Lisbon.

The presentation was an experience report on our use of Ada packages within existing non-Ada embedded microprocessor based systems.

Keywords: GCC, Modula-2, C, ZFA

1 History

In the late eighties, when we first started replacing discrete electronics with embedded microprocessors, there were very few Ada compilers available, especially for microprocessors, and those that did exist were slow, required vast resources and were very expensive.

As a consequence, and because we wanted to use a highly typed language rather than use the ubiquitous C, we decided to implement our systems in ISO 10514 Modula-2. Modula-2 is a programming language invented by Niklaus Wirth that has many features in common with Ada. For example, it's verbose non-ambiguous syntax and the separation of specification and implementation into separately compiled units.

Originally we used cross compilers, compiling Modula-2 source directly into the target machine code. However, over time it became increasingly difficult to find Modula-2 compilers for the new microprocessors that were coming onto the market.

For this reason we switched to using a Modula-2 translator that translates Modula-2 into C. This machine generated C is then compiled into the target machine code. We rarely look at this machine-generated C code – preferring to treat it as some form of intermediary "binary".

None the less our Modula-2 is translated into C and it is this C that is compiled and linked to form our embedded hard real-time systems. Later in this article, when I present how and why we have started using Ada in our systems, it should be noted that we are effectively talking about using Ada in a predominantly C environment. The fact that we ourselves don't actually program in C, or even know how to program in C, is a luxury we have been afforded but that shouldn't distract from the usefulness or relevancy of this article.

We have a large amount of well-established code that executes on a multitude of platforms and that uses our own proprietary multitasking run-time. Management is unlikely to sanction the conversion of this code base into Ada - if only because the risk of introducing errors would far out way any perceived benefit of coding exclusively in Ada.

However this is not to say that new features or features that have to be substantially modified couldn't be written in Ada, provided that an affordable Ada compiler exists for the target microprocessor architecture and if the code can be integrated into the existing system.

Until recently, we have been using the Wind River Diab tool chain to build our executables (in ELF format with DWARF debugging information) for Motorola M68332 and Coldfire microprocessors. We have no intention of touching these systems. However our most recent hardware is ARM based and we have also switched C compiler and now use the Gnu Compiler Collection (GCC).

In fact we use GCC version 6.3.1 to compile our C code for ARM which is the same version of the GCC that AdaCore releases under GPL 2017 for compiling Ada for ARM. So the challenge has been to write code in Ada and then use GNAT to compile it and link it together with our existing C code.

An important caveat is that we are not talking about using full Ada. A lot of the power of Ada comes from language features that depend on its runtime. However we already have a runtime. Rather than modify the Ada runtime to use our runtime or modify our runtime to use Ada's, we decided, at least for now, that the simplest course of action is to restrict ourselves to a subset of Ada that doesn't require a runtime.

This is what is known as the Zero Footprint profile for Ada.

Exactly what Zero Footprint Ada means for any particular system depends on which pragma restrictions are declared in the file System.ads

For example a typical ZFA could be defined as:

```
pragma Restrictions (No_Exception_Propagation);
pragma Restrictions (No_Implicit_Dynamic_Code);
pragma Restrictions (No_Finalization);
pragma Restrictions (No_Tasking);
pragma Restrictions (No_Delay);
pragma Discard_Names;
```

These restrictions mean we lose a lot of nice features of Ada. Features such as:

- Tasks

- Protected objects
- Controlled types
- The delay statement
- Dynamic storage allocation using new
- Exception propagation

In addition to the above we also voluntarily imposed additional restrictions to reduce Ada down to the level we wanted to support.

For example our target microprocessor has no fixed point hardware therefore any code that uses floating point will be exceedingly slow. To prevent the accidental use of floating point we added

```
pragma Restrictions (No_Floating_Point);
```

into System.ads.

Another restriction, at least initially, is to forego Ada functions that return unconstrained types. This is because variable length results are returned to the caller using what GNAT terms the secondary stack. However the microprocessors we are currently using have very little RAM, therefore we can ill afford the luxury of having a second stack for each and every task.

Consequently we added

```
pragma Restrictions (No_Secondary_Stack);
```

into System.ads.

The main consequence of this decision is that we can't write Ada functions that return strings nor can we use attributes such as 'image or 'img.

We also initially decided to restrict ourselves to writing pre-elaborated packages. By declaring all our packages "*with preelaborate*" or "*with pure*" and including

```
pragma Restrictions (No_Elaboration_Code);
```

in system.ads we forego elaboration.

Without elaboration:

1. Global variables can only be initialised to values evaluated at compile time
2. Packages may not have a body, i.e. code between the *begin* and *end* of the package implementation.
3. Pre-elaborated packages may only call packages that are themselves pre-elaborated or pure.

However, even with all these restrictions we believe that there is still enough left of Ada to make integration attractive. We have always considered Modula-2 to be a poor man's Ada. However, in our opinion, even a severely cut back Ada is better than programming in Modula-2 and we can only imagine how much of an improvement it must be over writing in C.

Ada is obviously syntactically superior to C and even though they share the same roots, Ada offers many advantages over Modula-2

For example:

- Named parameters
- Named fields in constructors
- Private types, functions and procedures.
- Bit level specification in representation clauses.

Representation clauses are extremely useful when interfacing to hardware and third party protocols. An enumeration that is not represented as a complete byte is accessed in most computer programming languages by a combination of bit masks and shifting – a typically error prone endeavour that is handled automatically by Ada.

2 Getting Started

The simplest form of integration is when a program written in Modula-2 calls a parameter-less procedure written in Ada.

To make procedures accessible from other modules, Modula-2 mangles the procedure names by prefixing them with the name of the module together with a separating underscore. Thus procedure Y defined in the definition of module X would be called X_Y.

In Ada a similar thing happens. The global procedure name is composed of the package name followed by a double underscore followed by the name of the procedure, and the whole name rendered to lowercase. Thus procedure Y defined in the specification of package X would be called x__y

Therefore to access an Ada procedure from C you first need to declare the Ada procedure as an external procedure

```
extern void adaunit__adaprocedure (void);
```

and then call it using its full mangled name

```
adaunit__adaprocedure();
```

To do this in Modula-2 we have to import the package and then call the procedure in the same way we would for a procedure written in Modula-2

```
IMPORT AdaUnit;
AdaUnit.AdaProcedure;
```

As this is written in the same way that a Modula-2 procedure would be called we need to inform the translator that the procedure being called is an Ada procedure rather than one written in Modula-2.

This is achieved by creating a foreign definition module that tells the Modula-2 translator which language the procedures within the module are written in.

```
DEFINITION MODULE ["Ada"] AdaUnit;
  PROCEDURE AdaProcedure;
END AdaUnit.
```

The above informs the translator that the procedure AdaProcedure in the module AdaUnit is written in Ada and therefore will have its global name mangled to adaunit__adaprocedure.

We then have to write and compile the procedure in Ada

```
package body AdaUnit is
  procedure AdaProcedure is
  begin
    null;
  end AdaProcedure;
end AdaUnit;
```

and then make a specification so that it is exported.

```
package AdaUnit is
  procedure AdaProcedure;
end AdaUnit;
```

The Ada package has to be compiled using GNAT and the Modula-2 translated into C which is then compiled by the GCC. The resultant objects then have to be linked together to produce an executable.

In order that certain Ada features are made available, the compiler requires a number of ads files to be found somewhere in the source search path.

A package implementation is not required because the implementation is intrinsic (built into) the compiler.

For example, using Ada.Unchecked_Conversion requires that the file a-unccon.ads to be found in the source file search path.

Unfortunately GNAT has the very strange restriction that these specification files MUST have the "crunched" file names listed below. It does not support their being named according to the more usual convention derived from the full name of the package they contain. This is presumably some sort of historical left over, which is a pity, because these names are both ugly and unreadable.

- a-except.ads (ada.exceptions)
- a-unccon.ads (ada.unchecked_conversions)
- interfac.ads (interfaces)
- s-maccod.ads(system.machine_code)
- s-stoele.ads (system.storage_elements)
- s-unstyp.ads (system.unsigned_types)

3 Debugging

If the executable had been written entirely in Ada and ran on a machine sitting on a nearby desktop, we could have used something like GPS for debugging. However this is not the case. Our code is a mixture of Modula-2, C and now Ada. Moreover the machines are physically remote and not easily accessible.

So when something goes wrong our machines generate a memory dump and then, sometime later, we use a static dump analyser. The analyser uses the debug information that is stored in the executable file by the compiler and linker. It expects this information to be written according to the DWARF standard.

Fortunately for us GNAT is based on the GCC, which accepts the switch -gdwarf-3. This switch causes GNAT to supply debugging information according to version 3 of the DWARF standard and to place this information into the ELF executable.

Our challenge has been to enhance our analyser to better support bit fields and sub-ranges – something it never had to deal with when the executables were built purely from C.

Another debugging problem concerns the GCC's link time optimisation feature. This feature is enabled using the -lto switch and is required for the in-lining explained later in section 7.

Entries within the DWARF debugging information are contained within compilation units. These compilation units are Ada packages or Modula Modules. The full global name of an entity can normally be derived by prefixing the name of the compilation unit with the name of the entity. Unfortunately a side effect of using the lto feature is that the compilation units are all renamed <artificial>!

To solve this problem all our Modula-2 & C variables and procedures have to be name mangled in order that we can differentiate and know in which module the entity was defined. We have to do this even if the entity is not exported, i.e. is only used locally and therefore, from the linker's perspective, does not have to have a globally unique name.

Fortunately for us, Ada also mangles all its names - even if they are not exported. So this is not a problem and we can therefore use Link Time Optimization.

4 Functions

To make our example a little more useful we can replace the parameter-less procedure with a function that increments a global variable and returns its new value.

```
package AdaUnit is
  function AdaFunction return Integer;
end AdaUnit;

package body AdaUnit is

  TheGlobal : Integer;

  function AdaFunction return Integer is
  begin
    TheGlobal := TheGlobal + 1;
    return TheGlobal;
  end AdaFunction;

end AdaUnit;
```

However when we try to link a program that calls AdaFunction the linker complains that it is missing a last chance handler for the function.

This is because the function will raise an exception when TheGlobal reaches Integer'last. If this situation is not explicitly handled, the Ada compiler will insert a call to the last chance handler **__gnat_last_chance_handler**.

Of course, one could define a last chance handler and then link this into the final program. However we chose not to. Instead we chose to always explicitly handle Ada implicit exceptions.

For example by rewriting the code so that the error situation cannot arise:

```
function AdaFunction return Integer is
begin
  if TheGlobal < Integer'last then
    TheGlobal := TheGlobal + 1;
    return TheGlobal;
  else
    return Integer'last;
  end if;
end AdaFunction;
```

or by catching the exception

```
function AdaFunction return Integer is
begin
  TheGlobal := TheGlobal + 1;
  return TheGlobal;
exception
when Constraint_Error =>
  return Integer'last;
end AdaFunction;
```

By adding the switch -gnatw.x the Ada compiler will generate a warning if an implicit or explicit exception is not covered by a local handler.

Unfortunately Ada doesn't always get it right and we often have false positives – occasions when Ada warns that an exception may be raised when in fact this is not possible.

In the following example Ada complains that *Constaint_Error* might be raised when calling The_Handler.all even though the explicit check for a null pointer precludes this.

```
type Handler is access procedure;
The_Handler : Handler;

procedure Test is
begin
  if The_Handler /= null then
    The_Handler.all;
  end if;
end Test;
```

Interestingly, if we switch off warnings for the duration of the code in question, the program still links. This therefore shows that the compiler was, in fact, smart enough to realise that the exception could not be raised.

Rather than disable and then re-enable warnings we prefer to use the pragma *Suppress* to remove the specific check.

Suppressing checks can be selective. Typically we place the code that is causing the problem within a declaration block and add the appropriate **pragma suppress** between the **declare** and **begin** statements.

For example:

```
declare
  pragma suppress (Access_Checks);
begin
```

We consider this less error prone than messing around with warnings but we also hope that, as the compiler is improved, it might one day warn us that these pragmas are no longer necessary.

5 Initialising Globals

Global variables can be initialised using the standard Ada syntax. In our example the global variable can be initialised to forty two by declaring it as:

```
TheGlobal : Integer := 42;
```

Initialising variables in this manner does not work without a runtime to initialise the variable. Zero footprint Ada does not have a runtime and so if used purely by itself it would require an alternative mechanism to initialise global variables. However we are using Ada within an existing system, the runtime of which will initialise **ALL** global variables, irrespective of the compiler used, provided that all the compilers adhere to a few conventions.

Fortunately for us, GNAT adheres to these conventions and so its global variables are initialised in the same way that global variables written in ether Modula-2 or C are.

How does this work?

Quite simply, global variables are placed in a section called .bss if they are initialised to zero or in a section called .data if they are initialised to anything else.

The following GCC linker script groups all the .bss variables along will all uninitialized variables (COMMON) together and sets two linker symbols to the start and end addresses of the area of memory they have been allocated. The same script groups all initialised data together, assigns another pair of linker variables to their start and end addresses and instructs the linker to place their initialization values into ROM.

```
.mdata :
{
  __Data_Start = ABSOLUTE(.);
  *(.data*)
  __Data_End = ABSOLUTE(.);
} > Ram AT > Rom
.bss :
{
  __Bss_Start = ABSOLUTE(.);
  *(.bss)
  *(COMMON)
  __Bss_End = ABSOLUTE(.);
} > Ram
__Data_Rom  = LOADADDR(.mdata);
__Bss_Size  = __Bss_End - __Bss_Start;
__Data_Size = __Data_End - __Data_Start;
```

The runtime has access to the linker defined global symbols __Data_Start, __Data_Rom, __Data_Size and Bss_Size. Using these symbols the runtime can initialise memory thus:

```
MOVE (DataRom(),DataStart(),DataSize());
FILL (BssStart(), 0, BssSize());
```

The first instruction copies the initial values of initialised variables into the space occupied by the variables. The second instruction initialises to zero all remaining variables.

6 Ada calling Modula-2

The examples so far have shown how code written in Modula-2 or C can call routines written in Ada however these Ada routines would be severely restricted if they were not able to communicate with portions of the application written in languages other than Ada.

To be useful our Ada routines need to be able to call routines written in Modula-2. This is achieved by declaring the function as an import using the C calling convention and by specifying its external name. In the case of Modula-2 the external name is the name of the module followed by an underscore followed by the name of the procedure.

For example, the Modula-2 module ModulaUnit could be defined as:

```
DEFINITION MODULE ModulaUnit;
  PROCEDURE ModulaFunction () : INTEGER;
END ModulaUnit.
```

And implemented as:

```
IMPLEMENTATION MODULE ModulaUnit;
  PROCEDURE ModulaFunction () : INTEGER;
  BEGIN
    RETURN 42;
  END ModulaFunction;
END ModulaUnit.
```

And then the function called from Ada as:

```
 procedure Example is
  function ModulaFunction  return Integer
  with Import, Convention => C,
      External_Name => "ModulaUnit_ModulaFunction";
begin
  TheGlobal := ModulaFunction;
end Example;
```

This is all very well provided that the types are base types that both Modula-2 and Ada agree are the same. Problems arise when the types are represented differently. In these cases a wrapper is required.

For example, in Modula-2 (and C) a Boolean is defined to be eight bits wide. In Ada the Boolean type is defined to be only one bit wide however the compiler is generally free to allocate more than this for objects of type Boolean – how much isn't defined by the language. Therefore when Ada calls a Modula-2 function that returns a Boolean we need to do this via a wrapper function.

For example:

If our Ada code wants to call the Modula-2 function Hardware_Is_Available from module Ip we first define the specification in Ip.ads as

```
function Hardware_Is_Available return Boolean;
```

and then define the wrapper in Ip.adb as

```
type Modula_BOOLEAN is new Standard.Boolean
with Size => 8;

 function Hardware_Is_Available return Boolean is

  function Ip_Hardware_Is_Available return
Modula_BOOLEAN
   with Inline, Import, Convention => C,
   External_Name => "Ip_HardwareIsAvailable";

begin
  return Boolean(Ip_Hardware_Is_Available);
end Hardware_Is_Available;
```

The astute will notice that the Modula-2 function that the wrapper calls is declared as Inline. Which brings us nicely onto the subject of in-lining.

7 In-lining

The relatively weak microprocessors we use cannot afford the overhead of superfluous calls. Certain time critical portions of our code must be in-lined for efficiency reasons. The GCC is very good at in-lining provided the –flto option is specified on the command line when compiling C and – Winline when linking. In addition, in order that Ada in-lines in the same way, we need to specify the switch –gnatn2 when compiling our Ada source code.

The result is very impressive; in-lining is possible between units as well as between languages. The example of the Boolean wrapper produces absolutely no extra code - the wrapper keeps Ada happy without any additional overhead.

8 Enumerations

In C, the amount of storage allocated to enumeration types defaults to the word size of the target machine. In our case this is 32 bits. However reserving 32 bits for every enumeration is extremely wasteful for microprocessors that are memory challenged, so we compile using the switch --short-enums which directs the GCC to use the least number of bytes possible to store any given enumeration. This turns out to have been a very fortunate decision because enumerations in Ada use the same storage strategy, and so by using this switch we make enumerations compatible between Ada and C.

9 Strings

Strings are another occasion when wrapper functions are required.

In the following example, the Modula-2 procedure takes a string as its parameter. Strings in Modula-2 are unconstrained arrays of character and so the procedure DefineComputerNameAs is defined as follows.

```
PROCEDURE DefineComputerNameAs (TheName :
ARRAY OF CHAR);
```

This translates into C as

```
extern void Nbns_DefineComputerNameAs(const
char [], unsigned long);
```

Where the unconstrained array of characters has been translated into two parameters, the first being the start address of the array and the second the number of elements in the array.

To call this from Ada we need to provide a wrapper. For example:

```
procedure Define_Computer_Name_As
(The_Name : String) is

  procedure Nbns_Define_Computer_Name_As
      (Name_Address : ADDRESS;
       Name_Size    : CARD32)
   with Inline, Import, Convention => C,
   External_Name =>
       "Nbns_DefineComputerNameAs";

begin
  Nbns_Define_Computer_Name_As
       (The_Name'address, The_Name'length);
end Define_Computer_Name_As;
```

10 Exception Handling

The above example is not quite right. We shouldn't pass the address of the String but the address of the first character of the string. However if we code this then we need to check that the string has at least one character and decide what to do if it doesn't.

Ideally we would raise an exception. Unfortunately zero foot print Ada precludes the propagation of exceptions, however this does not mean that we cannot define exceptions provided we catch them locally or use them for other purposes.

Note however that the –gnatwh compiler switch to detect declaration hiding does not detect the hiding of standard exceptions. The Standard exceptions

- Constraint_Error
- Program_Error
- Storage_Error
- Tasking_Error

are implicitly raised by compiler checks. Therefore, to avoid confusion, it is highly recommended not to declare exceptions with these names.

Our existing Modula-2 system has an exception concept. Our Modula-2 exceptions can be raised but not caught and are always fatal. They stop the machine and produce a memory dump for later analysis.

In the previous example, if we correct the code to pass the address of the first character, Ada will complain that this might raise an exception. So we need to include additional code that explicitly handles that situation.

```
Empty_Name : exception;
begin
  Nbns_Define_Computer_Name_As
    (The_Name(The_Name'first)'address,
     The_Name'length);
exception
when Constraint_Error =>
  HALT (Empty_Name'identity);
end Define_Computer_Name_As;
```

The procedure HALT saves the exception identity and stops the system. Our analyser can retrieve this identity, which is nothing more than the address of the exception, and convert it into its symbolic name.

11 Elaboration

Unlike C, Modula-2 has the concept of elaboration. It is not as powerful as Ada – global variables cannot be elaborated – but modules can have body code that is executed at start-up before any of the exported procedures can be called.

However our Ada packages only link to specific named routines and there is no concept of using “with” to import units written in anything other than Ada. Consequently there is no Ada syntax or mechanism whereby Ada can be instructed to elaborate a specific foreign unit.

And even if there were, we decided that all our Ada packages are either pure or pre-elaborate.

However this decision turns out to be too much of a restriction. Too much of our existing code requires the Modula-2 module bodies to be executed prior to their exported routines being made available. Not being able to elaborate our Ada packages was also inconvenient.

Therefore we changed our strategy and decided to implement and support elaboration.

The first problem was establishing the elaboration order. If unit A calls unit B then unit B must be elaborated before unit A is elaborated. If unit B calls other units then these must be elaborated before unit B and so on. If any unit calls a unit that has to be elaborated before itself, then this is a cyclic dependency and must be regarded as an error.

Because Modula-2 has the concept of elaboration our IDE already had a mechanism for determining the elaboration

order of Modula-2 modules. So all we had to do was extend this mechanism to include units written in Ada.

The IDE has to parse the Modula-2 source files and process the *IMPORT* statements and parse the Ada source files and process the *with* statements. Whilst the Modula-2 *IMPORT*s indicate a unit dependency, irrespective of language, the Ada *with* is restricted to indicating the package's dependency only on other Ada packages and does not include any dependency on units written in other languages.

We were therefore obliged to enhance our IDE to recognise a new pragma.

By default GNAT issues a warning when it encounters an unrecognised pragma. This warning can be switched off using the -gnatwG switch. Using this switch is potentially dangerous and contrary to the Ada Reference Manual specification that a warning be generated whenever an unrecognised pragma is encountered. Therefore we had to enhance our IDE to verify pragma names and issue our own error message if it detected any unrecognised pragmas, i.e. unrecognised by GNAT and not an extension implemented by our own IDE.

So solve the elaboration problem we recognised the new pragma *Modula_Import*. The pragma takes as its parameter the name of a Modula-2 module.

For example: **pragma** Modula_Import (ModulaUnit);

Note: It isn't quite that simple because Modula-2 module names can have names that aren't Ada identifiers. However how we handled this anomaly is a detail beyond the scope of this short article.

By processing the *IMPORT*s, *with*s and *pragma Modula_Import* statements, our IDE can build the dependency tree. Provided that there are no cyclic dependencies it can then generate a table of procedures that must be called at start-up before the main program is entered.

For example:

```
extern void
__attribute__((weak)) ModulaUnit_BEGIN(void);

extern void
__attribute__((weak)) adaunit___elabb(void);

typedef void (*Unit_List[ 1])(void);

static const Unit_List Unit_Body_the_list = {
  ModulaUnit_BEGIN,
  adaunit___elabb};
```

The name of the elaboration routine for a Modula-2 module is the name of the module followed by _BEGIN whilst the name of the elaboration routine for an Ada package is the name of the package followed by elabb if the implementation is being elaborated or elabs if the specification requires elaboration.

There is no easy way to detect whether or not an Ada package requires elaboration, so our IDE needs to assume that all Ada packages might be elaborated unless directed otherwise. This is implemented by the IDE building a table of weak links to possible elaboration routines.

The use of weak external links means that if the unit did not require elaboration and consequently the expected elaboration routine was not generated, the linker would not complain but instead leave the default null pointer in the table. These null entries obviously have to be skipped when processing the table.

```
int the_index;
 for (the_index = 0; the_index < 7ul; the_index++) {
   if (Unit_Body_the_list[the_index]!=0)
    Unit_Body_the_list[the_index]();
 };
```

To avoid possible cyclic dependencies it is sometimes necessary that Ada (and the IDE) be told that the package does not require elaboration. This is achieved using the aspect "*with preelaboration*" or "*with pure*".

12 Interrupt routines

Our applications require that we write interrupt routines. On ARM microprocessors, interrupt routines are nothing other than parameter-less procedures whose addresses are placed into the vector table.

Using standard Ada the address of the procedure is placed into the vector table using the *pragma Attach_Handler*. Unfortunately when we use this, GNAT complains that the argument of *pragma Attach_Handler* must be a protected procedure.

However protected types and procedures require a run-time and are therefore not allowed in the Zero Footprint Profile.

In any case, even if *Attach_Handler* was allowed, it is unlikely that it would of any use because we need a mechanism that allows a vector table to be generated that has entries of procedures written in a mixture of languages – not just Ada.

For this reason, our IDE builds the vector table. The IDE is instructed to add a procedure into the vector table by special constructs within the source.

In Modula-2 this is achieved by using the direct language specification "Vector"

For example:

```
PROCEDURE ["Export", "Vector=36"] InterruptHandler;
```

In order that a similar thing could be achieved from sources written in Ada, we further enhanced our IDE to recognise an additional **pragma** Use_Vector

For example:

```
procedure Interrupt_Handler with Export;
pragma Use_Vector (36);
```

In the above example, the pragma Use_Vector instructs the IDE to place the address of the exported, parameter-less procedure Interrupt_Handler into interrupt vector position 36.

13 The use of assembler

We haven't had cause to write much assembler but there will always be occasions when this is necessary. Fortunately this is possible. The GNAT package System.Machine_Code provides the procedure Asm which behaves in a similar and recognisable manner to that of the standard GCC embedded assembler but with the rather tiresome restriction that parameters can only be reference by position rather than by name.

In the following example, written in C, the procedure DisableInterrupts places the constant 1 into a register of its choice that we symbolically call Mask which the MSR instruction then loads into the Primask register.

```
__attribute__ ((always_inline)) inline
static void DisableInterrupts(void)
{
  asm volatile (
      "MSR primask, %[Mask];"
      :
      :[Mask] "r" (1)
      :"memory");
}
```

Unfortunately GNAT does not support the use of named parameters and therefore in Ada the register used to house the constant has to be referred to by its position in the list of inputs (starting at zero)!

```
with System.Machine_Code; use
System.Machine_Code;

procedure Disable_Interrupts with Inline is
begin
 Asm ("msr primask, %0;",
     Inputs   => Integer'asm_input ("r", 1),
     Clobber  => "memory",
     Volatile => True);
end Disable_Interrupts;
```

Referring to parameters by their numeric position rather than by name seems like a step back into the stone-age.

14 Results

In the guise of conducting a feasibility study, we did exactly what we originally stated we wouldn't do. Rather than wait until an opportunity arose that would benefit from being written in Ada we decided to convert two ARM based applications that already existed and had already been written in Modula-2.

We didn't convert the whole application; we left the run-time and a lot of low level interfaces written in Modula-2 but we did convert all the application specific modules into Ada packages.

These included interrupt routines, interfaces to hardware and, of course, interfaces to our proprietary multitasking runtime.

So the port wasn't exactly trivial but on the other hand because of the similarities between Modula-2 and Ada it wasn't that difficult either.

We are pleased to report that the conversions were very successful and we now have two of our ARM specific applications written in Ada.

This is not to say that the conversion didn't have any problems. Unfortunately we did introduce a few errors as part of the conversion process. These occurred when the conversion was more complex than a simple syntax change

We identified four areas where conversion errors were likely to occur:

1. Ada has no syntax to increment or decrement a variable so it is impossible to implement the Modula-2 procedures INC and DEC without resorting to generics.
2. Modifying the code to replace INC and DEC statements with X:=X+1 and X:=X-1 respectively presented an opportunity to accidently decrement when we should have incremented and vice versa.
3. The Ada attribute 'size returns the size of an object in bits whereas the equivalent Modula-2 SIZE procedure returns the size in bytes. Therefore one must remember to divide 'size by the number of bits in a byte.
4. Expressions in Modula-2 are evaluated left to right and so there is no need for the Ada constructs *and then* and *or else*. Care is therefore required when converting Modula-2 Boolean expressions.
5. In Modula-2 *in* parameters can be modified – thereby saving a local variable. In Ada this is not allowed and so a local variable must be explicitly created, initialised and then used instead of the original *in* parameter. This complicated code modification is another opportunity to introduce subtle conversion errors.

15 Conclusion

This article is an experience report. It does not present anything particularly clever or original. Far from it. Our goal in writing this article was to illustrate how easy it is to integrate Ada into an existing non-Ada system and thereby perhaps animate others in a similar situation to use Ada where previously it would not have been considered.

Alire: a Library Repository Manager for the Open Source Ada Ecosystem

Alejandro R. Mosteo
Instituto de Investigación en Ingeniería de Aragón, Mariano Esquillor s/n, 50018, Zaragoza, Spain
Centro Universitario de la Defensa de Zaragoza, 50090, Zaragoza, Spain; email: amosteo@unizar.es

Abstract

Open source movements are main players in today's software landscape. Communities spring around programming languages, providing compilers, tooling and, chiefly, libraries built with these languages. Once a community reaches a certain critical mass, management of available libraries becomes a point of contention. Operating system providers and distributions often support but the most significant or mature libraries so, usually, language communities develop their own cross-platform software management tools. Examples abound with languages such as Python, OCaml, Rust, Haskell and others.

The Ada community has been an exception to date, perhaps due to its smaller open source community. This work presents a working prototype tailored to the Ada compiler available to open source enthusiasts, GNAT. This tool is designed from two main principles: zero-cost infrastructure and a pure Ada work environment. Initially available for Linux-based systems, it relies on the semantic versioning paradigm for dependency resolution and uses Ada specification files to describe project releases and dependencies.

Keywords: Library Management, Dependency resolution, Open Source, Ada 2012.

1 Introduction

"If I have seen further it is by standing on ye sholders of Giants" wrote Sir Isaac Newton in a letter to Robert Hooke [1]. Believers in the virtues of open source licenses may recognize the sentiment; in nowadays rapidly evolving technological landscape, reuse of code is critical to adapt to new technologies, avoid past errors, stay on top of vulnerabilities, and foster collaboration. In the communities built around programming languages this can be seen in the publishing of free software under more or less permissive licenses [2]. Open source programmers want their code to be run and built upon.

However, the availability of code and simplicity of distribution, compared to pre-Internet generalization, has brought with itself its own problems, such as a difficulty to be aware of available libraries, obsolescence of code that becomes unmaintained (a form of *bit rot* [3]) and incompatibilities between versions of a same library, or among different libraries being used simultaneously.

To address those problems, one of the most notable efforts in the open source world are the different Linux distributions. Either based on distribution of source code, like Gentoo [4], or of binaries, like Debian [5], these communities have since long dealt with the problem of packaging consistent systems for different architectures. The difficulty of such a task is captured in the *dependency* or *DLL hell* expressions [6], and one of the most dreaded experiences is ending in a *broken* configuration during an upgrade.

Programmers, however, do not all use the same distribution, nor even the same operating system, since today they can resort to about half a dozen generalist operating systems. Given the polarizing nature of programming languages [7] it is then unsurprising that many languages have seen efforts aimed at providing an easy way of distributing libraries for those languages, as we shall discuss in Section 2. In some cases, like Rust [8], the tool for the distribution of libraries is an integral effort of the team developing the language.

The Ada language, perhaps because of its ties to closed development and today's considered niche place in the language landscape [9], has not seen such a tool appear (to the best of our knowledge), despite the notable amount of open source libraries available [10]. This work presents a tool that could be a first step in this direction, with the main contribution being the tool itself. The tool tries to appeal to the Ada programmer by using native Ada code to describe releases and its dependencies, thus avoiding the need to learn new formats. To use this information, the tool uses self-compilation to incorporate the required data into its catalog of libraries. A contributed byproduct is the semantic versioning library[1] that is used to describe dependencies among releases.

The project started as an informal discussion[2] under the name of Alire (from Ada Library Repository), and this work reflects the view of the author on how a tool that addressed the low-hanging problems of the open source Ada community could be brought to life. The tool itself is termed `alr`,[3] in the vein

[1] `https://github.com/alire-project/semantic_versioning`

[2] `https://github.com/mosteo/alire/issues`

[3] A monospace font is used throughout the paper to denote actual executable commands or logical entities such as files.

of other venerable command-line tools such as `git`, `svn`, etc., and to distinguish it from the general project.

The paper is structured as follows: Section 2 examines the situation in other languages and points the referents taken for this tool. Section 3 introduces the design of `alr` and some use cases. Next, Section 4 presents details about the implementation mechanisms underpinning the design. A brief discussion follows on the open questions this design leaves and, lastly, concluding remarks and future directions close the paper in Section 5.

2 Related Work

The problem of library distribution has been tackled in two main ways, namely distribution of binaries and of source code. The former has the advantage of speed for the user, because it saves the step of compilation. The latter allows the complete tailoring of the building process to one's environment, and reduces the work load and hardware requirements on maintainers. Furthermore, for purely interpreted languages the distribution of sources is unavoidable.

Once libraries are obtained, we see yet two possibilities: installation of packages system-wide, as if they where integral parts of the platform, or local installation in a confined or user sandbox (that sometimes can be the default user environment). In Python's `pip` [11], e. g., libraries are installed globally if run as superuser. If run as a regular user, they will be installed in the user's environment. These two options present to the user a default environment that can become broken [6] when dependencies are improperly managed, and for that reason it is recommended [11] to use a sandbox or virtual environment for each development context (Fig. 1). Some packagers, like OPAM [12] or Nix [13], avoid that duplication by using a common store where individual releases are isolated (i.e., there is not a "current" version of any library).

Mainstream languages such as Java, C, and C++ also have a variety of tools at their disposal, the problem being in this case the lack of a standardized unique (or prevalent) go-to tool. Since these languages do not natively consider the build consistency problem as Ada does, their tools may also include complex building aspects, like Gradle/Maven do for Java [14]. A main player for the C/C++ world, Conan [15], is instead a build-system agnostic package manager that however relies on YAML configuration files and Python scripts, increasing the technical burden.

When one inspects the many solutions out there, like Rust's `cargo`, Python's `pip` and `easy_install`, OCaml's `opam`, D's `dub`, Haskell's stackage and `cabal-install`, to name more examples, a few common traits arise. The backend is usually some kind of database that in its simplest form is merely a set of files under version control in a public repository or in dedicated servers. Submission of new libraries becomes then the merging of a pull request into the stable branch of the catalog. Fetching of a library involves the download of a file bundle or checkout of a particular commit.

The other salient aspect these tools address is dependency resolution. When building a project with a complex set of

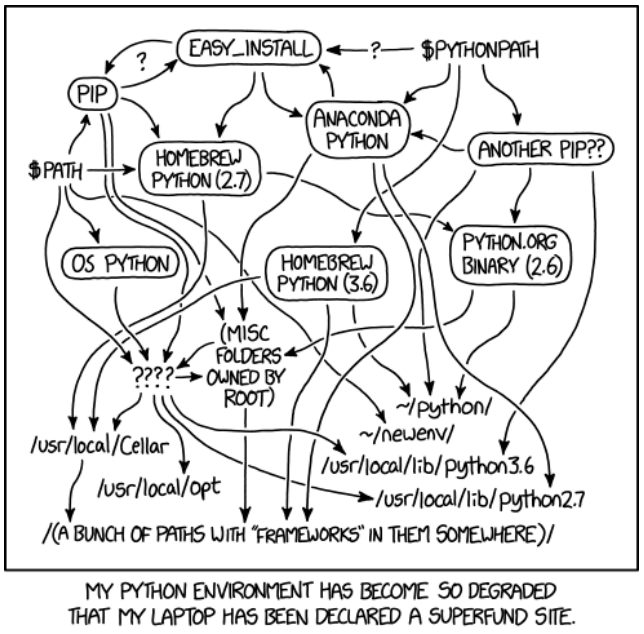


Figure 1: Library management problems have reached the level of Internet running joke (`https://xkcd.com/1987/`)

dependencies, it may happen that two (or more) subprojects depend on the same libraries with some version restrictions. From all the possible combinations, only one that satisfies all dependent projects can be chosen, or if an incompatible request is made a resolution conflict appears. Again, a common approach is to use semantic versions [16] of the form `M.m.p`, where `M` stands for *major* version (one that is backwards incompatible), `m` is the *minor* version (one that is backwards compatible within the same major version) and `p` is a *patch*, a mere bug fix release that should be API compatible with other `M.m` releases. These dependencies are usually represented in some textual description of a release, like key-value lists, JSON, XML, or the own language syntax when it is interpreted.

Semantic versioning is not the only solution to the dependency upgrade issue, but in many cases semantic versions can encompass other paradigms like *calendar versioning* [17] that are less strict in their specification. At a minimum, pinning of versions and careful manual updating is a worst-case scenario that often is unavoidable if projects do not follow a strict backwards-compatible release policy.

3 Design objectives and use cases

For `alr`, after reviewing these solutions, the following decisions were taken, given the constraints of a lack of guaranteed funding and the idiosyncrasies of the Ada language and GNAT build tools:

- The objective is to help develop software, but not to configure the system. Hence, the mode of operation cannot depend on installing the compiled libraries, thus entirely avoiding any possibility of breaking the user's system. The tool operates in user-space and the libraries are stored as source code.

- A sandbox approach is applied for every working project to avoid variations due to build scenarios of a same dependency, which in turn ensures reproducible builds given a compiler/platform combination.
- To not depend on private servers nor live processes, the Alire catalog and code releases are stored in public Version Control System (VCS) services such as GitHub, BitBucket, etc., with which the open source community is used to work.
- New releases are incorporated into Alire by means of a pull request into the catalog repository. Since this is a manual process, at this time Alire can only be considered a curated system.[4]
- An indexed release is described using Ada code that is verified by means of compiling it, relying only on a single specification file that is part of the `alr` source code. The aim is to stay within the Ada realm as much as possible. In its present form, the `alr` tool only requires familiarity with the GNAT [18] compiler.
- Library developers should be minimally impacted for integration into Alire, if at all. This is achieved ultimately by only requiring a GNAT project file (GPR file) that can be created by Alire maintainers without bothering library authors uninterested in this tool.

Ada adopted the idea of library items [19] that can be submitted to the compiler independently. This concept, together with the well-defined dependency and elaboration rules, has spared Ada developers to an extent the quagmire of dependency-building tools such as autoconf, automake and CMake [20]. Given that nowadays there is a single open source Ada compiler, namely GNAT in its GPL and FSF editions, at this time `alr` relies on GNAT aggregate project files to completely manage the building process, without the need to modify the environment. This solution lets programmers use dependencies as usual, merely "with-ing" their project file.

3.1 Components of the Alire project

The Alire project is divided in the following main parts:

- The catalog of projects is a repository hosted under the name of `alire`.[5] It fulfills the same role as, e.g., the `crates.io-index`[6] project in the Rust community. It comprises the database of known projects and the minimal Ada types needed to represent that information. This way, commits to its repository should be for the most part, once development stabilizes, just additions to the catalog.
- The command-line tool available to users to interact with the Alire catalog is named `alr`, as its repository.[7] Again, this allows development on the tool with minimal disturbance to the catalog. It fulfills the role of the `cargo`[8] tool for Rust.
- The indexed code releases from third parties can be in any online repository, with the implicit assumption that the longest lived a repository is, the better. Current free offerings favored by developers are the usual suspects: GitHub, BitBucket, GitLab, etc. Of course, forks of particular releases could be made to ensure high availability.

3.2 Main use cases

Depending on the role of the user, a number of applications can be found for package managers. In its current form `alr` already enables the following use cases:

- Packagers: authors or entities wishing to disseminate their code can publish well-defined releases of their projects with the proper dependencies necessary to build them. Volunteers can also package popular Ada projects to increase their exposure. The Alire catalog knows about all licenses curated by GitHub[9], making explicit the rights granted by publishers.
- Developers: be it with the aim of publishing a project in Alire or not, developers can use `alr` to declare dependencies to be used in their own Ada projects. These dependencies are resolved into a valid solution, their code fetched, and a project file is generated that allows edition/compilation with the GNAT toolchain.
- Final users: despite the 'library' in Alire, more generally any packaged project can be also a binary tool or application. A single `alr get` command allows the retrieval, compilation and verification of target executables of such a binary project.

3.3 Introduction to `alr`

The prototype being discussed in this work is available for testing with a number of representative projects already indexed (see Fig. 6 in last page). Once installed and run without arguments, the user is greeted by the help screen shown in Listing 3.1, which will not be unfamiliar to similar tools users:

```
Ada Library Repository manager (alr)
Usage : alr [global options] command [options] [arguments]

Valid commands:

   build    Upgrade and compile current project
   clean    GPRclean project and dependency cache
   compile  GPRbuild current project
   depend   Manage dependencies of working project
   get      Fetch a project or show its metadata
   init     Create a project or generate its metadata
   list     See indexed projects in database
   pin      Pin dependencies to current versions
   run      Launch a project executable
   search   Search text in project names and properties
   test     Test deployment of releases
   update   Update alire catalog or project dependencies
   version  Shows alr diagnostics
   with     Locate index file of project

Use "alr help <command>" for information about a command
```

Listing 3.1: Help screen of `alr`

[4] The same happens in other languages. For example, in the Haskell community the Stackage project arose as a curated alternative to the `cabal-install` breakage-prone tool.

[5] `https://github.com/alire-project/alire`

[6] `https://github.com/rust-lang/crates.io-index`

[7] `https://github.com/alire-project/alr`

[8] `https://github.com/rust-lang/cargo`

[9] `https://choosealicense.com/`

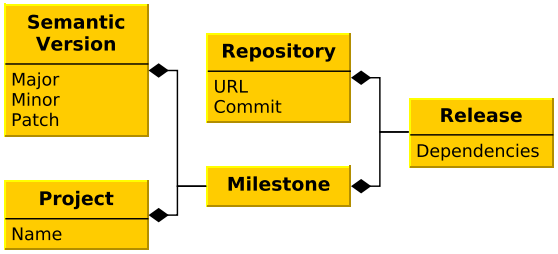


Figure 2: Entities in the Alire catalog.

Before diving into these commands, an explanation on the terminology being used (in the remainder of the paper and in the Alire source code) is in order (see also Fig. 2):

- A *project* refers to what also is typically called a library in the software world; e.g., GtkAda[10], AWS[11], etc.
- A *milestone* is a project name plus a semantic version; i.e., a particular version of a project.
- A *release* is the actual materialization of a milestone, available online and indexed by Alire. A release must provide one or more GPR files that build it.

The most straightforward function of `alr` is to retrieve a particular project and build it. Projects can contain libraries, which are useful to other projects, but also executables, in which case the compilation process will result in one or more executables ready to be run. This is achieved with the `alr get <project>` command. The result will be a folder containing the requested project and its dependencies, so compilation will immediately succeed.

Alternatively, `alr` can create new projects to start easily working within the Alire ecosystem. This is achieved with the `alr init [--bin|--lib] <project>` invocation. Initially the project will not have dependencies; required libraries can be added directly with `alr depend --add <project>` or with especially formatted comments in the user own GPR file.

Any project obtained by each of these two means can be called an alr-enabled or aware project, since it contains a metadata file that allows `alr` to perform its functions. Once within the folder tree of an alr-aware project, we can use the rest of commands (see Fig. 3). The `compile` command launches the `gprbuild` tool with a generated aggregate project file that makes dependencies available without needing to fiddle with paths. The `update` command refreshes the catalog and upgrades the dependencies of the working project.

There are also compound commands that group functions for common combinations: `run` will compile and then launch the resulting executable, whereas `build` will ensure that dependencies are up to date to then compile the project.

The commands interrelations have been designed to guarantee success, in the sense that compilation should always succeed if the requested dependencies are valid. `alr` will also detect the manual addition of new dependencies by the user and fetch them before a new compilation.

[10] `https://github.com/AdaCore/gtkada`
[11] `https://github.com/AdaCore/aws`

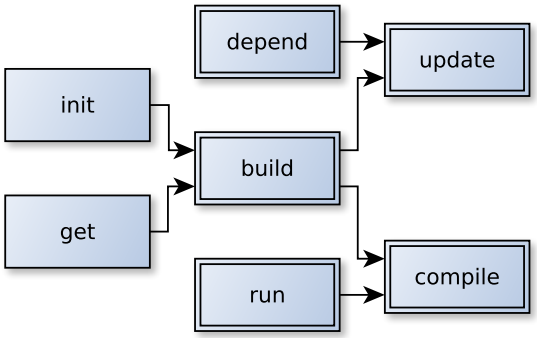


Figure 3: Relationships among commands. Single-frame commands can be used anywhere in the filesystem, whereas double-framed ones are to be used within an alr-enabled project.

To conclude this section, we show how dependencies are represented in a working project. As advanced, this is done in a package specification that can be compiled to verify its correctness, and which is initially generated by `alr`. A dependency on RxAda [21] has been already added.

```
with Alire.Index.Rxada;

package Alr_Deps is

   Current_Root : constant Root := Set_Root (
      "My_Shiny_Project",
      Dependencies =>
         Rxada.Project.Within_Major ("1.1"));

end Alr_Deps;
```

Listing 3.2: Metadata file in an alr-enabled project.

If the user wishes to compile this file directly (instead of through the `alr` tool), it is enough to add the Alire project itself as a dependency of the working project.

Restrictions on dependencies are described using the usual Ada comparison operators, and named functions for the semantic versioning specific operators caret (`'^'`) and tilde (`'~'`). This way there is no possible confusion on what is being asked for (In some implementations, the caret and tilde act differently on pre-1.0 versions). In the example, we request any future version of RxAda that is at least 1.1 but within the same major number, hence backwards-compatible.

4 Implementation details

This section presents some lower level details on `alr` implementation, particularly those aspects that present a specific idiosyncrasy of the tool when compared with its homologues for other languages.

GNAT is currently the only open source Ada compiler available, and its GPR project files are the preferred way to conveniently manage the building process. For these reasons, `alr` takes advantage of these project files, and in particular uses aggregate projects to make available the dependencies to be included in the compilation of a project.

4.1 Alire-mandated files

For `alr` to be able to perform its project-specific commands (see Fig. 3), it needs three critical files to be present:[12]

- `myproject.gpr` (henceforth the project file): this is a regular GPR project file that builds the project. In practice, GNAT projects typically already have one or several project files, so this is not a special requirement. `alr` provides ways of querying the name of these project files for the benefit of client projects.
- `alr_deps.ads` (henceforth the metadata file): this file is used by `alr` as a telltale that it is being run inside an alr-enabled project. It must contain the project name and its dependencies, as already shown in 3.2. It is initially generated by `alr init`, or could be hand-crafted if needed. It can also be regenerated on demand and manipulated through `alr depend`.
- `alr_build.gpr` (henceforth the environment file): this file is generated by `alr` to set up the environment paths required to find any projects the current project depends on. It can also be used to work in the GNAT GPS IDE.

Of these three files, the only one that is entirely the responsibility of the project author (or maintainer) is the `myproject.gpr` one. Its contents are arbitrary, as long as they succeed in building the library or executable. At a minimum, they must point the compiler to the source files of the project. On the other extreme, `alr_build.gpr` is regenerated by `alr` whenever necessary to properly configure the building environment (namely, whenever dependencies change or the file is not found). `alr_deps.ads` lies in the middle, since it is initially generated by `alr` but it must by tailored by the developer to their needs to indicate their dependencies.

Finally note that, for the inclusion of a project into the Alire catalog, only the project file is needed, since the contents of the metadata file will appear in the Alire index itself (see Listing 4.1), and the environment file is regenerated from that information. Each Alire index file contains the releases for the project named as the enclosing package. Besides textual versions, dependencies within the index can be specified using other indexed releases:

```
with Alire.Index.Libhello;

package Alire.Index.Hello is

   function Project is new Catalogued_Project
     ("""Hello, world!"" demonstration project");

   Repo : constant URL :=
      "https://github.com/alire-project/hello.git";

   V_1_0_1 : constant Release :=
      Project.Register
        (V ("1.0.1"),
         Git (Repo, "8cac0afdd"),
         Dependencies => Libhello.V_1_0.Within_Major);
   -- V_1_0 is an existing release of Libhello

end Alire.Index.Hello;
```

Listing 4.1: Release in the index with one dependency.

[12] "myproject" is a placeholder for an actual project name.

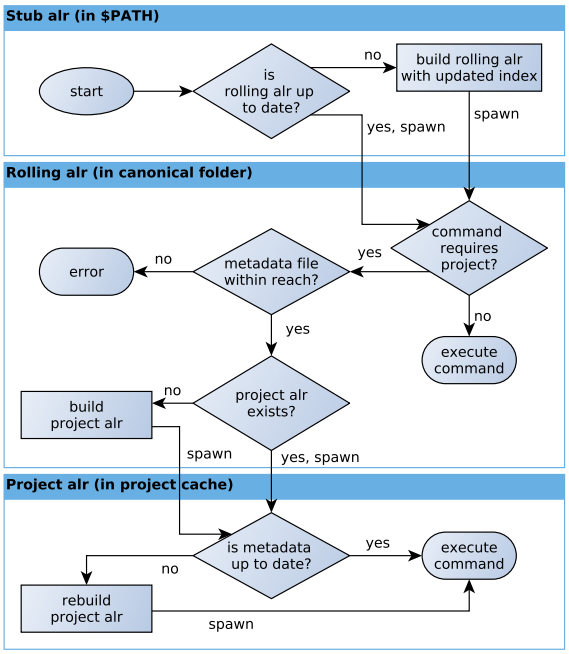


Figure 4: Launch sequence of `alr` for command execution.

A rich set of operations exists that allows the expression of not only simple dependencies, but also of conditional dependencies on the compiler version, platform properties, availability of native packages, and so forth. Also, to simplify indexing and clarifying the declarations, a base release can be taken as a template and modified with "extending" and "replacing" operations. For details the reader is directed to examples in the Alire database itself.[13]

4.2 Self-compilation of `alr` and working projects

Package managers are expected to have an up-to-date catalog, and also that the tool itself is up-to-date. In this case, a catalog update could be achieved in several ways: parsing text files that contain some specific format, or loading a binary database, for example. However, maintaining the tool up-to-date will involve compiling it from updated sources and replace the current executable. Also, incorporating the dependencies of a working project (parsing the `alr_deps.ads` file) would need either a custom parser or compilation and processing with ASIS [22] or a similar technology like libadalang [23].

As an alternative, `alr` solves all these necessities in a single and perhaps uncommon way: whenever the need is detected, `alr` recompiles itself, incorporating into the build fresh metadata and updated index files. This way all needed and up-to-date information is incorporated into `alr` without the need to parse any external files, since the compiler already does the work for us.

To manage this process of self-compilation, up to three different `alr` executables may exist and be called in succession, with specific responsibilities. All three come from the same sources, with a small set of variations for the specific purposes, and are deployed in different locations (see Fig. 4):

[13] Syntax examples: `https://github.com/alire-project/alire/blob/master/index/alire-index-alire.ads`

1. A *stub* `alr` is built during installation (or could be provided by the platform). This binary is never recompiled, acting as a fallback, and will typically be in the system PATH. Its purpose is to build the *rolling* `alr` from updated sources (for example to include new index releases) and launch it.
2. The *rolling* `alr` has an updated index and can already execute commands that are not project-specific (see Fig. 3). If, however, the command requires a project, and furthermore a project metadata file is in scope, then it builds and launchs a *project* `alr`, incorporating into the build the metadata file of the project (and so compiling-in the project dependencies).
3. The *project* `alr` contains the project metadata and is able to carry out project-specific actions. Prior to doing so, it checks its self-consistency by comparing the hash of the metadata file in scope with a hash stored internally that was computed by the *rolling* `alr`. If they do not match, this means that the *project* `alr` is outdated, in which case it is rebuilt with current metadata and launched to take over the command.

4.3 Final example

The creation of new projects from templates or downloading of releases do not really merit any special discussion, since they do not pose particular technological challenges. However, inspecting the filesystem after the issuing of an `alr get --compile hello` command will allow to bring into focus everything that has been reported up to this point. This command simultaneously fetches a project and its dependencies, generates the needed files and builds the whole configuration. The project itself is a plain "Hello, world!" example artificially split into having to depend on a library (libhello) that performs the actual output to the terminal.

Fig. 5 shows the relevant parts of a filesystem in which such a command were issued in the user's home folder. From top to bottom, the following relevant folders and files can be located:

- The stub alr can be anywhere in the user's path, here shown in `/usr/bin/alr`.
- `$XDG_CONFIG_HOME/alire/` is the canonical location in which updated sources are checked out. Inside, the `alire/index/` folder contains the catalog files, and the most recently built rolling `alr` executable is found in `alr/bin/alr`.
- `hello_1.0.1_65725c20/` is the folder in which the requested project, `hello`, has been deployed. The semantic version and abbreviated commit hash are appended to univocally identify the project. The project own organization is an internal affair of the project author; in this example the minimal project and main files are shown. Alire files can be found inside the `alire` subfolder:
- `<project>/alire/` contains firstly the metadata and environment files. The metadata file can be manually edited or manipulated through `alr depend`. The build file is regenerated on changes to the metadata file, and is useful to launch builds, or to edit from GPS.

```
/
├── usr/bin/alr (stub)
└── $HOME/
    ├── $XDG_CONFIG_HOME/
    │   └── alire/
    │       ├── alire/index/ (contents omitted)
    │       └── alr/
    │           ├── bin/alr (rolling)
    │           └── src/ (contents omitted)
    └── hello_1.0.1_65725c20/
        ├── alire
        │   ├── alr_build.gpr
        │   ├── alr_deps.ads
        │   └── cache/
        │       ├── projects/
        │       │   └── libhello_1.0.0_ce78e770/ (contents omitted)
        │       └── session/
        │           ├── alr (project)
        │           ├── alr_deps.ads
        │           └── alr-session.ads
        ├── hello.adb
        └── hello.gpr
```

Figure 5: Filesystem details (comments parenthesized).

- `<project>/alire/cache/`, finally, contains files that the user does not need to directly know about, and that can be deleted at any time since `alr` can download or generate them again as needed. The `projects/` folder contains downloaded dependencies (in this case a particular release of the libhello dependency). The `session/` folder contains generated files for the project build of `alr`. This folder is passed as-is to GPRbuild so it finds the following files:
  - `alr_deps.ads` is a copy of the metadata file.
  - `alr-session.ads` is a file generated at every rebuild that stores the hash of the current metadata, among other information.
  - `alr` is the built project `alr`.

The generated environment file for this example is shown in Listing 4.2:

```
aggregate project Alr_Build is

   for Project_Files use (" ../ hello.gpr");
   -- Root project being compiled

   for Project_Path use
     ("cache/projects/libhello_1.0.0_ce78e770");
   -- Project file paths of dependencies

   for External ("ALIRE") use "True";
   -- Flag that this is an Alire build

end Alr_Build;
```

Listing 4.2: The environment file is a GPR aggregate project file.

4.4 Discussion

At the time of this writing `alr` offers commands and features that make feasible the distribution and reuse of Ada libraries exclusively using Ada tooling and free, public repositories. (Appropriate index files could also enable its use within private environments.) Rich dependencies can be expressed conditionally, and native packages can be used where available. Finally, triggers allow the execution of external programs

at the post-fetch and post-build stages. These features are enough to cover a wide range of needs expected from typical source-oriented package managers.

Substantive effort has been devoted to the testing of both the tool and the packaged projects: through continuous integration, every `alr` master commit is tested to vet proper operation of the `alr` commands, and to verify that releases build properly in supported platforms (which include Debian testing, Ubuntu LTS, and GPL 2017 at this time). Outstanding open issues are:

Windows port: although technically not difficult, the lack of a platform package manager would limit the initial availability of projects with complex dependencies (e.g., GtkAda) that are natively supported in Linux variants.

Cross-platform builds: given the relevance of Ada in the embedded world, this is a feature that has already been pointed out to be important, and that is slated for inclusion in a future release if ongoing interest in `alr` is evidenced.

Given the presented design, compilation times of `alr` itself could be a point of contention since such compilations happen every time the metadata file changes (i.e., whenever dependencies are added or removed). To assess that point, experimental runs were conducted for different catalog sizes. However, since only a few files are recompiled every time (session and metadata files, and one body that uses them in `alr`), the impact is mostly limited to the time it takes to redo the binding and linking. Times measured with a middle-range[14] computer are shown in Table 1. Although not negligible, there is wiggle room until the issue becomes a pressing bottleneck.

	Indexed files		
Releases per file	100	1000	10000
1	1.82	3.73	34.09
10	1.94	4.52	44.83

Table 1: Average times (in seconds) for 100 `alr` recompilations after metadata changes, for different number of files in the catalog and releases per file. Compiler version was GNAT GPL 2017 using `-j0` switch.

5 Conclusions

This work presented an Ada tool, its underlying design, and supporting infrastructure that facilitates easy dissemination and reuse of third-party Ada projects. This is achieved by indexing and tagging code releases in public repositories with a semantic version, which in turn enables the possibility of dependency resolution and easy upgrades. The whole setup only requires a recent GNAT Ada compiler and enables effortless downloading and compilation of indexed projects.

The design is based around a metadata file which is itself written in Ada and incorporated into the tool by recompilation triggered by the tool itself, when needed. This process allows users and developers of the tool alike to remain within the realm of pure Ada code. The Ada syntax employed in index files has a rich feature set that allows the expression of complex conditional dependencies on the availability of native packages or other platform characteristics. This syntax is however only relevant to packagers, since users can add or remove dependencies through tool commands.

Alire is available under an open source license to interested parties at `https://github.com/alire-project`.

[14] Intel® Core™ i3-2015 (4 execution threads), 16GB RAM, SSHD disk.

Acknowledgments

This work has been supported by projects ROBOCHALLENGE (DPI2016-76676-R-AEI/FEDER-UE), ESTER (CUD2017-18), SIVINDRA (UZCUD2017-TEC-06) and ROPERT (DGA-T45_17R/FSE). The author thanks the regulars at `comp.lang.ada` for insightful discussions on the topic.

References

[1] I. Newton, H. W. Turnbull, and J. F. Scott (1959), *The correspondence of Isaac Newton / edited by H.W. Turnbull*. Published for the Royal Society at the University Press Cambridge.

[2] C. Peterson, *How I coined the term 'open source'*. Available at `https://opensource.com/article/18/2/coining-term-open-source-software`.

[3] M. Odersky and A. Moors (2009), *Fighting bit rot with types (experience report: Scala collections*, in LIPIcs-Leibniz Int. Proceedings in Informatics, vol. 4, Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[4] G. K. Thiruvathukal (2004), *Gentoo linux: the next generation of linux*, Computing in science & engineering, vol. 6, no. 5, pp. 66–74.

[5] L. Brenta and S. Leake, *Debian policy for Ada*. Available at `https://people.debian.org/~lbrenta/debian-ada-policy.html`.

[6] S. Eisenbach, V. Jurisic, and C. Sadler (2003), *Managing the evolution of .NET programs*, in International Conference on Formal Methods for Open Object-Based Distributed Systems, pp. 185–198, Springer.

[7] A. Stefik and S. Hanenberg (2014), *The programming language wars: Questions and responsibilities for the programming language community*, in Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, pp. 283–299, ACM.

[8] N. D. Matsakis and F. S. Klock II (2014), *The Rust language*, ACM SIGAda Ada Letters, vol. 34, no. 3, pp. 103–104.

[9] D. Hamilton and P. Pape (2017), *20 years after the mandate*, CrossTalk, p. 15.

[10] Ada Information Clearinghouse, *Ada free tools and libraries*. Available at `http://www.adaic.org/ada-resources/tools-libraries/`.

```
$ alr search --list
NAME                                VERSION             DESCRIPTION
ada_lua                             0.0.0-5.3           An Ada binding for Lua
adacurses                           6.0.0               Wrapper on different packagings of NcursesAda
adayaml                             0.3.0               Experimental YAML 1.3 implementation in Ada
adayaml.server                      0.3.0               Server component
alire                               0.4.0               Alire project catalog and support files
alr                                 0.4.0               Command-line tool from the Alire project
apq                                 3.2.0               APQ Ada95 Database Library (core)
aunit                               2017.0.0            Ada unit test framework
eagle_lander                        1.0.0               Apollo 11 lunar lander simulator (Ada/Gtk/Cairo)
globe_3d                            20180111.0.0        GL Object Based Engine for 3D in Ada
hangman                             1.0.0               Hangman game for the console
hello                               1.0.1               "Hello, world!" demonstration project
libadacrypt                         0.8.7               A crypto library for Ada with a nice API
libhello                            1.0.0               "Hello, world!" demonstration project support library
mathpaqs                            20180114.0.0        A collection of mathematical, 100% portable, packages
openglada                           0.6.0               Thick Ada binding for OpenGL and GLFW
pragmarc                            2017.2007.0         PragmAda Reusable Components (PragmARCs)
rxada                               0.1.0               RxAda port of the Rx framework
sdlada                              2.3.1               Ada 2012 bindings to SDL 2
semantic_versioning                 0.3.1               Semantic Versioning in Ada
simple_components.connections       4.27.0              Simple Components (clients/servers)
simple_components.connections.ntp   4.27.0              Simple Components (Network Time Protocol)
simple_components.connections.secure 4.27.0             Simple Components (clients/servers over TLS)
simple_components.core              4.27.0              Simple Components (core components)
simple_components.odbc              4.27.0              Simple Components (ODBC bindings)
simple_components.sqlite            4.27.0              Simple Components (SQLite)
simple_components.strings_edit      4.27.0              Simple Components (strings)
simple_components.tables            4.27.0              Simple Components (tables)
simple_logging                      1.0.0               Simple logging to console
steamsky                            2.1.0-dev           Roguelike in sky with steampunk theme
whitakers_words                     2017.9.10           William Whitaker's WORDS, a Latin dictionary
xml_ez_out                          1.6.0               Creation of XML-formatted output from Ada programs
```

Figure 6: Current alr catalog as listed by the `alr search` command.

[11] K. Reitz and T. Schlusser (2016), *The Hitchhiker's Guide to Python: Best Practices for Development*, O'Reilly Media, Inc.

[12] F. Tuong, F. Le Fessant, and T. Gazagnaire (2012), *OPAM: an OCaml package manager*, in SIGPLAN OCaml Users and Developers Workshop.

[13] E. Dolstra and A. Löh (2008), *NixOS: A purely functional linux distribution*, ACM Sigplan Notices, vol. 43, no. 9, pp. 367–378.

[14] B. Muschko (2014), *Gradle in action*, Manning.

[15] Conan, the C / C++ package manager for developers. Available at `https://conan.io/`.

[16] S. Raemaekers, A. Van Deursen, and J. Visser (2014), *Semantic versioning versus breaking changes: A study of the maven repository*, in 14th Int. Conf. on Source Code Analysis and Manipulation (SCAM), pp. 215–224.

[17] M. Hashemi (2016), *Calendar versioning*. Available at `http://calver.org/`.

[18] E. Schonberg and B. Banner (1994), *The GNAT project: a GNU-Ada 9X compiler*, in Proceedings of the conference on TRI-Ada'94, pp. 48–57, ACM.

[19] S. T. Taft, R. A. Duff, R. L. Brukardt, E. Ploedereder, P. Leroy, and E. Schonberg (2014), *Ada 2012 Reference Manual. Language and Standard Libraries: Int. Standard ISO/IEC 8652/2012 (E)*, vol. 8339, Springer.

[20] J. Al-Kofahi, T. N. Nguyen, and C. Kästner (2016), *Escaping AutoHell: a vision for automated analysis and migration of autotools build systems*, in 4th Int. Workshop on Release Engineering, pp. 12–15, ACM.

[21] A. R. Mosteo (2017), *Rxada: An Ada implementation of the ReactiveX API*, in Ada-Europe International Conference on Reliable Software Technologies, pp. 153–166, Springer.

[22] C. Colket (1995), *Ada semantic interface specification ASIS*, ACM SIGAda Ada Letters, no. 4, pp. 50–63.

[23] P.-M. de Rodat and R. Amiard (2018), *Easy ada tooling with libadalang*, in Ada-Europe International Conference on Reliable Software Technologies.

The IRONSIDES Project: Final Report

Barry S. Fagin and Martin C. Carlisle

Dept of Computer Science, US Air Force Academy 80840 Tel: 1-719-333-3338; email: barry.fagin@usafa.edu.

Abstract

In a project intended to improve the security of internet software, the authors developed IRONSIDES: A DNS server written in Ada/SPARK. Our long-term goals were a) to show that a fully functional component of the internet software suite could be written with provably better security properties than existing alternatives, b) to show that it could be done within the relatively modest re-sources available for a research project at an undergraduate university, c) to determine the suitability of Ada/SPARK for such a project, and d) to compare the performance of the resulting software to existing alternatives and determine to what extent, if any, the addition of provable security properties affects performance. We report our conclusions from this multi-year project.

Keywords: Ada, DNS, formal methods, internet software, performance analysis, SPARK.

1 Introduction

The Domain Name System (DNS) is the internet protocol that transforms hostnames (e.g. cnn.com) into IP addresses (e.g. 151.101.0.73). Originally proposed by Mockapetris in [1], it is a distributed database protocol that uses the internet as a tree structure to manage records containing information about machine names and properties.

Software that implements this protocol is referred to as a DNS server. This term can also describe the machine that runs a DNS server. Servers responsible for resolving names in a single zone (typically a company, university, or similarly scoped institution) are called authoritative servers. If queried about names outside the zone for which they are responsible, authoritative servers reply with a failure message, the equivalent of "I don't know".

Servers capable of resolving names for any publicly visible machine on the internet are called recursive. They use a recursive process to travel up the distributed internet tree structure to determine the name of the machine in question. In modern DNS practice, most recursive solvers do not use full recursion to traverse the name tree. Instead, they refer queries to a publicly available fully recursive DNS server (for example, Google's public DNS at 8.8.8.8), and then cache the result for future use.

DNS is a vital internet protocol. Unfortunately, because it dates from the early days of networking, it contains security flaws that require mitigation to prevent malicious actors from exploiting the system [2]. Additionally, most DNS software is written in older languages with inherent security problems. These languages do not lend themselves to rigorous software design and provable security properties. The two most popular DNS servers, BIND and WINDNS, have a large number of known security flaws, including crashing in response to the injection of bad data and bugs that permit remote execution [3], [4]. These are described in more detail in the sections that follow.

2 The IRONSIDES Project

The authors believed many of the security problems with DNS servers, web servers, and other internet software could be avoided with the use of better programming tools, such as the use of different programming languages and formal methods. They chose Ada and SPARK as an appropriate development environment to implement a provably secure DNS server from the ground up.

The SPARK language and toolset from Altran UK is used in the creation of software systems with provable correctness and security properties [5]. SPARK is a subset of Ada, augmented with special annotations. These annotations appear as ordinary comments to Ada compilers, but are visible to SPARK's pre-processing tools used to validate software. SPARK is a mature technology and has been used on several projects, including an open-source OS kernel provably free from runtime errors [6], the British Air Traffic Control System [7], and multi-level security workstations [8]. Accordingly, given our prior institutional experience with Ada [9], we chose SPARK and Ada as the platform for constructing DNS software that would not be subject to most of the vulnerabilities that afflict DNS implementations currently deployed around the world.

The SPARK toolset generates verification conditions (VC's) that it then attempts to verify. VCs include assertions that variables always remain in type, array bounds are never exceeded (a common source for buffer overflow vulnerabilities), pre- and post- conditions are always met, and so forth. When a VC has been proved by SPARK, it is said to be discharged.

2.1 Milestone 1: An authoritative server on Ubuntu

The first IRONSIDES milestone was achieved with the successful construction of an authoritative server, tested against BIND on Ubuntu [10]. The original test bed and performance results are shown in Figure 1 and Figure 2, respectively.

We were pleasantly pleased to discover that the authoritative IRONSIDES DNS server performed significantly better than BIND under Linux.

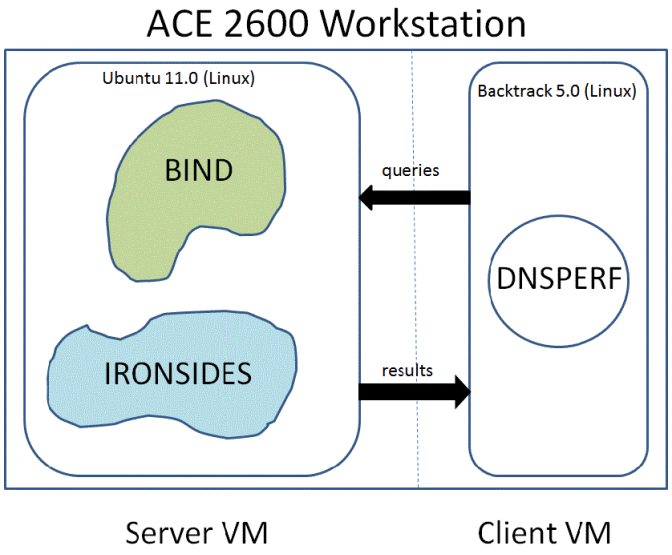


Figure 2 IRONSIDES original test bed for authoritative servers

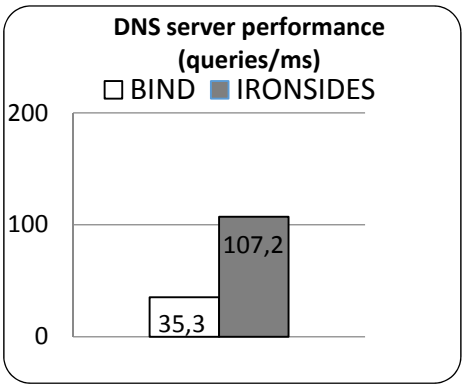


Figure 3 BIND and IRONSIDES performance under Linux

2.2 Milestone 2: An authoritative server on Windows

The next milestone was porting IRONSIDES to Windows, and testing it against both WinDNS and BIND [11]. The test bed was similar, except the virtual machine used ran Windows Server 2008. Performance results are shown below:

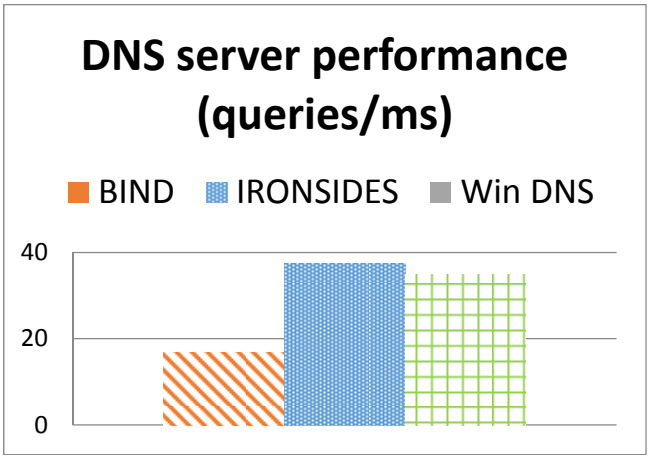


Figure 4 Performance comparison on Windows

We fully expected IRONSIDES to perform better than BIND, but were surprised to find it outperformed Windows DNS on its own native OS by 7%.

At this point, the proof requirements of IRONSIDES looked like this:

Table 1 Proof requirements of IRONSIDES authoritative

	Total	Simplifier	Victor	Examiner
Assert/Post	3106	2209	884	13
Precondition	561	0	532	29
Check stmt.	12	0	12	0
Runtime check	3750	0	3704	46
Refinement. VC s	44	42	2	0
Inherit. VCs	0	0	0	0
Totals:	7473	2251	5134	88
%Totals:		30%	69%	%

Victor invokes an optional theorem-prover to discharge VC's that the first two stages of the tools (the Examiner and the Simplifier) cannot.

2.3 Milestone 3: A recursive server and detailed performance comparisons

Recursive servers are more complex than authoritative ones, requiring more sophisticated data structures, cache management, and tasking. Building on our experience with the authoritative version, we next added recursive query functionality to IRONSIDES. The resulting basic structure (little changed to the present day) is shown in Figure 4.

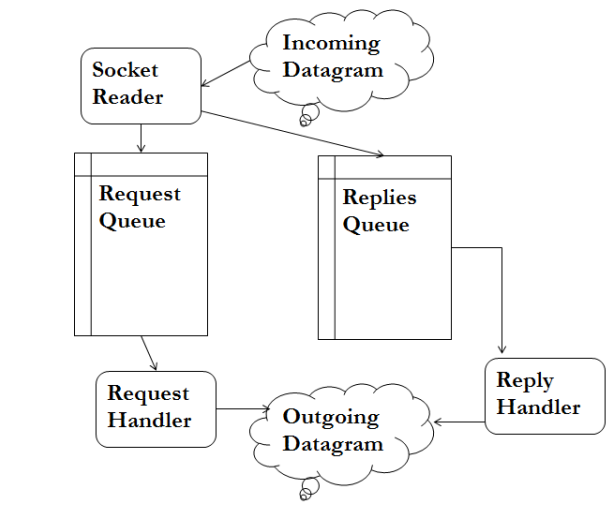


Figure 5. IRONSIDES recursive service structure

This structure was implemented with the modules and data dependency relationships shown in Figure 5.

Lines indicate a data dependency, transitive dependencies are implied. Their functions are:

- spark_dns_main: Top-level executable.
- udp_query_task: Concurrently executing task responsible for all incoming DNS traffic.

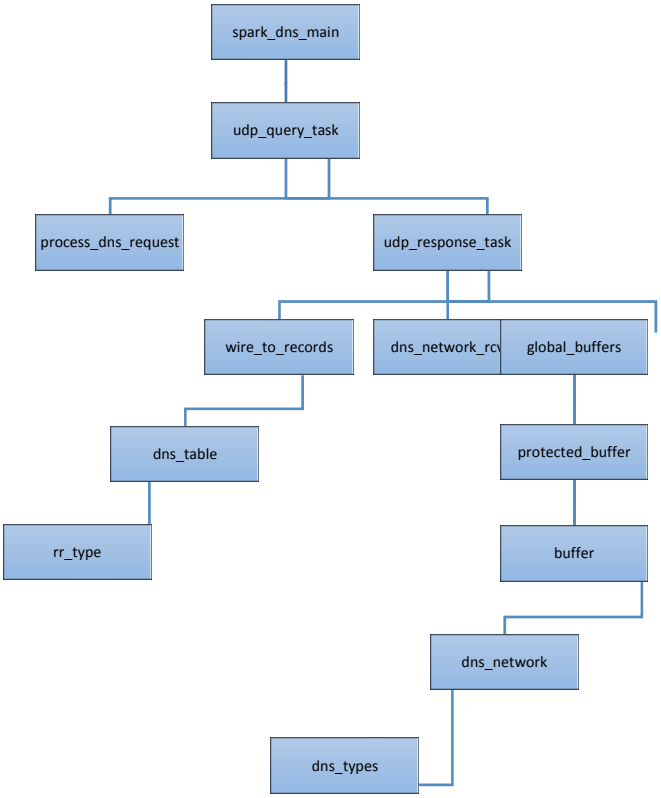


Figure 5. IRONSIDES recursive service structure implementation

- udp_response_task: Concurrently executing task responsible for managing all responses from upstream servers.
- process_dns_request: Interprets incoming packet, queries DNS table, queues query if answer not found.
- wire_to_records: Builds DNS resource records from DNS packets on the wire.
- dns_network_rcv: SPARK wrapper for network traffic to guarantee no overflows.
- global_buffers: Query and response queues.
- protected_buffer: ADT for the query and response queues.
- buffer_pkg: ADT for a queue.
- dns_table: Cache of DNS resource records.
- rr_type: Top-level package for all DNS resource record types.
- dns_network: Handles low-level network IO.
- dns_types: Data types for working with DNS packets.

The proof requirements for the recursive version were:

	Total	Exam.	Simp.	Victor
Assert/Post	3510	2248	1194	68
Precondition	641	0	609	32
Runtime check	9705	0	9502	203
Refinem. VCs	98	98	0	0
Totals:	13954	2346	11305	303
%Totals:		17%	81%	2%

For static code size, we measured the following:

Total Lines	14448
Blank Lines	1268
Comments	4142
SPARK Lines:	1713
Ada lines	9038
Ada statements	6917
SPARK statements	806

Once we had produced a validated recursive server, we were ready to do a detailed performance comparison with a variety of both open-source and proprietary DNS servers [12]. As shown in Figure 6, 2e expanded the test bed to include a virtual machine running each server/OS combination, a VM running the Resperf performance analyzer [13], and a VM running the network simulator INETSIM [14].

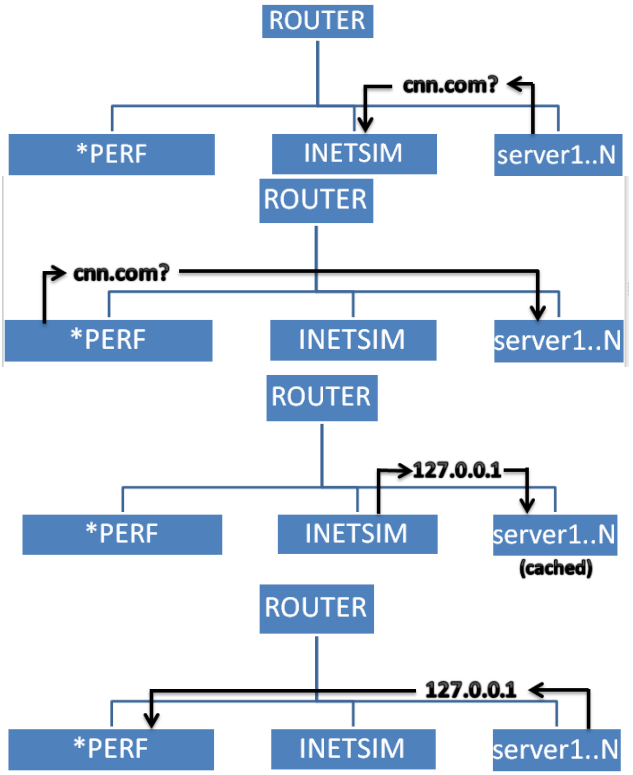


Figure 6 Recursive server test bed

When we ran the server in authoritative mode under Ubuntu, IRONSIDES continued to outperform BIND and others, although the gap had narrowed from about 3x to about 2x, as shown in Figure 7.

Figure 8 shows how under Windows, however, WinDNS now performed slightly better, perhaps due to improvements in later releases or the increased complexity of IRONSIDES required to support recursive queries.

The number of queries handled as a function of increasing requests per second for recursive servers is shown in the two charts included in Figure 9.

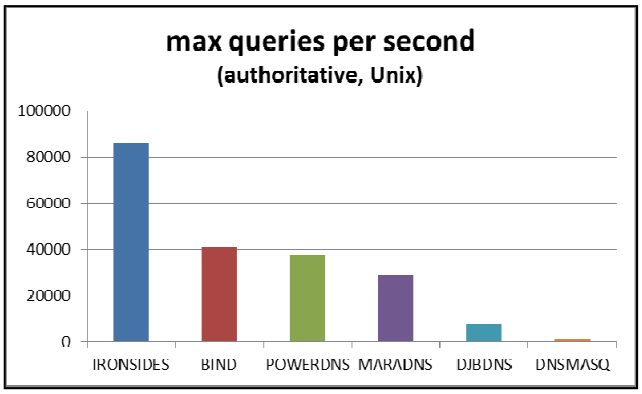


Figure 7 Performance comparison of authoritative DNS servers under Unix

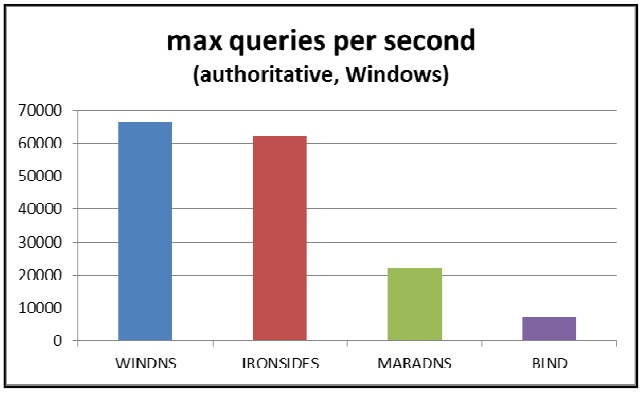


Figure 8 Performance comparison of authoritative servers under Windows

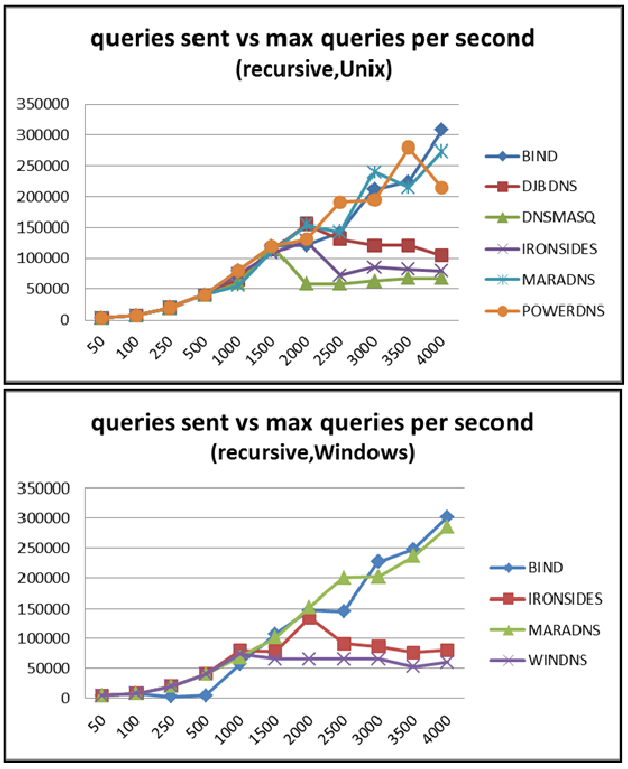


Figure 9 Performance of recursive servers

Up to 1500 queries per second, the performance of all the servers is essentially indistinguishable. At higher values, IRONSIDES, DNSMASQ and DJBDNS dropped off fairly rapidly. Surprisingly, under Windows, BIND also did the best.

On the other hand, in terms of queries lost, WinDNS and IRONSIDES performed best, as shown in Figure 10.

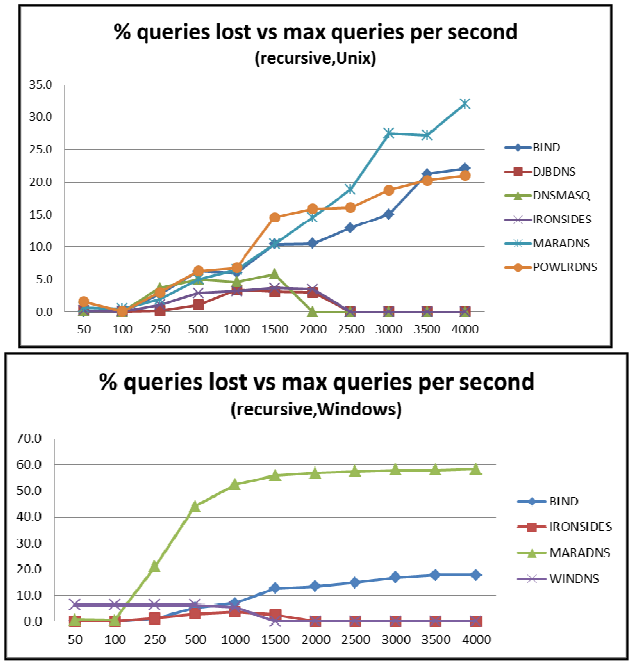


Figure 10 Queries lost by DNS servers

IRONSIDES had the second lowest latency for Unix DNS servers, but the longest latency for Windows servers. We believe this is due to latency being extremely important to Microsoft, and to IRONSIDES policy of trying to handle every query it can (BIND, by contrast, drops queries if it is too busy):

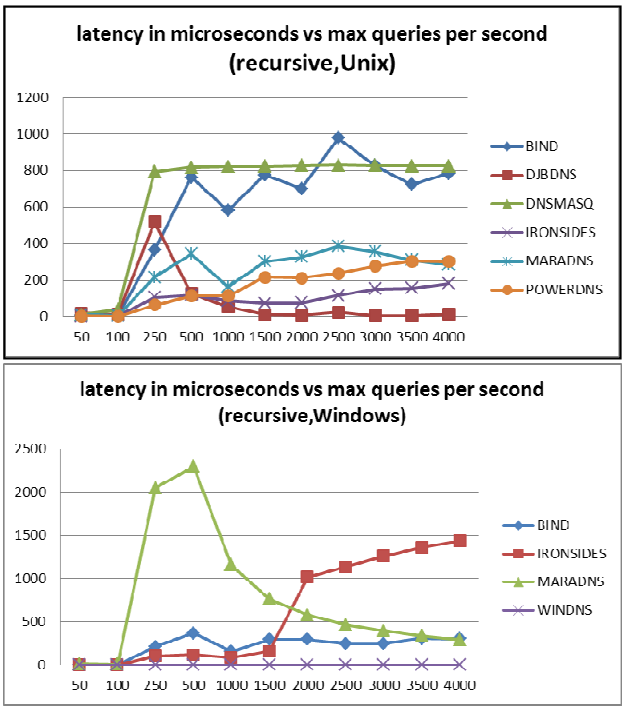


Figure 11 Latency of DNS servers

For further details, the reader is referred to the references.

3 Insights from experience

The results from the authoritative server design process gave our first hints that performance did not need to be sacrificed to improve security. In fact, there were clear examples in which the use of formal methods actually improved performance. For example, data flow analysis identified redundant or ineffective statements that in turn permit the removal of inefficient code. Code that has been proven exception-free no longer requires runtime bounds checking, so that can be eliminated as well.

We also learned, however, that there were cases were total reliance on formal methods and proof negatively impacted performance. Because SPARK requires all data structures to be statically allocated, data structures must be sized at the upper limits of expected use. Explicit initialization of such structures, while required for validation, is inefficient and wasteful. In those rare cases, we explicitly told the tools to relax that requirement. This improved IRONSIDES performance by almost 30%. Thus we believe allowing users to override formal proof requirements when appropriate is an important feature that formal methods tools should always support.

It is crucial to always remember the role of the compiler. Despite our confidence in the tools to help us produce crash-proof software, we found one combination of operating system, compiler and optimization level where a fully validated version of IRONSIDES crashed with an exception. This was due to a code generation error in the version of GNAT shipped with Ubuntu, long since corrected. Still, until formal methods have progressed sufficiently to the point where they can prove the correctness of compilers for a given target architecture and OS, programmers should continue to exercise healthy skepticism when compiling and testing verified software.

Our experience with the tools produced results we would describe as both impressive and humbling. Despite both of us having computer science PhD's, over 50 years of combined industry and academic experience, and an extensive knowledge of programming languages and software engineering practice, the tools still caught boundary conditions and potential problems that in principle we could have found but did not. This is the whole point of using formal systems, but the experience is nonetheless humbling. Perhaps it will become less so as formal methods and proof tools become a standard part of the software engineering process.

IRONSIDES has numerous provable security properties absent from all the other servers tested, including BIND and WinDNS. These include:

1) No classic buffer overflow
2) No incorrect calculation of buffer size
3) No improper initialization
4) No ineffective statements
5) No integer overflow/wraparound
6) No information leakage
7) All input validated
8) No allocation w/o limits (no resource exhaustion)
9) No improper array indexing
10) No null pointer dereferencing
11) No expired pointer dereferencing (use after free)
12) No type confusion
13) No race conditions
14) No incorrect conversions
15) No uncontrolled format strings
16) All loops guaranteed to terminate

With all these advantages, we were pleased to discover that IRONSIDES also performs comparably to servers with security problems, including the industry standards of BIND and WinDNS. IRONSIDES offers comparable performance at nominal loads, trailing off only under maximal loading. This is particularly significant considering each server's respective development costs. BIND is produced with an industrial consortium. WinDNS is bundled with the flagship product of a multibillion dollar software company. IRONSIDES was written by the equivalent of a little more than one professor at an undergraduate university with near full-time teaching duties.

4 Conclusions and future work

The success of IRONSIDES indicates that formal methods can be used both improve the security properties of software without incurring significant performance penalties, and in some cases can actually improve performance. This was done in an environment with significantly fewer resources available than comparable products.

Why then are similar approaches not more widely adopted? Existing products have greater sunk costs. In all fairness to them, they also offer more functionality and a better support base, options that are not available to the time and resource constrained environments present in the academic development of prototype software. Programming with formal methods also requires the use of a relatively unfamiliar language (particularly in the United States) as well as comfort with mathematical logic and proof. Most software engineers are not yet trained to work this way. Perhaps, however, over time this will change as the advantages of formal methods and proof are shown to produce more reliable software that keeps a company's name out of the newspapers.

Since the latest stable release of IRONSIDES, Ada has added more features that meld it more tightly with SPARK. We hope to upgrade IRONSIDES in the future to support these features and to examine their effects.

We hope this work will be further extended to apply formal methods and performance analysis outside the DNS domain, in the hopes of continued confirmation that internet software can be made provably more secure without significant sacrifices in performance. Web servers, for example, suffer from similar security problems for similar reasons. ICS and SCADA systems are currently

attractive targets for hacking, and formal methods have been used to improve their security [30], but the effect of formal methods on performance in this domain remains unknown. These are the subject of current work at the Academy Center for Cyberspace Research.

Acknowledgments

This work was supported in part by the US Air Force Office of Scientific Research under grant #1220961, the US Department of Defense Advanced Research Projects Agency, and the Academy Center for Cyberspace Research.

References

[1] https://tools.ietf.org/html/rfc882.

[2] Carnegie Mellon University Software Engineering Institute (2008), *Multiple DNS implementations vulnerable to cache poisoning*, available online at http://www.kb.cert.org/vuls/id/800113.

[3] Internet Security Consortium, BIND *9 Security Vulnerability Matrix*, available online at https://kb.isc.org/article/AA-00913/0/ BIND-9-Security-Vulnerability-Matrix.html

[4] https://technet.microsoft.com/library/security/MS15-127

[5] J. Barnes (2003), *High Integrity Software: The SPARK Approach to Safety and Security*, Addison-Wesley Publishing, 0-321-13616-0, ©.

[6] R. Buerki and A. Rueegsegger (2013), *Muen - an x86/64 separation kernel for high assurance*, Technical Report, University of Applied Sciences Rapperswil (HSR), Switzerland. Available on line at http://people.cs.ksu.edu/~danielwang/Investigation/Formal_Verification/muen-report.pdf.

[7] AdaCore (2017), *GNAT Pro chosen for UK's next generation ATC system*, AdaCore Technologies press release, available online at http://www.adacore.com/customers/uks-next-generation-atc-system/.

[8] AdaCore (2010), *Spark PRO adopted by secunet*, AdaCore Technologies press release, available online at http://www.adacore.com/customers/multi-level-security-workstation/

[9] R. Sward, M. Carlisle, B. Fagin and D. Gibson (2003), *The case for Ada at the USAF Academy*, Proceedings of the ACM SIGAda International Conference on Ada pp 68-70.

[10] M. Carlisle and B. Fagin (2012), *IRONSIDES: DNS with no single-packet denial of service or remote code execution vulnerabilities*, GLOBECOMM 2012, Anaheim CA.

[11] B. Fagin and M. Carlisle (2013), *Provably secure DNS: A case study in reliable software,* 2013 International Conference on Reliable Software Technologies, Berlin, Germany pp 81-93.

[12] B. Fagin, M. Carlisle and B. Klanderman (2017), *Making DNS Servers Resistant to Cyber Attacks*, COMPSAC 2017, Turin, Italy pp 566-571.

[13] http://www.nominum.com/measurement-tools/

[14] Internet Services Simulation Suite, http://www.inetsim.org/

[15] J. Groote, A. Osaiweran and J. Wsesselius (2011), *Analyzing the effects of formal methods on the development of industrial control software*, 2011 IEEE Conference on Software Maintenance, Williamsburg VA, pp 467-472.

[16] H. Boulakhrif (2015), *Analysis of DNS resolver performance measurements*, Masters' Thesis, University of Amsterdam, available at https://www.nlnetlabs.nl/downloads/publications/ os3-2015-rp2-hamza-boulakhrif.pdf.

Designing Multitask Control Software in a Multiprocessor World

Bo I. Sandén

Colorado Technical University, 4435 N. Chestnut, Colorado Springs, CO 80907, USA; Tel: +1 719-531-9045; email: bsanden@acm.org

Abstract

In single-processor real-time control systems, tasks are often scheduled to deadlines to ensure that they can all have a processor when needed. This can be done on multiprocessors too, but adding cores/processors can be simpler and more flexible than loading each core or processor as heavily as possible. This article compares two software solutions for a pick-and-place system: an existing deadline-driven implementation and an event-driven one without deadlines. The existing implementation schedules tasks to periodic deadlines on a dual-core processor. It uses elaborate mode-change logic to allow the controlled pick-and-place system to run at two predetermined speeds. The event-driven solution, on the other hand, requires additional cores or processors but could support a continuous range of speeds, which allows the controlled physical system to stop and restart without manual intervention. It can also be generic and support multiple configurations because deadlines need not be pre-computed for each. Arguably, the event-driven solution is easier to understand and change.

1 Introduction

Even as multicore and multiprocessor hardware is becoming ubiquitous, much real-time software design still focuses on processor scheduling: Making sure that every task gets enough processor time to meet its deadlines determines the software architecture. If the number of cores/processors is indeed severely limited, such solutions are useful when the hardware needs the results of certain computations by specific times. In that situation, they can improve on the single-task cyclic executives that have long prevailed in real-time software. A downside is the potential rigidity of a design where time constraints are built right into the architecture.

With multicore and multiprocessor hardware increasingly available, processor access may not remain the major concern. That means that deadline-driven architectures can often give way to an event-driven style. While a deadline-driven architecture may only be able to run a physical system at certain, constant speeds, an event-driven system could adjust its speed to the load, even stop entirely, and then restart automatically. Letting the physical system fall back gracefully to a maintenance mode can also be easier when we need not build a schedule and mode-change logic for each possible configuration. We shall illustrate this difference in complexity by means of a model problem with pick-and-place robots described by Saez et al [3]. It represents a class of useful control systems and is easy to understand yet nontrivial.

1.1 The pick-and-place problem

Shown in Figure 1, the pick-and-place system separates workpieces into cylinders and cubes. As pieces arrive on the input conveyor belt, a camera captures successive frames. A software module called *segmentation* determines the number of pieces and their positions in each frame. It inserts this into an *ImageBuffer*, which the *recognition* module uses to determine the type and orientation of each piece. It puts that information in a queue called *ToDo* together with the piece's position on the conveyor.

Depending on the piece count in each freshly segmented frame, either Robot0 alone or Robot0 and Robot1 both move pieces. Repeatedly, a robot first retrieves a *ToDo* entry describing a workpiece, then picks the piece off the input conveyor, and finally places it by type on one of two output conveyors.

The input conveyor runs at *fetching* speed until a nonempty frame is segmented, then at a slower *working* speed while pieces remain to be picked, and then again at *fetching* speed. There are three system states: *Fetch* with no robot picking, *Normal* with only Robot0 picking, and *Overload* with Robot0 and Robot1 both picking.

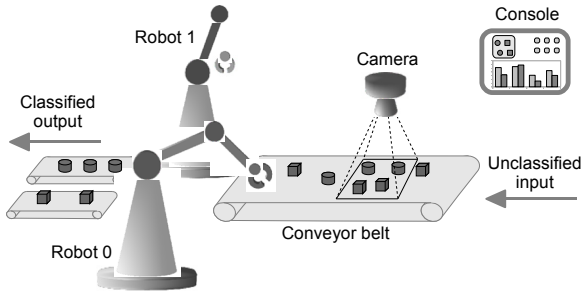


Figure 1 Schematic of the pick-and-place plant [3]

Unlike, say, the software controlling a car's airbags, which must complete its computation within so many milliseconds, the pick-and-place system imposes no deadlines directly on individual tasks. Instead the timing constraints apply to the speeds of the conveyors and robots:

- With workpieces continuously arriving, what are the greatest fetching and working speeds that let the two robots pick every piece in every frame?

- In state *Normal*, which maximum threshold piece-count allows a single robot to pick every piece?

Saez et al [3] describe a deadline-driven architecture that captures nicely and intuitively the concurrency inherent in the problem description. For example, each robot has an instance of a task type *robot* that takes it through its motions. Simplifying their solution somewhat, we introduce another task type, *frames*, with two instances, each of which executes *segmentation* and *recognition* for successive frames. With those two tasks, the processing of different frames can overlap.

We want to compare the deadline-driven solution and an event-driven architecture with the same set of tasks. Because it is more straightforward, we begin by describing the event-driven solution and then discuss the deadline-driven one.

2 Event-driven solution

The event-driven solution assumes that there are as many cores or processors as there are tasks, which eliminates the need for task scheduling altogether. Although the small number of tasks in this case is quite manageable, we could most likely do with fewer processors: For example, a single one may be sufficient for both *robot* tasks. No task is tied to a specific core or processor.

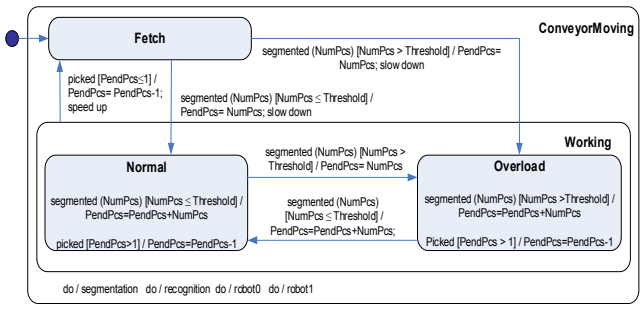


Figure 2 State diagram for event-driven solution [5]

The state diagram in Figure 2 shows the three system states and the superstates *ConveyorMoving* and *Working*. It labels each state-transition arrow with a triggering *event*, sometimes followed by a bracket with a *condition guarding* the transition, and finally, preceded by a slash, any *action* the system takes in response. Event/action pairs, such as "picked [PendPcs > 1] / PendPcs = PendPcs - 1" that do not involve a state change appear inside the icons of the states where they occur. The keyword "do" indicates an *activity*, which some task performs in a certain state or superstate.

In this solution, a robot need not do the placing in the same state where it picked a piece; one or two state transitions can intervene. More broadly, in an event-driven solution such as this, not all system-state changes necessarily affect all tasks immediately. Instead, each task queries the current state when it is prepared to react to a state change. For example, *robot1* finds out the system state once its physical robot has placed a workpiece and is at rest at a holding position. The task then blocks, if necessary, until the system is back in *Overload* and both robots are allowed to pick.

With Figure 2 as a roadmap, here is how this solution works:

- *Fetch* is marked as the initial state. The event *segmented (NumPcs)* signals that *segmentation* has identified a nonempty frame with *NumPcs* pieces. The next state depends on *NumPcs*:
  - If NumPcs ≤ Threshold, enter Normal.
  - If NumPcs > Threshold, enter Overload.
- In *Normal*, Robot0 alone picks. As each new frame is segmented, *NumPcs* is added to *PendPcs*, which is the number of pieces yet to be placed. State transitions occur in two cases:
  - If NumPcs > Threshold, enter Overload.
  - If PendPcs = 0, enter Fetch.
- In *Overload,* both robots pick. For each new frame, *NumPcs* is added to *PendPcs*. State transitions occur in two cases:
  - If *NumPcs* ≤ *Threshold*, enter *Normal*; Robot1 stops picking.
  - If *PendPcs* = 0, enter *Fetch.*

We implement the state machine as a protected object (PO). The tasks report the occurrences of significant events such as *segmented* or *picked* by calling the corresponding operations on that *state-machine PO* [4].

Another PO is *ToDo*, a queue of entries each holding the coordinates on the conveyor of one workpiece to be picked. The *frames* tasks feed it new entries. We want the robots to pick pieces roughly in the order they appear on the conveyor, so they should retrieve *ToDo* entries in that order. However, when two *frames* tasks run *recognition* concurrently, a leading piece in one frame may be submitted to *ToDo* before a trailing piece in the previous frame. *ToDo* can deliver the entries in the proper order by keeping track of each piece's frame number.

2.1 Resource contention

We assume that the robot arms can move between the input and output conveyors simultaneously without interfering with each other, and also that each robot has a holding position close to the input conveyor and another close to the output conveyors, out of the other robot's way.

Upon transition from *Normal* to *Overload*, both robots may be at their holding positions near the input conveyor ready to pick. Because each robot picks pieces located anywhere across the conveyor, it needs exclusive access to avoid clashing with the other robot. With a physical robot at rest, its task first consults *ToDo* under exclusive access. Once it has retrieved the coordinates of a piece to pick, it locks the input conveyor and then waits for its physical robot to pick and get out of the way before unlocking. In a similar fashion, a single lock can protect both output conveyors.

2.2 Generic solution

Absent the need for deadlines, we can pattern the tasks on concurrent processes found in the problem description [4, 5]. The tasks are generally the same as in the deadline-driven solution:

- The *robot* tasks represent two sequentially operating components of the physical system, which proceed independently except where they need exclusive access to a shared resource.
- Similarly, the *frames* tasks represent sequential processes, each working on a different frame.
- A state-machine PO maintains the system state, shown in Figure 2.
- Other POs ensure mutual exclusion of robots accessing the same conveyor.
- The PO *ToDo* is the interface between *frames* tasks and *robot* tasks.

The event-driven approach makes a generic solution possible where the working and fetching speeds as well as the threshold are parameters to be calculated for each particular installation. We can determine these speeds by simulation or by trial runs with each physical configuration at varying speeds and with realistic piece-count distributions. We can adjust the speeds and the threshold on the fly.

Because such trial runs will also reveal any shortage of processor time, separate schedulability analysis becomes unnecessary: If the system manages to pick and place all the pieces at a certain conveyor speed under all expected conditions, then that proves that all tasks involved have sufficient processor access. Because the speeds vary, this solution relies on markers along the input conveyor to determine the position of each piece.

To deal with unlikely scenarios such as a number of unusually heavily loaded frames arriving directly after one another, the input conveyor could slow down automatically and even stop when the robots are too far behind to pick all the pieces while they are still reachable. The robots would continue picking and placing while the conveyor slows down, possibly stops, and then speeds up. Many systems of this general type should also support planned and unplanned stops followed by restart without the need for manual cleanup.

To allow for robot maintenance, the system should be able to degrade gracefully to a state where a single robot does all the work at a lower conveyor speed. Robot0 and Robot1 should each be able to operate alone. At the points where a physical robot is at rest, its task could routinely check whether it is expected to stop working. If so, the task would report this event to the state-machine PO.

Such enhancements are not necessarily easy unless incorporated in the initial architecture, but can be designed to work in the same vein as the original system: We add necessary states and transitions to the system-state machine shown in Figure 2, and modify the existing tasks as necessary to reflect additional states local to each robot.

We would need to add a task only if, say, a third robot were somehow introduced. In that case, each output conveyor should have its own lock so two robots can place at the same time on different conveyors. A robot that must reach across one conveyor to the other would need both locks. To avoid deadlock, all robots must lock the conveyors in a static order as, for example, always the "cube conveyor" before the "cylinder conveyor".

3 Deadline-driven solution

In order to run the pick-and-place system software on a dual-core computer with barely sufficient processing power and on a single core whenever possible, Saez et al. [3] relied on task scheduling: When the conveyor moves at a constant speed, the timing can translate into hard task deadlines. They precalculated the *working* speed based on the segmentation time per frame plus the time needed to recognize and move each piece. They chose a *fetching* speed low enough to let the conveyor slow down when a nonempty frame arrives so the robots can pick all the pieces. Three distinct *modes* of operation, *FetchMode*, *NormalMode*, and *OverloadMode*, replace the three simple states in Figure 2.

Figure 3 corresponds to Figure 6 in [3], which primarily shows the workings of the mode manager, a module similar to the state-machine PO in some ways. In Figure 3, activities indicate which states allow which tasks to execute. For example, "do/*robot1*" in *OverloadMode* means that *robot1* runs in that superstate only, not in *NormalMode* or *FetchMode*; its physical robot picks and places only in *OverloadMode*.

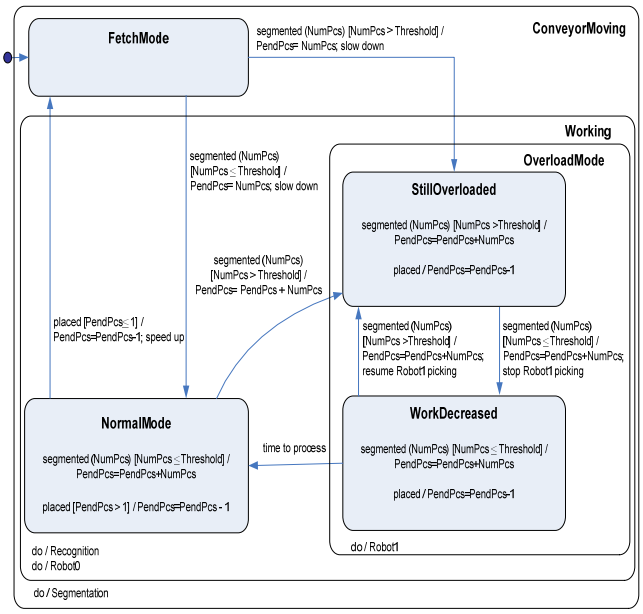


Figure 3 State diagram for deadline-driven solution

While the tasks are essentially the same as in the event-driven solution, one technical difference is that *segmentation* has its own single task while *recognition* is a task type with two instances, one for each core. Tasks run on a separate schedule for each core in each mode [3]:

- *FetchMode*: *Segmentation* alone runs on a single core.

- *NormalMode*: One *robot* task runs on a single core together with *recognition* and *segmentation* as follows:
  - Repeatedly, the camera captures a frame at time t, say.
  - *Segmentation* has its deadline at t+10, when *recognition* starts.
  - *Recognition*'s deadline is at t+50, when the next frame is captured.
  - The *robot* task gets three units of processor time in each successive interval of 10 time units.
- *OverloadMode* uses two cores, each running one *robot* task plus one *recognition* task. The *segmentation* task runs on each core by turns.

For each mode, the authors proved that the tasks indeed finish their computations by deadline. The following five mode transitions occur, each requiring a mode-change operation to phase continuing tasks into their new schedules as necessary:

- FetchMode to NormalMode
- FetchMode to OverloadMode
- NormalMode to FetchMode
- NormalMode to OverloadMode
- OverloadMode to NormalMode

The mode-manager module [2] handles these changes. While Figure 2 shows a transition from the superstate *Working* to *Fetch*, Saez et al [3] avoid the direct transition from *OverloadMode* to *FetchMode*, which would have been an additional mode change to be worked out. Instead, a transition to *FetchMode* occurs immediately after the mode change to *NormalMode* if *PendPcs* = 0. (Figure 3 does not show this.)

The situation in *NormalMode* illustrates the kind of resource conflict such careful task scheduling can resolve when there is only one core: The single robot may need relatively little processor support but is on the critical path as its physical movements limit the input conveyor's speed. *Recognition*, on the other hand, is computing intensive, and could occupy the processor for some time, analyzing one piece after the other. For this reason, the *robot* task needs higher priority so it can preempt *recognition* when necessary to keep its physical robot going.

The superstate *OverloadMode* is where the complications of the deadline-driven solution are most plain. It has two substates:

- *StillOverloaded*, which uses two robots and two *recognition* tasks. A state transition occurs in one case:
  - When *segmentation* finds a frame with *NumPcs* ≤ *Threshold*, enter *WorkDecreased*. Robot1 stops picking.
- *WorkDecreased*, which gives tasks that are not allowed to run in *NormalMode* time to wrap up ahead of the mode change. When *WorkDecreased* is entered, *recognition* of the previous frame may still be in progress, and Robot1 may be at any point in its cycle: In the worst case, it has just retrieved a *ToDo* entry and is poised to pick and place a piece. *WorkDecreased* allows a fixed amount of time for all this to complete. The event *time to process* signals the end of the allotted interval. State transitions occur in two cases:
  - When *time to process* occurs, enter *NormalMode* via a mode change.
  - If a frame with *NumPcs* > *Threshold* is segmented, return to *StillOverloaded*. Robot1 resumes picking along with Robot0.
- While most of the mode-change machinery is hidden in standard software modules, handling "wrap-up states" such as *WorkDecreased* becomes a part of the application development: The transition to the new mode must wait until all tasks not allowed to run there reach suitable stopping points in their logic. As in the case of *WorkDecreased*, a straightforward solution may be to allow a sufficient, fixed amount of time for this. While it might be more precise to exit the state as soon as all tasks are ready, that may require considerable coordination so that any task involved can determine whether it is the last one to wrap up and must initiate the mode change.

Although such wrap-up states can complicate the software design, *WorkDecreased* appears not to impact the efficiency of this particular system much. The picking and placing continues. However, if another task were poised to start upon entry to *NormalMode*, it might be delayed for a fixed period for no reason.

3.1 Setting the parameters for the deadline-driven solution

In reality, the deadline-driven solution is not as neatly periodic as it may look, nor does the input conveyor always run at one of its constant speeds: *FetchMode* actually has an acceleration substate reached upon transition from *NormalMode*, while *StillOverloaded* and *NormalMode* include deceleration substates reached upon the transitions from *FetchMode*. (Figure 3 does not show these substates.)

When a frame is segmented in *FetchMode*, where the input conveyor travels fast, the next frame comes sooner than in superstate *Working,* where the conveyor travels more slowly. In the worst case, that first frame is as full as physically possible. Taking a conservative approach, we can choose speeds slow enough to let the robots always move every piece before the next frame has been segmented. But in reality, a full frame may rarely follow an empty one, so the system might then be underutilized much of the time only to accommodate a rare case. The event-driven solution, on the other hand, can adjust the conveyor speed as necessary – and even stop the conveyor temporarily – to deal with such extreme cases.

The parameter *Threshold* must take into account not only the number of pieces a single robot can move per time unit but also the number of pieces that a single core can segment and recognize. The event-driven solution can

separate these concerns and make two *frames* tasks available in every state as discussed in section 2.

4 Conclusion

As we have seen, taking a multimoded, deadline-driven approach to the pick-and-place problem complicates the software relative to working with a few cores or processors more. While Saez et al [3] must be considered quality documentation of the design including its rationale, it focuses on the scheduling. Understanding the details of how and why the tasks interact is not always easy based on their paper only.

In both software solutions we have discussed, each task can be identified from the problem description without further software design. Each mirrors a sequential process in the problem environment: either the stepwise movements of a robot or the segmentation and recognition of successive frames. This simplifies the deadline-driven solution: If we want to stop one robot, for example, we stop exactly its corresponding task. Basing tasks on such sequential processes tends to be intuitive and works equally well with an event-driven design approach [4, 5].

Not every deadline-driven architecture may convert this smoothly into an event-driven one, however. In some architectures, tasks may be identified by means of functional decomposition. In such a solution, each input may "visit" multiple, periodically scheduled tasks one after the other [1]. In practice, this may produce many tasks, some of which may always run sequentially rather than concurrently, making it difficult to tell how many processors/cores are enough.

Acknowledgment

Jorge Real readily and helpfully answered many detailed questions on the workings of the pick-and-place system.

References

[1] R. Kazman, L. Bass, and M. Klein (2006), *The essential components of software architecture design and analysis,* Journal of Systems and Software, vol. 79, issue 8, pp 1207-1216.

[2] J. Real and A. Crespo (2004), *Mode-change protocols for real-time systems: A Survey and a new Proposal*, Real-Time Systems, vol. 26, issue 2, pp 161 -197.

[3] S. Saez, J. Real, and A. Crespo (2012). *An Integrated Framework for Multiprocessor, Multimoded Real-Time Applications* In: M. Brorsson and L. M., Pinho (eds.), Ada-Europe 2012. LNCS, vol. 7308, pp. 18-34. Springer, Heidelberg.

[4] B. I. Sandén (2011), *Design of Multithreaded Software: The Entity-Life Modeling Approach*, IEEE Computer Society Press/Wiley.

[5] B. I. Sandén (2017), *Entity-life modeling.* In Encyclopedia of Software Engineering, Taylor and Francis.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch