The journal for the international Ada community

Ada User Journal

Volume 43
Number 1
March 2022

Produced by Ada-Europe

Information on Subscriptions and Advertisements

Ada User Journal (ISSN 1381-6551) is published in one volume of four issues. The Journal is provided free of charge to members of Ada-Europe. Library subscription details can be obtained direct from the Ada-Europe General Secretary (contact details above). Claims for missing issues will be honoured free of charge, if made within three months of the publication date for the issues. Mail order, subscription information and enquiries to the Ada-Europe General Secretary.

For details of advertisement rates please contact the Ada-Europe General Secretary (contact details above).

ADA USER JOURNAL

Volume 43
Number 1
March 2022

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

After two years into the global COVID-19 pandemic, my first wish and hope for 2022, as Editor-in-Chief of the AUJ, is that we get back to track concerning the timeliness of publication and delivery of new issues, for which it will be important that the flow of materials, namely from Ada-related events, returns to what it was before.

For this first AUJ issue in 2022, we challenged the presenters at the "Ada DevRoom" at FOSDEM, which took place in February 2022 as an online event, to prepare short papers derived from their presentations for inclusion in what we called "Proceedings of the 11th Ada Developer Room at FOSDEM 22". This year the response was overwhelming, with a total of 11 contributions being received. We decided to make it easier for authors, not imposing restrictions in terms of paper size, so we ended up with contributions ranging from 1 to 8 pages, as the reader will find. The topics are diverse, but in all cases, strongly related to the Ada language and Ada tools.

These informal proceedings start with an introduction prepared by Dirk Craeynest, one of the Ada DevRoom organizers, who describes the scope of the event and provides a program overview. Then we include the 11 received contributions. The reader will find papers talking about the personal experience of newcomers to Ada programming (Ada Looks Good, Now Program a Game Without Knowing Anything; The Outsider's Guide to Ada: Lessons from Learning Ada in 2021), papers related to the use and exploitation of Ada language features (The Ada Numerics Model, Use (and Abuse?) of Ada 2022 Features to Design a JSON-Like Data Structure), to the management and development of Ada projects (Alire Update; SweetAda: Lightweight Development Framework for Ada-Based Software Systems; Implementing a Build Manager in Ada; Exporting Ada Software to Python and Julia), to the use of specific Ada packages (Overview of Ada GUI), to both the development of Ada projects and the use of specific packages (Getting Started with AdaWebPack), and to the use of SPARK to prove the correctness of Ada libraries (Proving the Correctness of the GNAT Light Runtime Library).

We hope the reader will enjoy reading these contributions, and perhaps will be tempted to watch the presentations that were recorded during the event, available on its webpage (https://fosdem.org/2022/schedule/track/ada/).

As usual, this issue also includes the News Digest section prepared by Alejandro R. Mosteo and the Calendar and Events section prepared by Dirk Craeynest. We note that one of the forthcoming events will be the 2022 edition of the HILT (High Integrity Language Technologies) Workshop, taking place in October, which this year is organized by ACM SIGAda in cooperation with Ada-Europe. The respective Call for Papers is included in the Forthcoming Events section.

Antonio Casimiro
Lisboa
March 2022
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Alejandro R. Mosteo

Centro Universitario de la Defensa de Zaragoza, 50090, Zaragoza, Spain; Instituto de Investigación en Ingeniería de Aragón, Mariano Esquillor s/n, 50018, Zaragoza, Spain; email: amosteo@unizar.es

Contents

[Messages without subject/newsgroups are replies from the same thread. Messages may have been edited for minor proofreading fixes. Quotations are trimmed where deemed too broad. Sender's signatures are omitted as a general rule. —arm]

Preface by the News Editor

Dear Reader,

As I write this preface, we are in between two big Ada events: the FOSDEM Ada Developer Room, which brings together open source enthusiasts presenting their latest developments [1], and the Ada-Europe Int. Conf. on Reliable Software Technologies (AEiC 2022), which this year will return, fingers crossed, as an in-person event at Ghent, Belgium [2]. Information about both can be found in the "Ada-related Events" section.

A piece of news that has made some ripples in the Ada community is the recently announced collaboration between AdaCore and Ferrous Systems to provide a safety-qualified Rust toolchain. The newsgroup saw some reactions to this announcement [3], and a discussion about the merits, similarities and differences between Rust and Ada and their respective strong points.

Glad tidings come for macOS users with the announcement of builds of GCC 12 and SPARK 2014 for this operating system, thanks to the volunteer efforts of Simon Wright [4], with GCC also available for the M1 architecture. And for the lovers of space, some of us wonder whether there is some Ada in the Webb telescope [5]. (Spoiler: probably not.)

Sincerely,
Alejandro R. Mosteo.

[1] "Ada Developer Room at FOSDEM 2022", in Ada-related Events.

[2] "CfC Ada-Europe 2022 Conference", in Ada-related Events.

[3] "AdaCore Joins with Ferrous Systems to Support Rust", in Ada and Other Languages.

[4] "macOS GCC 12.0.1, SPARK2014", in Ada-related Resources.

[5] "Ada in James Webb Space Telescope?", in Ada Inside.

Ada-related Events

CfC Ada-Europe 2022 Conference

[Deadline is past; announcement kept for the record. –arm]

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC Ada-Europe 2022 Conference - 27 Feb - second deadline
Date: Mon, 31 Jan 2022 16:44:08 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

UPDATED Call for Contributions

26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022)

14-17 June 2022, Ghent, Belgium

www.ada-europe.org/conference2022

Organized by Ada-Europe in cooperation with ACM SIGAda, SIGPLAN, SIGBED and the Ada Resource Association (ARA)

* 2nd DEADLINE 27 February 2022 *

#AEiC2022 #AdaEurope #AdaProgramming

General Information

The 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022) will take place in Ghent, Belgium, in the week of 14-17 June, in dual mode, with a solid core of in-presence activities accompanied by digital support for remote participation. The conference schedule comprises a journal track, an industrial track, a work-in-progress track, a vendor exhibition, parallel tutorials, and satellite workshops.

Schedule

16 January 2022 Submission deadline for journal-track papers, tutorials and workshop proposals.

27 February 2022: Submission deadline for industrial-track and work-in-progress-track abstracts.

14 March 2022 Notification of invitations-to-present for journal-track papers. Notification of acceptance for all other types of submission.

3 April 2022: Publication of advance program.

Topics

The conference is an established international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of developing, running and maintaining challenging long-lived, high-quality software systems for a variety of application domains including manufacturing, robotics, avionics, space, health care, transportation, cloud environments, smart energy, serious games. 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:

- Real-Time and Safety-Critical Systems: design, implementation and verification challenges, novel approaches, e.g., Mixed-Criticality Systems, novel scheduling algorithms, novel design and analysis methods;
- High-Integrity Systems and Reliability: theory and practice of High-Integrity Systems, languages vulnerabilities and countermeasures, architecture-centred development methods and tools;
- Reliability-oriented Programming Languages (not limited to Ada): compilation and runtime challenges, language profiles, use cases and experience reports, language education and training initiatives;

- Experience Reports: case studies, lessons learned, and comparative assessments.

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

Call for Journal-track Submissions

Following the journal-first model inaugurated in 2019, the conference includes a journal-track that seeks original and high-quality submissions that describe mature research work in the scope of the conference. Accepted papers for this track will be published in the "Reliable Software Technologies (AEiC2022)"

[Submission details removed. Call is closed now.]

Authors who have successfully passed the first round of review will be invited to present their work at the conference. Ada-Europe, the main conference sponsor, will cover the Open Access fees for the first four papers to gain final acceptance, which do not already enjoy OA from personalized bilateral agreements with the Publisher.

Call for Industrial-track Submissions

The conference seeks industrial practitioner presentations that deliver insight on the challenges of developing reliable software. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer review process. Interested authors shall submit a short (one-to-two pages) abstract, by 27 February 2022, via https://easychair.org/conferences/?conf=aeic2022, strictly in PDF, following the Ada User Journal style (cf. http://www.ada-europe.org/auj/).

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference, and will also be invited to expand their contributions into full-fledged articles for publication in the Ada User Journal, which will form the proceedings of the Industrial track of the Conference.

Prospective authors may direct all enquiries regarding this track to the corresponding chair, Alejandro R. Mosteo, at the listed address.

Call for Work-in-Progress-track Submissions

The Work-in-Progress track seeks two kinds of submissions: (a) ongoing research, and (b) early-stage ideas. Ongoing research submissions are 4-page papers that describe research results that are not mature enough to be submitted to the journal track as yet. Early-stage ideas, are 1-page papers that pitch new research directions that fall in the scope of the conference. Both kinds of submission must be original and shall undergo anonymous peer review. Submissions by recent MSc graduates and PhD students are especially sought.

[Submission details removed. Call is closed now.]

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference, and will also be offered the opportunity to expand their contributions into 4-page articles for publication in the Ada User Journal, which will form the proceedings of the WiP track of the Conference.

Academic Listing

The Journal of Systems Architecture, publication venue of the journal-track proceedings of the conference, was ranked Q1 (SJR) in the year 2020, also featuring 72th percentile in CiteScope (Scopus). The Ada User Journal, venue of all other technical proceedings of the conference, is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Awards

Ada-Europe will offer an honorary award for the best technical presentation, to be announced in the closing session of the conference.

Call for Tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic or practitioner slant, including hands-on or practical elements.

[Submission details removed. Call is closed now.]

The authors of accepted full-day tutorials will receive a complimentary conference registration, halved for half-day tutorials. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

The conference welcomes satellite workshops centred on themes that fall within the conference scope. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference proper.

[Submission details removed. Call is closed now.]

Call for Exhibitors

The conference will include a vendor and technology exhibition. Interested providers should direct inquiries to the Exhibition Chair.

Venue

The conference will take place in the heart of the city of Ghent, Belgium, capital of the East Flanders province, a halfhour train ride north-west of Brussels. Ghent is rich in history, culture and higher-education, with a top-100 university founded in 1817.

Organizing Committee

* Conference Chair
Tullio Vardanega, University of Padua, Italy
tullio.vardanega at unipd.it

* Journal-track Chair
Jérôme Hugues, Carnegie Mellon University, USA
jjhugues at sei.cmu.edu

* Industrial-track Chair
Alejandro R.Mosteo, Centro Universitario de la Defensa, Zaragoza, Spain
amosteo at unizar.es

* Work-in-Progress-track Chair
Frank Singhoff, University of Brest, France
frank.singhoff at univ-brest.fr

* Tutorial and Workshop Chair
Aurora Agar Armario, NATO, the Netherlands
aurora.agar at ncia.nato.int

* Exhibition & Sponsorship Chair
Ahlan Marriott, White Elephant GmbH, Switzerland
software at white-elephant.ch

* Publicity Chair
Dirk Craeynest, Ada-Belgium & KU Leuven, Belgium
dirk.craeynest at cs.kuleuven.be

* Local Chair
Vicky Wandels, University of Ghent, Belgium
Vicky.Wandels at UGent.be

*** Previous Editions

Ada-Europe organizes annual international conferences since the early 80's. This is the 26th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain ('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04), York, UK ('05), Porto, Portugal ('06), Geneva, Switzerland ('07), Venice, Italy ('08), Brest, France ('09), Valencia, Spain ('10), Edinburgh, UK ('11), Stockholm, Sweden ('12), Berlin, Germany ('13), Paris, France ('14), Madrid, Spain ('15), Pisa, Italy ('16), Vienna, Austria ('17), Lisbon, Portugal ('18), Warsaw, Poland ('19), and online from Santander, Spain ('21).

Information on previous editions of the conference can be found at http://www.ada-europe.org/confs/ae.

Our apologies if you receive multiple copies of this announcement.

Please circulate widely.

Dirk Craeynest, AEiC 2022 Publicity Chair
Dirk.Craeynest@cs.kuleuven.be

* 26th Ada-Europe Int.Conf. Reliable Software Technologies (AEiC 2022) * June 14-17, 2022, Ghent, Belgium * www.ada-europe.org/conference2022

Ada Developer Room at FOSDEM 2022

[Past event, for the record. –arm]

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada Developer Room at FOSDEM 2022 - Sun 6 Feb - online
Date: Thu, 3 Feb 2022 20:14:24 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Call for Participation

11th Ada Developer Room at FOSDEM 2022

Sunday 6 February 2022, online from Brussels, Belgium

Organized in cooperation with Ada-Belgium [1] and Ada-Europe [2]

fosdem.org/2022/schedule/track/ada/

www.cs.kuleuven.be/~dirk/ada-belgium/events/22/220206-fosdem.html

#AdaFOSDEM #AdaDevRoom #AdaProgramming #AdaBelgium #AdaEurope #FOSDEM2022

FOSDEM [3], the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organized early each year in Brussels, Belgium. It is highly developer-oriented and brings together 8000+ participants from all over the world. The 2022 edition takes place on Saturday 5 and Sunday 6 February. It is free to attend and no registration is necessary. This year, for obvious reasons, it has been turned into an online event, just like last year.

In this edition, the Ada FOSDEM community organizes once more 8 hours of presentations related to Ada and Free or Open Software in a s.c. Developer Room. The "Ada DevRoom" at FOSDEM 2022 is held on the 2nd day of the event, and offers introductory presentations on the Ada programming language, as well as more specialised presentations on focused topics, tools and projects: a total of 13 Ada-related presentations by 12 authors from 8 countries!

Program overview:

- Introduction to the Ada DevRoom, by Fernando Oleo Blanco, Germany
- Introduction to Ada for Beginning and Experienced Programmers, by Jean-Pierre Rosen, France
- Ada Looks Good, Now Program a Game Without Knowing Anything, by Stefan Hild, Germany
- The Ada Numerics Model, by Jean-Pierre Rosen, France
- 2022 Alire Update, by Fabien Chouteau, France, Alejandro Mosteo, Spain
- SweetAda: Lightweight Development Framework for Ada-based Software Systems, by Gabriele Galeotti, Italy
- Use (and Abuse?) of Ada 2022 Features to Design a JSON-like Data Structure, by Alejandro Mosteo, Spain
- Getting Started with AdaWebPack, by Max Reznik, Ukraine
- Overview of Ada GUI, by Jeffrey Carter, Belgium
- SPARKNaCl: a Verified, Fast Re-implementation of TweetNaCl, by Roderick Chapman, UK
- The Outsider's Guide to Ada: Lessons from Learning Ada in 2021, by Paul Jarrett, USA
- Proving the Correctness of the GNAT Light Runtime Library, by Yannick Moy, France
- Implementing a Build Manager in Ada, by Stephane Carrez, France
- Exporting Ada Software to Python and Julia, by Jan Verschelde, USA
- Closing of the Ada DevRoom, by Dirk Craeynest, Belgium, Fernando Oleo Blanco, Germany

The Ada at FOSDEM 2022 web-page will have all details, such as the full schedule, abstracts of presentations, biographies of speakers, and pointers to more info, including live video streaming and chat, plus recordings afterwards. For the latest information at any time, contact Fernando Oleo Blanco <irvise@irvise.xyz>, or see:

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

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

[3] https://fosdem.org/2022/

Dirk Craeynest, FOSDEM Ada DevRoom team
Dirk.Craeynest@cs.kuleuven.be (for Ada-Belgium/Ada-Europe/SIGAda/WG9)

Ada Developer Room Videos Online

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada Developer Room at FOSDEM 2022 - videos online
Date: Sun, 20 Feb 2022 14:23:10 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

** Presentations and video recordings available online ***

11th Ada Developer Room at FOSDEM 2022

held on Sunday 6 February 2022, online from Brussels, Belgium

https://fosdem.org/2022/schedule/track/ada/

All presentations and video recordings from the 11th Ada Developer Room, held at the online FOSDEM 2022 event recently, are available.

Yet another full day with 13 Ada-related talks by 12 authors from 8 countries!

[See program overview in the previous message. –arm]

Thanks once more to all presenters and helpers for their work and collaboration, thanks to Fer for coordinating the DevRoom, thanks to all the FOSDEM organizers and volunteers, thanks to the many participants for their interest, and thanks to everyone for another nice experience!

#AdaFOSDEM #AdaDevRoom #AdaProgramming #AdaBelgium #AdaEurope #FOSDEM2022

Dirk Craeynest, FOSDEM Ada DevRoom team

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

Ada-related Resources

[Delta counts are from Nov 1st to May 9th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: Mon, 2 May 2022 11:39:21 CET
To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 3_302 (+88) members [1]
- Reddit: 8_005 (+357) members [2]
- Stack Overflow: 2_212 (+87) questions [3]

- Libera.Chat:75 (=) concurrent users [4]
- Gitter:115 (+24) people [5]
- Telegram: 139 (+9) users [6]
- Twitter: 30 (-197) tweeters [7]
  53 (-223) unique tweets [7]

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

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

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

[4] https://netsplit.de/channels/details.php?room=%23ada&net=Libera.Chat

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

[6] https://t.me/ada_lang

[7] http://bit.ly/adalang-twitter

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: Mon, 9 May 2021 11:45:21 CET
To: Ada User Journal readership

Rosetta Code: 900 (+54) examples [1]
39 (+1) developers [2]

GitHub: 763* (=) developers [3]

Sourceforge: 274 (+1) projects [4]

Open Hub: 214 (=) projects [5]

Alire: 243 (+48) crates [6]

Bitbucket: 88 (=) repositories [7]

Codelabs: 53 (=) repositories [8]

AdaForge: 8 (=) repositories [9]

*This number is unreliable due to GitHub search limitations.

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

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

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

[4] https://sourceforge.net/directory/language:ada/

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

[6] https://alire.ada.dev/crates.html

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

[8] https://git.codelabs.ch/?a=project_index

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

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: Mon, 9 May 2021 11:50:21 +0100
To: Ada User Journal readership

[Positive ranking changes mean to go up in the ranking. —arm]

- TIOBE Index: 27 (+4) 0.46% (+0.04%) [1]
- PYPL Index: 17 (=) 0.81% (-0.13%) [2]
- IEEE Spectrum (general): 31 (=) Score: 38.8 (=) [3]
- IEEE Spectrum (embedded): 9 (=) Score: 38.8 (=) [3]

[1] https://www.tiobe.com/tiobe-index/

[2] http://pypl.github.io/PYPL.html

[3] https://spectrum.ieee.org/top-programming-languages/

Ada "Coin" Updated for Ada 2022

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada - In Strong Typing We Trust - "coin" updated for Ada 2022
Date: Sat, 5 Feb 2022 16:04:59 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Ada - In Strong Typing We Trust - "coin" updated for Ada 2022

As of today, a new version of the traditional "Ada coin" is available for promotional use at http://www.cs.kuleuven.be/~dirk/ada-belgium/pictures/ada-strong.html

Coinciding with the final stages in the ISO standardization of the latest Ada programming language revision, referred to as "Ada 2022", and for the occasion of the 11th Ada Developer Room at FOSDEM 2022, a new update was made available, adding "2022".

Enjoy!

Dirk Craeynest

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

New Ada Forge

From: William <william@sterna.io>
Subject: New Ada Forge: catalog of (almost) all Ada open source code & tools
Date: Sat, 12 Feb 2022 19:47:36 +0100
Newsgroups: comp.lang.ada

Hello Ada lovers!

I've the pleasure to announce a ground-up update of AdaForge.org

https://www.adaforge.org

The purpose of this site is to bring to the Ada developer a catalog of (almost) all Ada open source code and tools existing in different public repositories.

==> This catalog is structured according to a software developer perspective (taxonomy).

Note: AdaForge.org references 100% of the Alire 'crates' packaging repo. :-)

I'm excited to hear some feedback from you,

Kind regards,
William

Ada-related Tools

AdaStudio-2021 Release 01/10/2021 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce: AdaStudio-2021 release 01/10/2021 free edition
Date: Fri, 1 Oct 2021 22:31:52 -0700
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2021 new release, based on Qt-6.2.0-everywhere Qt 6.2.0 opensourc without qtwebengine,extended with modules from Qt-5.15: qtgraphicaleffect qtlocatio qtgamepad qtspeech qtx11extras qtwinextras Qt 6 is a new long time project and I hope to add qtwebengine in next releases.

Qt6ada version 6.2.0 open source and qt6base.dll ,qt6ext.dll (win64),libqt6base.so,libqt6txt.so(x86-64) built with Microsoft Visual Studio 2019 x64 Windows, gcc x86-64 in Linux.

Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit, Linux x86-64 Debian 10.4 Qt-6.2.0 everywhere opensource prebuilt binaries for win64 and amd64 are included into AdaStudio-2021

AdaStudio-2021 includes the following modules: qt6ada, vtkada, qt6avada, qt6cvada and voice recognizer.

Qt6Ada is built under GNU GPLv3 license https://www.gnu.org/licenses/lgpl-3.0.html.

Qt6Ada modules for Windows, Linux (Unix) are available from Google drive https://drive.google.com/folderview?id=0B2QuZLoe-yiPbmNQRl83M1dTRVE&usp=sharing

[List of detailed file contents omitted. —arm]

The full list of released classes is in "Qt6 classes to Qt6Ada packages relation table.docx"

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Sat, 2 Oct 2021 16:00:50 +0200

Hi Leonid,

I have been following your work for a few years. I like the Qt ecosystem (even with their change of heart) and very specially VTK. Thank you for your work. I hope to use it in the future for my projects.

I first wanted to say that the webpage that is indicated on your CV and where QtAda has been living is unreachable. Google says it has been blocked since it is suspicious. Do you receive the same message? [https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/adastudio.html]

> Qt6Ada is built under GNU GPLv3 license https://www.gnu.org/licenses/lgpl-3.0.html.

Is it GPLv3 or LGPLv3? I am asking since you mention GPLv3 but link LGPLv3.

Once again, thank you for maintaining this lovely software suite!

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Thu, 7 Oct 2021 22:59:20 -0700

Qt6Ada is built under GNU LGPLv3 license, sorry for my mistake.

I built a web page from my google drive and it worked well, but now I have got a message from Google and I don't know why. Old link to AdaStudio no longer works. The new is https://drive.google.com/drive/folders/0B2QuZLoe-yiPbmNQRl83M1dTRVE?resourcekey=0-b-M35gZhynB6-LOQww33Tg&usp=sharing

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 6 Oct 2021 23:28:33 +0200

I have been playing today with qt5ada and I think I can shed some light on the issue [of some seemingly missing C files —arm].

The Ada sources that call the Qt procedures are in the AdaStudio/qt5ada/qt5adasrc.tar.bz2 file. That is the source file. There are no C files there. That library contains all of the Ada wrapper.

However, that is indeed not enough to use it. It requires a fully functional Qt5 installation (and a very complete one, with bells and whistles). The binaries are provided in the other *.tar.bz2 files (except the demos file). There is also the qt5adax86-64.tar.bz2 file which weighs about 6Mb. That seems to be the relevant file to build qt5ada from source in Linux.

It comes with different files to setup the file structure and environment. I must admit, I have not tried to build it with the provided files in qt5adax86-64.tar.bz2

These "build files" expect you to have a Qt5 installation in your local /usr/local folder. I suppose that is where the qt5.15x86-64.tar.bz2 comes into place, after all, it should unpack in the directory written in the environment file.

Of course, the question is: where are the instructions to build this all from source? The short answer is in the document "How to use Qt5Ada.docx" that is present in AdaStudio/qt5ada. That sheds more light into the procedure. But it still expects you to use the precompiled Qt5 binary. And, I must be honest, it is not clear and easy to follow, you need to adapt the generic instructions to what is on your system...

Then the question becomes: "How can I build _everything_ from source? Specially with the system provided libraries, such as the system provided Qt5." Well... That is not so simple. I understand why Leonid has set up things this way. Correctly setting the compiler flags and directories for system installed libraries is a nightmare. I tried to compile qt5ada with my system provided Qt5 (OpenSUSE Tumbleweed), it is not trivial _at all_. There can be problems with the Qt5 version, there can be problems with the plugins, compiler flags, etc. Can it be done? Most likely, but it will require some elbow grease. There is a reason to why most Qt projects use CMAKE to build and link themselves; because it is not an easy task.

So I would say that the instructions need to be cleaner and that in its current state, there is only one easy solution to building qt5ada, and it requires the binaries provided. But I would also say that all the source files needed are in there. The prebuilt Qt5 binary seems to be the standard unmodified Qt5 distribution, so no surprises there. And that a lot of extra work would be needed to make qt5ada work seamlessly with the system provided libraries.

SweetAda 0.9

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: ANN: SweetAda 0.9 released
Date: Sun, 3 Oct 2021 06:32:40 -0700
Newsgroups: comp.lang.ada

I've just released SweetAda 0.9.

SweetAda is a lightweight development framework to create Ada systems on a wide range of machines. Please refer to https://www.sweetada.org.

Release notes @ https://www.sweetada.org/release_notes.html (delayed)

Downloads available @ https://sourceforge.net/projects/sweetada.

Clone repository @ https://github.com/gabriele-galeotti/SweetAda

Release notes

There are too many changes, so I will list only the most important features of this release.

- Windows environment does not need the grep utility, nor a dos2unix utility (which is now provided internally); elftool is now optionals and its use is configurable in configuration.in
- RTS can be build from sources by means of "CPU=<cpu> make rts" command (the RTS type is being picked up from configuration.in as usual), every RTS branch will be named like the toolchain triplet being used
- Both SweetAda and RTS are fully buildable in Linux, Windows/cmd.exe, Windows/MSYS and OS X; you should only to have online a "make" and "sed" (and for Windows these are available as zip packages in Sourceforge); due to this, there are no RTS packages anymore
- SweetAda does not relies on SweetAda toolchains, you can use your own GNU toolchain, or whatever GNAT you can pick, just be sure to use Ada 2020

The final result is a package that is fully auto-consistent, because the core, RTS and utilities are fully provided in both source form and executable form. Since SweetAda toolchains are by no means eligible as the unique compilers for the system, they will slowly fade away.

GCC Release Notes, aka, Ada Is Still Alive!

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: GCC release notes, aka, Ada is still alive!
Date: Mon, 11 Oct 2021 20:41:18 +0200
Newsgroups: comp.lang.ada

Hi everybody,

I have been meaning to write this message for a long while, so here it goes.

Reading Phoronix [1] for years, I noticed that with every new GCC release, the biggest changes to GCC and its languages were mentioned. However, Ada was pretty much never present.

Today, just a few moments ago in #netbsd, someone asked whether Ada had finally been dropped out of GCC... I am not even mad. GCC's release notes have not mentioned Ada since GCC 8 [2], [3], [4]; and even in GCC 7 and 8 the notes are minute.

So I would like to ask whether someone would like to help me get release notes ready. I am not saying that I will be doing

much, but I would like to breathe some fresh air into how Ada is seen and how much people hear about it.

I personally do not like marketing since good products stand on their merits, not slogans or shininess. But there is no reason to not put publicly what is going on.

Yes, AdaCore has been doing some very nice followups to the development of Ada in their blog [5]. But the people that go there are already aware of Ada. And since AdaCore is phasing out their GNAT CE system in favour of FSF builds (included in Alire), the relevance of GCC's releases grows.

Note, I am not implying that AdaCore should write the releases. They are doing the bulk of work in GNAT, so I do not think they _need_ to do more. Personally I am glad with what they are doing, but of course, they can write the releases if they so want.

I am especially saddened by the fact that GCC has gotten a substantial amount of support for Ada 2022 and it is not even mentioned. No wonder why people think Ada is dead!

So, if you have any recommendation, or would like to help, then you are more than welcomed!

P.S.: I am already doing my part GNAT in NetBSD x86_64 is working! It has 9 failed ACATS tests, but they are minor. A thousand thanks go to J. Marino and Tobiasu for their enormous help in #ada. Today I will see if I can compile it for armv6 and run it on my RPi!

[1] https://www.phoronix.com/scan.php?page=home

[2] https://gcc.gnu.org/gcc-7/changes.html

[3] https://gcc.gnu.org/gcc-8/changes.html

[4] https://gcc.gnu.org/gcc-9/changes.html

[5] https://blog.adacore.com/

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Tue, 12 Oct 2021 05:54:33 -0700

Most, if not all, of what is in this blog post [1] is applicable to GNAT/GCC 11.

[1] https://blog.adacore.com/ada-202x-support-in-gnat

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue, 12 Oct 2021 21:37:25 +0300

> Most, if not all, of what is in this blog post [1] is applicable to GNAT/GCC 11.

I guess the main point of Fernando was that it would be nice if someone could add all the new changes between versions 11 and 12 to https://gcc.gnu.org/gcc-12/changes.html before GCC 12 is released.

gcc-X/changes.html traditionally lists some items for all other language frontends, but there is never anything for Ada.

The git history for gcc-12/changes.html page is visible at

https://gcc.gnu.org/git/?p=gcc-wwwdocs.git;a=history;f=htdocs/gcc-12/changes.html;h=f38fd2bef9c4089369e6f9315590ebffd8b24f5c;hb=HEAD

(that is gcc-wwwdocs repository at gcc.gnu.org/git).

Maybe someone with enough free time (and enough knowledge about the changes) could take look and provide a patch for GCC web page maintainers?

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 13 Oct 2021 18:32:30 +0200

Thank you to everybody that commented on the topic.

We could use the Changelog present in the gcc/ada directory to triage commits more precisely (credit goes to Stéphane).

> I decided to try an example. I must confess that I don't know where the cutoff point for GCC 11 was and what it changes actually did

To be honest, we could try to write the changelog for GCC 11 with the information given by Fabien (AdaCore) and what we find out. If for whatever reason the GCC people do not want to make large changes to the already released changelog, we could compile a larger list for GCC 12.

I think the most important aspects are:

- Ada 2022, which has a long list of changes on its own;
- Improvements to systems (VxWorks, RTMS, etc), as it shows that Ada is present in more places than what meets the eye;
- Deprecations and fixes;
- General improvements in the library, SPARK and with the GCC ecosystem.

I think Ada has somewhat acceptable support for OpenMP, which was improved in the past few years, for example. It has also been increasing SPARK support in the libraries.

[...]

I want to sign up for GCC's gcc mailing list (general discussion) and ask the GCC people what would be the preferred way to move forward. Hey, maybe they would like to have Ada changelogs for all past releases! If I hear anything back I will tell you.

Though if someone wants to start, I see no problem on sharing diffs here. Not the most ideal place, but it is a good forum to share ideas.

From: Simon Wright <simon@pushface.org>
Date: Wed, 13 Oct 2021 20:59:47 +0100

> - General improvements in the library, SPARK and with the GCC ecosystem.

Not sure how to work SPARK into a GCC note, since it's not part of the GCC ecosystem?

"There's extensive support for possible static analysis of code, e.g. via SPARK, in the form of annotations that can optionally be compiled as runtime assertions."

From: Stéphane Rivière <stef@genesix.org>
Date: Thu, 14 Oct 2021 10:24:20 +0200

> So, if you have any recommendation, or would like to help, then you are more than welcomed!

I second that and I would like to help, if I may.

According to gcc-mirror on github, Ada basecode is above C++

C 47.7%
Ada 17.5%
C++ 14.9%
Go 7.4%
GCC Machine Description 4.7%
Fortran 2.4%
Other 5.4%

git clone https://github.com/gcc-mirror/gcc
git log > log.gcc (volume: 124M)
cat log.gcc | grep AdaCore > log.ada (1M, ~25K contribs since 2005)
grep "\[Ada\]" ./log.gcc > log-oneliner.ada (190K, 3200 lines)
grep -B 2 -A 20 AdaCore log.gcc > log-detail.ada
cat log-detail.ada | grep -B 2 -A 20 [ada] log.gcc > log-changes.ada

It seems that everything is there to create a more or less relevant changelog.

But AdaCore's comments are one thing, sorted and relevant information for developers are another.

A raw copy/paste would be useless, we would have to analyze the changelog to give back useful information.

We should also edit a changelog for each GCC release. The above metrics were made on master.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 20 Oct 2021 10:42:20 +0200

The discussion thread on the GCC ML has been started. You can find it here: https://gcc.gnu.org/pipermail/gcc/2021-October/237600.html

Do not hesitate to add any comments!

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 20 Oct 2021 22:14:57 +0200

> The discussion thread on the GCC ML has been started.

Okay, we already had a couple of comments and they cover everything needed. Arnaud has volunteered to be the "supervisor". So here is my plan: crowdsourcing! :D

I would like to write a (simple) list of changes for each version here, on the CLA. If you want to add something __copy__ (do not quote) the list from the previous person/reply/modification and add your proposed changes. You can also make comments if you would like anything changed. If "CHECK" or if "TODO" are written by somebody, it means that something needs to be checked or that it needs to be expanded; respectively. After the list is mostly completed, we could create a patch(es) to send to GCC. The quality of this list is not going to be great, treat it like a checklist. Obviously, if you want to discuss something about the changes, do quote the relevant section.

[...]

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 25 Oct 2021 20:47:05 +0200

Diff: add to GCC 12 the deletion of gnatxref and gnatfind (the patch was posted today in the ML). The -gnat2020 has been commented too in GCC 10 and -gnat2022 in GCC 12. Also, we have explicit permission by Arnaud to copy as much code as necessary from AdaCore's blog.

LIST OF CHANGES

GCC 12

- Introduction of the -gnat2022 flag in gnatmake (-gnat2020 is a deprecated alias).
- gnatfind and gnatxref tools have been deleted. They have been deprecated for years and have been substituted by gprbuild tools.
- Further library improvements in both quality and performance.
- The use of contracts has been extended in the "Ada library" allowing for further checks at runtime or a deeper static analysis with the SPARK prover.
- Further improvements to embedded systems such as VxWorks and RTMS. CHECK maybe be more specific/generic.

GCC 11

- Better Ada 2022 support. The parallel keyword is still unsupported.
- TODO name the additional features. See [1], obviously, with some code examples.
- Addition of the Jorvik profile. CHECK, see [2], maybe code examples?
- Additional non-standard features [3]. CHECK if this applies to GCC 11 or 12.
- A bug was fixed were previous GCC versions allowed XXX construct CHECK. This is not allowed by the standard. Some software was making use of XXX (which is, once again, not allowed) and it has to be patched.
- General library improvements in both clarity and performance.
- The use of contracts has been extended in the "Ada library" allowing for further checks at runtime or a deeper static analysis with the SPARK prover.
- Further improvements to embedded systems such as VxWorks and RTMS. CHECK maybe be more specific/generic.

GCC 10

- Introduction of the -gnat2020 flag in gnatmake (-gnat2020 is a deprecated alias). It enables newer features present in Ada 2022 (still to be ratified). These features are still experimental.
- Some Ada 2022 features are available already with the use of the -gnatX (gnat eXtensions switch).

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 27 Oct 2021 09:52:26 -0700

> - gnatfind and gnatxref tools have been deleted. They have been deprecated for years and have been substituted by gprbuild tools.

What "gprbuild tool" replaces gnatxref?

From recent discussions in an AdaCore ticket, the replacement for gnatxref is libadalang, either via the LSP Ada Language Server, or a similar custom wrapper.

GCC Updated in NetBSD!

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: GCC updated in NetBSD!
Date: Tue, 19 Oct 2021 23:47:36 +0200
Newsgroups: comp.lang.ada

Hello everybody! I bring good news!

GCC with Ada support has been updated in NetBSD! Now versions 10 and 11 should work on x86 and x86_64 NetBSD machines! You can find them in pkgsrc-wip (gcc10-aux) [1] and Ravenports (gcc11) [http://www.ravenports.com/]!

First things first, the acknowledgements: a big thank you goes to J. Marino who did the original gcc-aux packages and who provided most if not all the work when it came to fixing the threads and symbols. Another big thank you goes to tobiasu who correctly picked up that the pthread structure wrappers were not correct and had to be remade. Another big thank you goes to Jay Patelani for his help with pkgsrc.

So, long story short. Most of the work that had been done up until a few weeks ago was done correctly, but the failing tests (most related to tasking) were failing in very strange ways. It happened that the pthread structure memory that the Ada wrapper was using was incorrect, so we were getting completely erratic behaviour. Once that got fixed, pretty much all tests passed. J. Marino also took the time and effort to create __gnat_* function wrappers to all the symbols that the NetBSD people have renamed. This is a much cleaner fix and allows for the renamed functions to generate the correct symbols since now they are getting preprocessed. It should also be more "upstream friendly". The issue, however, remains if NetBSD decides to rename more functions that are still being linked directly.

There are still some failing ACATS tests (about 10). Some are related to numerical precision and a couple others. They are mostly the same failing tests in both GCC 10 and 11. J. Marino ran the ACATS tests on a DragonflyBSD (or was it FreeBSD?) machine and the same tests were failing there too. So we suspect is is a common limitation on *BSDs and it is unlikely that this will ever affect anybody. There is also the issue of stack unwinding when it contains a signal trampoline [2], read the following thread to gain more information about this.

[1] https://github.com/NetBSD/pkgsrc-wip/tree/master/gcc10-aux

[2] https://mail-index.netbsd.org/tech-kern/2021/10/15/msg027703.html

I have started trying to get GCC to xcompile to arm* on NetBSD. I think I am somewhat close, but further hacking on NetBSD's src is needed (and I think the RTS is not getting picked up correctly). So do not get your hopes up. I mean, I have a working gcc x86_64 NetBSD host to NetBSD arm* xcompiler, it is the native gcc on arm* that is not getting built correctly.

From: Richard Iswara <haujekchifan@gmail.com>
Date: Wed, 20 Oct 2021 12:01:40 +0700

A big applause for your hard work identifying the problem in the first place.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Tue, 19 Oct 2021 23:43:23 -0700

When I was working at AdaCore, we used to run our internal CRM and the ticket-management tool that processes all email

on a FreeBSD machine, because the sysadmin was very fond of that system. The CRM was (is?) based on AWS (Ada Web Server), so using tasking pretty heavily. We never had any problem at the time.

I guess AdaCore has given up on FreeBSD, like they have macOS.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 20 Oct 2021 20:44:01 +0200

> I guess AdaCore has given up on FreeBSD, like they have macOS.

Well, GCC officially supports FreeBSD x86* and AFAIK, arm too. Though, AFAIK, the gcc-aux packages from freshports have been left without a maintainer...

And good news everybody! I have managed to get GPRBuild working and Alire too! I even got the GNATColl components built using Alire ^^. Pretty easy if you ask me :P

The mayor issue I am facing now is with make... I tried building AWS with Alire but it could not, since it was using make, which in *BSD world is BSD make, aka, bmake, not GNU make, aka gmake... Anyhow, I am very happy to see so many packages getting built without issues in NetBSD :D

There is the problem where GPRBuild says that the "lib" option is not supported on the OS. I don't think it is suprissing since GPRBuild probably does not know anything about NetBSD.

I am also getting warnings from gnatmake:
/home/fernando/bootstrap_ada/alire/src/alire/alire-toolchains.adb:331:8:

warning: frame size too large for reliable stack checking which probably come from NetBSD having a small stack by default.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Tue, 2 Nov 2021 21:32:56 +0100

A bit of a followup.

The package gcc10-aux has been updated in pkgsrc-wip. I am now the maintainer. As requested by some pkgsrc developer, I have made the package explicitly depend on gcc6-aux. That way, it may be used as a base Ada compiler for all the packages that need Ada (although this is just the first step). I have also rebased it on the new skeleton of gcc10 from pkgsrc-current. Hopefully the review period and inclusion into pkgsrc-current will not take much time.

[...]

From: Fernando Oleo Blanco <irvirse_ml@irvise.xyz>
Date: Thu, 23 Dec 2021 12:52:42 +0100

Well well well...

I come with a Christmas present... Ada running on NetBSD-powerpc! It should run on any powerpc "port", in NetBSD terms also known as evbppc, macppc and amigappc.

It is not perfect, but it is there.

Here are the results from ACATS 4.1X running natively on the macppc port (as created by https://github.com/alarixnia/mkimg-netbsd)

=== acats Summary ===

# of expected passes	2490
# of unexpected failures	62
# of expected failures	1487
# of unresolved testcases	11
# of unsupported tests	116

*** FAILURES: c324006 c350a01 c452003 c452005 c452006 c452a02 c52103x c52104x c52104y c552a01 c552a02 c611a04 c650b04 c760a02 c96001a c96008a c96008b cb1010a cb1010c cb1010d cc40001 cc51007 cdd2b03 cdd2b04 cxa4010 cxa4011 cxa4021 cxa4022 cxa4023 cxa4030 cxa4031 cxa4032 cxa4033 cxa4035 cxaa022 cxab004 cxab005 cxac004 cxag001 cxag003 cxai001 cxai009 cxai010 cxaia01 cxaib05 cxaib06 cxaib08 cxb4002 cxb4005 cxb5002 cxb5003 cxd1003 cxd1004 cxd1005 cxd2002 cxd2003 cxd2004 cxd2006 cxd3001 cxd3002 cxd6001 cxd6002

/home/fernando/ACATS-master/run_all.sh completed at Thu Dec 23 10:13:16 UTC 2021

The compiler is GCC from the NetBSD src tree, which is an older GCC 10 version. Which means (following the results from previous runs) that 28 failures where expected; 6 from shortcomings from NetBSD and the rest from GCC 10 not passing newer tests. That means this system generated at least 34 new failures. This may be for a number of reasons, both related and unrelated to GCC-Ada. Still, I think they are rather good! I believe a lot of cxa failures were due to the system running on low memory. Also, the compiler was built against NetBSD 9.99.92, but the actual host is 9.2, and NetBSD is not backwards compatible; so that may explain other failures.

Just for your own enjoyment, these tests took about 2 days to run, since I am emulating powerpc on a virtualised NetBSD-x86_64 system :P

The reason I tried to run powerpc is because, to put it bluntly, NetBSD has to fix their shit with aarch64 and mips64 and because they do not provide binaries for POWER. NetBSD just works if you use their tooling, but the moment something out of the ordinary of what has to be built, fecal matter impacts the air impeller (credit to a reddit user for that one).

Merry Christmas everybody!
Fer

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Thu, 10 Feb 2022 20:21:53 +0100

One last update on GCC 10 on NetBSD.

As I have already said in other messages, it works great. The package is still under wip since no maintainer has stepped up to take care of the review. I also have not pushed it further.

I would recommend the use of Ravenports, since it has GCC 11, which is newer and works on FreeBSD too.

I have given up on trying to port it to other arches. It should be as simple as adding them to the Makefile.rtl. There is a minor bug on my patchset, the x86 intrinsics are also present on the arm sections, I need to delete that.

The reason for giving up on supporting other arches is mostly due to NetBSD not upstreaming support for those arches. For example, the official binutils does not have support for aarch64-netbsd. It is only present in NetBSD's src. And it only works when used within NetBSD's src. This makes everything more complex than needed and I do not have the will to push through with it.

Regarding the use of other Ada tools in NetBSD. I added support for grpbuild a few months ago, so you should be able to just use it. Notice, when using GCC 10 only V21 of AdaCore tools work. Newer versions (currently v22) need GCC 11. The rest of the tools seem to compile without much fuzz at all. So I say that my work is mostly complete.

I will try to get gcc10-aux pushed to stable however; sometime after March.

For now, I will try to update the Ada changelog in GCC and write an article about Ada-Scheme for the AUJ.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Mon, 14 Mar 2022 22:21:49 +0100

Quick update. The package has now been upstreamed and is now part of the official pkgsrc distribution!

You can find it here:
https://cdn.netbsd.org/pub/pkgsrc/current/pkgsrc/lang/gcc10-aux/index.html

Binaries are still not available since it just got added.

This is a nice conclusion to this journey... But there is something else brewing behind the scenes... AVR support is coming to Alire thanks to Fabien and we are ironing out some of the issues there :D

Simple Components v4.59

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components for Ada v4.59
Date: Sat, 6 Nov 2021 13:04:27 +0100
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.

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

Changes to the previous version:

- The primitive operation Clear was added to GNAT.Sockets.Server.Connect;
- Julia bindings moved to the version 1.6.3;
- The functions Eval_char_array, Get_Safe_Restore, Load_File_String, Set_Safe_Restore were added to Julia bindings;
- Functions To_Julia and Value added to Julia bindings for Ada types Time and Day_Duration;
- To_Julia defined on tuples fixed when types of elements are not directly visible.

Dokan Ada Bindings 2.0

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Dokan Ada bindings 2.0
Date: Sun, 16 Jan 2022 17:07:54 +0100
Newsgroups: comp.lang.ada

Dokan is a user-space file system for Windows 32- and 64-bit. It consists of a driver and a library. The driver routes the I/O requests to the file system device to the library callback.

A sample implementing a memory resident files system is provided.

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

Changes to the version 1.5.0:

- The code and the API were reworked to accommodate the Dokan major version 2.

AdaControl 1.22r16c

From: J-P. Rosen <rosen@adalog.fr>
Subject: [Ann] New version of AdaControl
Date: Wed, 8 Dec 2021 17:51:44 +0100
Newsgroups: comp.lang.ada

AdaControl 1.22r16c is mainly a bug fix release (no new rule), but improvements in the static evaluator provides better results and avoids false positives in several rules.

Enjoy!

Renaissance-Ada Made Open Source

From: Pierre Van De Laar <pierre.van.de.laar@gmail.com>
Subject: Renaissance-Ada, a toolset for legacy Ada software, made open source
Date: Thu, 27 Jan 2022 04:32:00 -0800
Newsgroups: comp.lang.ada

Dear Members of comp.lang.ada,

We would like to inform you that we have made Renaissance-Ada, a toolset for legacy Ada software, open source:

https://github.com/TNO/Renaissance-Ada

The Renaissance-Ada project builds on top of LibAdalang and includes the following functionality

* Dependency Graph Extractor that produces a graphml file for visualization and querying with e.g. yEd and Neo4J.
* Rejuvenation Library that allow analysis and manipulation of Ada code based on concrete patterns.
* Rewriters_Library that enables automatic rewriting of Ada code based on concrete patterns.
* Code Reviewer that automatically reviews Ada code based on a large list of rewrite rules.

If you have any question about this toolset don’t hesitate to contact me!

GWindows Release, 29-Jan-2022

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: GWindows release, 29-Jan-2022
Date: Sat, 29 Jan 2022 13:48:46 -0800
Newsgroups: comp.lang.ada

GWindows is a full Microsoft Windows Rapid Application Development framework for programming GUIs (Graphical User Interfaces) with Ada. GWindows works only with the GNAT development system, but with some effort, GWindows could be made pure Ada. GWindows is free and open-source!

Changes to the framework are detailed in gwindows/changes.txt or in the News forum on the project site.

In a nutshell (since last announcement here):

427: GWindows.Image_Lists: added color options; includes features of "extended" Ex_Image_List_Type in package GWindows.Image_Lists.Ex_Image_Lists, which is marked as obsolescent.

424: GWindows.Application: added function Screen_Visibility.

423: GWindows.Application: added Enumerate_Display_Monitors.

422: GWindows.Base: added Set_Foreground_Window.

421: GWindows.Base: added Set_Active_Window.

417: GWindows.Common_Controls.Ex_Tb (toolbar): is now 64-bit compatible; see LEA http://l-e-a.sf.net/ , LEA_GWin.Toolbars for an example.

414: GWindows.Scintilla: method names are "de-camel-cased": e.g.: "Move_Caret_Inside_View" instead of "MoveCaretInsideView".

412: GWindows.Scintilla: works on both Intel x86 32-bit and x64 64-bit types of platforms.

411: GWindows.Common_Controls.Ex_List_View: method On_Free_Payload is now public and can be overridden with effect.

410: GWindows.Common_Controls.Ex_List_View: Sort can use a comparison method not based on strings (e.g. a numerical comparison).

GWindows Project site:

https://sf.net/projects/gnavi/

GWindows GitHub clone:

https://github.com/zertovitch/gwindows

Enjoy!

macOS GCC 12.0.1, SPARK2014

From: Simon Wright <simon@pushface.org>
Subject: ANN: macOS GCC 12.0.1, compatible SPARK2014
Date: Fri, 25 Feb 2022 18:21:04 +0000
Newsgroups: comp.lang.ada

GCC 12.0.1 of 20220204 (only Ada, C, C++, built on El Capitan, runs up to Monterey) available at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.0.1.

SPARK2014 built against it (provers CVC4, Z3, Alt-Ergo; CVC4 requires Sierra and upwards) available at

https://github.com/simonjwright/spark2014/releases/tag/macos-0.1.

Needs GCC 12.0.1 installed. Running the test suite on the ug* tests (the examples in the User Guide) results in one failure (aside from the missing CodePeer one) unless you build with -j2 (where 2 is less than the number of processors in your machine).

A note on building the latter at https://forward-in-code.blogspot.com/2022/02/spark2014-and-fsf-gcc.html.

UXStrings Package Available (UXS_20220226)

From: Blady <p.p11@orange.fr>
Subject: [ANN] UXStrings package available (UXS_20220226).
Date: Tue, 1 Mar 2022 21:47:49 +0100
Newsgroups: comp.lang.ada

The objective of UXStrings is Unicode and dynamic length support for strings in Ada.

UXStrings API is inspired from Ada.Strings.Unbounded in order to minimize adaptation work from existing Ada source codes.

Changes from last publication:

- Ada.Strings.UTF_Encoding. Conversions fix is no longer needed with GNAT CE 2021

- A few fix

Available on GitHub https://github.com/Blady-Com/UXStrings and also on Alire https://alire.ada.dev/crates/uxstrings.html

Feedback is welcome on actual use cases.

GCC 12.0.1/Apple Silicon

From: Simon Wright <simon@pushface.org>
Subject: [ANN] GCC 12.0.1/Apple silicon
Date: Wed, 23 Mar 2022 21:08:25 +0000
Newsgroups: comp.lang.ada

Find GCC 12.0.1 and tools for M1 Macs at https://github.com/simonjwright/distributing-gcc/releases/tag/aarch64-apple-darwin21-1

About double the size of the x86_64 (Intel) equivalent.

Ada Inside

Ada in James Webb Space Telescope?

From: Nasser M. Abbasi <nma@12000.org>
Subject: is Ada used in James Webb Space Telescope software?
Date: Sun, 26 Dec 2021 07:18:41 -0600
Newsgroups: comp.lang.ada

Anyone knows if Ada is used in James Webb Space Telescope software.

Either in the control systems or in the embedded software for the Telescope.

https://www.jwst.nasa.gov/

I sure hope they did not use javascript or Python or C for the software.

There is some talk in the following link about its software but I could not find what language they used.

https://www.nasa.gov/feature/goddard/2020/nasa-s-james-webb-space-telescope-completes-comprehensive-systems-test

From: Peter Chapin <peter@pchapin.org>
Date: Thu, 30 Dec 2021 08:30:54 -0500

> Anyone knows if Ada is used in James Webb Space Telescope software.

It is likely they used C. Specifically, C99. I say this because in my dealings with NASA (related to my work with CubeSats), the people I've talked with made it clear that NASA is now a C shop. Both my colleague and I have extolled the virtues of Ada and SPARK to NASA engineers, but we get the usual reaction: too much investment in C to take any other option seriously... except maybe C++ (JPL, at least, does some work with C++ so that might also be on the JWST).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 26 Dec 2021 15:23:43 +0100

Since it was 30 years in development, I would not dismiss QBasic...

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 26 Dec 2021 11:22:56 -0800

> Since it was 30 years in development, I would not dismiss QBasic...

Don't forget Forth! It was used on many space projects.

https://web.archive.org/web/19990125085748/http://forth.gsfc.nasa.gov/

From: John Mccabe <john@mccabe.org.uk>
Date: Sun, 26 Dec 2021 15:57:42 -0800

> Don't forget Forth! It was used on many space projects.

Interesting. I didn't realise there had been so many projects in Forth. I started to learn/use Forth at one point, as it looked like we (Matra Marconi Space) might be forced to use the RTX2010 as it was one of very few space qualified processors with hardware floating point support. In the end we used the MA31750, with Ada, instead.

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 26 Dec 2021 16:37:00 -0800

> I didn't realise there had been so many projects in Forth.

Much of Forth's early development was at the Kitt Peak observatory where I think Charles Moore worked for a while, so it was popular with the astronomy community and maybe indirectly with the spaceflight community through there and JPL. As a more general matter, hardware designers (electrical engineers who sometimes have to muck with embedded software but aren't really into programming as a topic) tend to like it because of its simplicity and directness.

> In the end we used the MA31750, with Ada, instead.

Interesting. I hadn't heard of the MA31750 but it appears to be a 16 bit processor that implements the MIL-STD-1750A instruction set(!), which I didn't know about either. Apparently it was made in the 1980s but has since been superseded by SPARC architecture cpu's.

I wonder if targeting GCC to the RTX2010 might have been feasible. Can I ask what Ada compiler you used for the MA31750? It looks like GCC supported the MA31750 until version 3.1, but I don't know whether GNAT existed then.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 27 Dec 2021 09:44:26 +0200

>> I didn't realise there had been so many projects in Forth.

Forth is of course one of the few ways to get a self-hosted but fairly fast interactive compiler/editor system on small processors.

In the 1980's I was working in radio astronomy and we were planning to use Forth to replace HP BASIC on an HP2100 16-bit mini for telescope control and data acquisition. I had a little crush on Forth at the time, but fell out of love with it when I found that some astronomy SW had defined the word 2000.0 as a procedure to convert stellar coordinates to the year 2000 ephemeris... very clear :-(

Fortunately IMO we chose to use HP-Algol instead, and much later changed to Ada on a MicroVAX.

> Can I ask what Ada compiler you used for the MA31750?

Like John, I used Ada on an MA31750. We used the TLD Ada compiler, where (IIRC) TLD stands for the main author, Terry L. Dunbar. GNAT was around, but I don't remember if it had support for the MA31750 -- I doubt it. We used gnatp 3.<something> for testing the MA31750 SW on workstations (Sun Solaris on SPARC, IIRC), but the customer (Matra Marconi Space) specified TLD Ada for the target, so there was never a question of using GNAT instead.

That project developed the on-board SW for the ozone-monitoring instrument GOMOS on the ESA ENVISAT satellite.

I believe ENVISAT used MA31750 and TLD Ada for all its systems.

From: John Mccabe
<john@mccabe.org.uk>
Date: Tue, 28 Dec 2021 02:24:54 -0800

> [the MA31750] appears to be a 16 bit processor that implements the MIL-STD-1750A instruction set(!)

There were 3 or 4 different implementations of the MIL-STD-1750A instruction set architecture around the time. It was an interesting one; it was fairly small, but had some relatively complex instructions that were really useful. The MA31750 was GEC-Plessey Semiconductors' 2nd version, I believe, although if I remember correctly, this was the one that had the FPU, or maybe it was the MMU, integrated into a single device, using silicon-on-sapphire for rad-hardness. There were two other implementations I particularly remember that were rad-hard, one by IBM, which had better claimed performance but was really expensive and special order only (I think we paid £7500 or so for each MA31750, so you may be able to imagine what I mean by "really expensive"), and one by another US company that went into Chapter 11 protection around the time we were talking to them!

> Can I ask what Ada compiler you used for the MA31750?

I'm almost 100% sure GNAT wasn't available for the MIL-STD-1750A; it was a very niche market and we weren't aware of any C compilers we could've used at the time, even if we'd wanted to.

The Ada compiler we used was the same as Nikolas; TLD. I was also working on part of ENVISAT (the Tile Control and Interface Unit - TCIU, although some of my colleagues were also using it on the main ASAR control system). Although Nikolas mentions Matra Marconi Space mandating TLD, that would've come down from Dornier who'd apparently done a deal with TLD. I don't know what happened with TLD after that, but some geezer from the Irvine Compiler Corporation contacted me once when they were following up on some unpaid license fees related to part of the TLD compiler.

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Tue, 28 Dec 2021 12:59:32 +0200

> Although Nikolas mentions Matra Marconi Space mandating TLD, that would've come down from Dornier who'd apparently done a deal with TLD.

Yes, a considerable part of our requirements came from Dornier via Matra Marconi Space (France). We sometimes had fun trying to understand how the French had interpreted requirements written in English by the Germans. The two other languages had left their imprints on the "English" wording :-)

From: John Mccabe
<john@mccabe.org.uk>
Date: Fri, 31 Dec 2021 02:26:14 -0800

> Yes, a considerable part of our requirements came from Dornier via Matra Marconi Space (France).

We didn't really have that problem. On TCIU most of our requirements came from Dornier - > MMS-UK (ASAR instrument prime) - > Alcatel - > MMS-UK (TCIU team). Both MMS-UK teams were in Portsmouth. Alcatel were only there because of 'juste retour'* and they didn't even seem to bother trying to interpret the MMS-UK ASAR requirements, they just changed the front page to have "Alcatel" on it. We basically had a shed-load of requirements placed on us that had nothing to do with what the TCIU needed to do, and Alcatel never did get round to formally specifying the bit we really did need from them (the TCIU -> T/R Module - an Alcatel device - interface) as far as I can remember!

It was good in a way, but Alcatel certainly, and possibly also Alenia, played politics all the way through. We were required to go through Alcatel to get them to clarify some of the requirements that were relevant and had come from MMS-UK. As they had no idea what they meant, Alcatel had to go to MMS-UK to get the clarification. Fortunately Alcatel appeared to want to do as little work as possible for their money so they'd just forward the clarification from MMS-UK without bothering to try to understand it.

I'm sure lots of people have been in similar situations, but the inefficiency could've been disastrous, especially as we (the MMS-UK teams) had been working directly with each other on ASAR for years before Alcatel were put in to split us up, and we used the same canteen!

Ah well, those were the days. Apologies for going so far off-topic, but it was nice to reminisce :-)

* Similarly, the ASAR CESA (Central Electronics SubAssembly) requirements came Dornier - > MMS-UK - > Alenia - > MMS-UK.

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Subject: [OT] ESA project memories (was Re: is Ada used in James Webb Space Telescope software?)
Date: Fri, 31 Dec 2021 23:18:49 +0200
Newsgroups: comp.lang.ada

> We didn't really have that problem. On TCIU most of our requirements came from Dornier - > MMS-UK (ASAR instrument prime) - > Alcatel - > MMS-UK (TCIU team). Both MMS-UK teams were in Portsmouth.

Interesting :-). I had a similar, but inverse, experience in a later project (SW for the Flexible Combined Imager instruments on the MTG satellites) where Thales Alenia Space (France) was both our customer for the whole SW and our subcontractor for a part of the SW. It led to a number of "direct" communications and decisions between the two TAS-F teams that bypassed our team (in Finland) and of which we learned later. But not much harm done, overall a good project.

> Alcatel were only there because of 'juste retour'*

I can't complain about "juste retour" as without it much less ESA work would be given to Finnish companies, especially earlier when Finland was a new ESA member with no experience in ESA work.

(For those not in the know: "juste retour" is the ESA policy by which ESA tries to give enough project work to each of its member countries to correspond to the country's share of ESA membership fees.)

[...]

> I'm sure lots of people have been in similar situations [...]

Although splitting work up into several companies does easily make for inefficiency, in can also have the benefit of documenting stuff that otherwise might be lost in internal e-mails or face-to-face discussions. That is, if the companies involved do their work properly, and don't act as you describe for Alcatel. But perhaps the Alcatel technical people did as well as they could to mitigate a poor higher-level decision, by being basically a transparent conduit, as you describe.

From: John Mccabe
<john@nospam.mccabe.org.uk>
Date: Wed, 5 Jan 2022 16:43:11 -0000

> Interesting :-). I had a similar, but inverse, experience in a later project [...]

It would be inappropriate of me to say whether or not that sort of behaviour occurred on ASAR, although I seem to remember occasions where Alcatel waived their right to be piggy-in-the-middle as some of the discussion about SAR pulse timing and the effect of shifting things around a bit, to deal with the fact that we would've needed a mid-90s supercomputer (and a substantial re-design of the TCIU -> T/R Module interface) to achieve what was originally specified, would've fried the brains of the people who were actually involved :-)

<snip>

> Although splitting work up into several companies does easily make for inefficiency, it can also have the benefit of documenting stuff that otherwise might be lost in internal e-mails or face-to-face discussions. [...]

To be fair (to MMS!), the actual documentation that was produced at the instrument level was pretty good. To be fair to Alcatel, as I mentioned, we'd been working without them on this for a long time before ESA decided to mandate that they should "manage" the TCIU development as a subcontract, so they were forced to pick up on stuff they pretty much hadn't cared about before.

Ironically none of this helped with the documentation from Alcatel; the TCIU -> T/R Module interface I mentioned, for example. We went through 3 rounds of TCIU Software Requirements reviews (i.e. SRR, then re-visited at ADR and DDR or something like that), where our assumptions on how that interface worked (based on rough sketch ideas we'd been given rather than formal specification) were described, before someone at Alcatel bothered to read it and say "nah, doesn't work like that" (presumably in French) :-)

Ada and Other Languages

AdaCore Joins with Ferrous Systems to Support Rust

From: Paul Rubin <no.email@nospam.invalid>
Subject: Adacore joins with Ferrous Systems to support Rust
Date: Wed, 02 Feb 2022 00:57:33 -0800
Newsgroups: comp.lang.ada

https://blog.adacore.com/adacore-and-ferrous-systems-joining-forces-to-support-rust

Ferrous Systems is apparently a Rust support company based in Germany. From the linked page:

"Ferrous Systems and AdaCore are announcing today that they're joining forces to develop Ferrocene - a safety-qualified Rust toolchain, which is aimed at supporting the needs of various regulated markets, such as automotive, avionics, space, and railway."

No mention about whether there will be any type of FOSS or community release. No word on whether the compiler and/or toolchain will be based on the existing stuff, or something new. Wonder how they will safety-certify anything in Rust when the language itself doesn't even have a formal spec. But, it is an interesting development.

Is the writing on the wall for Ada?

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 2 Feb 2022 13:04:42 +0000

I see this going one way, Ada loses out as the Rust side uses AdaCore to get what they want.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Wed, 2 Feb 2022 07:29:12 -0800

If possible please tell what Rust has to offer over Ada. From a quick look at the Rust book it seemed weaker in structured programming, generic programming, type system.

Thanks.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 02 Feb 2022 08:19:37 -0800

> Is the writing on the wall for Ada?

Yes. And it says:

As long as people care about quality software engineering, they will use Ada.

:)

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 2 Feb 2022 16:36:46 +0000

> If possible please tell what Rust has to offer over Ada.

[...] not a lot, only the borrow checker stuff.

From: John Mccabe <john@mccabe.org.uk>
Date: Thu, 3 Feb 2022 15:29:17 -0800

> If possible please tell what Rust has to offer over Ada.

A very nasty syntax, and there's an annoying thing where it moans about what you've called your project.

TBH that's about as far as I got with Rust; I can't be doing with pedantic restrictions that have no technical benefit (as far as I can see).

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Wed, 2 Feb 2022 10:48:44 -0800

> Is the writing on the wall for Ada?

Don't worry too much, people said that already more than 30 years ago... But perhaps the company will rebrand itself RustCore :-) ?

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 02 Feb 2022 12:03:03 -0800

> But perhaps the company will rebrand itself RustCore :-) ?

If the new IDE is called Oxide, watch out ;-).

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 02 Feb 2022 12:06:16 -0800

> I see this going one way, Ada loses out as the Rust side uses AdaCore to get what they want.

I don't think this is two companies merging. It's two companies working jointly on a particular product. Idk any more than the press release though.

Regarding Rust vs Ada, I've never heard anything from anyone who is a real expert at both. Superficially it looks to me like Rust's type system really is more precise than Ada's in general, although it doesn't have integer range types. In other stuff like modules, Ada is probably still ahead.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Wed, 2 Feb 2022 21:07:44 +0100

> If possible please tell what Rust has to offer over Ada.

Embedded systems reportedly still mostly use C. Consequently, C is the thing to which a language's merits must be compared. If Rust managers manage to persuade C shops to use Rust, the this is a reasonable attempt at improving C based program production. If Ada influences the process, then it has had a purpose. :-)

Perhaps it is easier to add guarantees to Rust programs than to C programs. Also, C programmers might feel more at home when using Rust. Java programmers also, even when it is just curly braces.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 3 Feb 2022 01:34:40 +0000

> [...] have integer range types.

Apparently, they have ranges as templates, can't see how that compensates.

[...]

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 02 Feb 2022 18:20:04 -0800

> [Ada vs Rust] Er, try learning both and you'll see?

It's a big effort to become expert at either, let alone both. Basic or superficial knowledge isn't helpful for such comparisons. I've read "Ada Distilled" (Ada 95 version) but still have no clue understanding most of the Ada discussions in this newsgroup, so there is a big gap between basic and advanced knowledge of Ada.

>> [type] system really is more precise than Ada's [...]

From what I've heard, Rust's type system is similar to Haskell's. Haskell's type system can verify stuff just by typechecking, that might be doable in Ada using SPARK and maybe an external proof assistant, but not with just types. Example: a red-black tree using Haskell GADT's (Generalized Algebraic Data Types):

https://www.reddit.com/r/haskell/comments/ti5il/redblack_trees_in_haskell_using_gadts_existential/

> Ada's ahead in most things.

Idk, I'd like to know more. I've never done anything serious in Ada and nothing at all in Rust, but C++ seems a lot more fluid than Ada, and Rust is supposed to compare well with C++. C++ of course is terrible in terms of reliability but I'm only referring to the effort needed to bang out a chunk of code.

From: Luke A. Guest
<laguest@archeia.com>
Date: Thu, 3 Feb 2022 02:52:25 +0000

> It's a big effort to become expert at either, let alone both. Basic or superficial knowledge isn't helpful for such comparisons.

You don't need to learn both languages inside and out. You pick a project and implement it in both languages, that project has to be specific to what you are wanting to know about whether it's binding or tasking or whatever, Ultimately with Ada, you can get by knowing the Ada subset and the representation clauses and do most of what you need to do to compare to another language, obviously if you want to compare tasking, you need to go further.

> [...] there is a big gap between basic and advanced knowledge of Ada.

Learn the basic part, it's equivalent to Pascal with proper type ranges, Pascal only has subtypes iirc. That's the main part of the RM, no annexes, you can do without access and tagged/controlled types to start with. At uni, we didn't even touch on tagged types, but used controlled types, it is possible.

> Haskell's type system can verify stuff just by typechecking

So it's no different than a C++ compiler checking against classes or a variant of C with strong typing? Still no ranges, subranges, ranges with holes in, etc. This is where the power is.

[...]

From: Björn Lundin
<b.f.lundin@gmail.com>
Date: Thu, 3 Feb 2022 10:54:43 +0100

> a lot of effort in Ada programming goes into making programs never crash. For example, if the program runs out of memory during operation it might crash, so Ada programs are often written to never allocate new memory after a startup phase. In C++ or Rust, it's important not to get wrong answers like 2+2=5, but if your program runs out of memory and crashes, go get a bigger computer. So idiomatic C++ and Rust programming uses dynamic allocation freely, and that makes some kinds of programming more convenient, at the expense of tolerating possible crashes.

That depends on the domain. Perhaps it is true in embedded. I use Ada for large systems, talking to databases, and lots of administrative work. On a win/Unix server. We would not dream of pre-allocate memory at startup. We allocate and dispose as needed.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Thu, 3 Feb 2022 21:20:08 -0600

> How else would you do controlled types [if not with tagged types]?

Ada 9x originally had a bunch of magic attributes (similar to streaming). It was very complex and got dumped in the dustbin during "scope reduction". Later on, some of us were bemoaning that a critical feature (finalization) had gotten lost in Ada 9x, and Tucker came up with the idea to build it on top of tagged types as a massive simplification (at the loss of a bit of power). People often complain that Ada finalization is complex, and it is, except all of the alternatives are a lot more complex. :-)

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Thu, 3 Feb 2022 21:38:16 -0600

...

> Well are you familiar with red-black trees? They are a data structure similar to B-trees which you may have seen. Basically the trees have nodes that are coloured either red or black, and there are some rules such as that internal red nodes can have only black children, and that enforces some invariants that keep the trees balanced so that lookups and updates are guaranteed to run fairly fast.

> Now these trees have been implemented in many languages, and if you look at almost any implementation, there is usually a test suite or internal debugging code to make sure that the invariants are preserved after doing an update. It is quite easy to make a mistake after all. But the Haskell code doesn't have those tests. Why not? Because the invariants are enforced by the datatype! If you make a mistake and mess up an invariant, your code won't compile! It's the difference between checking a subscript at runtime, and verifying that it in range with SPARK. SPARK lets you get rid of the runtime check. Haskell's type system is able to do similar things.

Cool, and most likely useless. OOP is like that in many ways, it lets you make a bunch of static checks, but to get them, you have to contort your design in awful ways. And then it is hard to extend, as you have a huge number of routines to override to get anything done.

[...]

The key for a programming language design is to minimize what Ada calls erroneous execution (undefined behavior), because it is that possibility which stop proofs in their tracks (or at least should; at least some tools ignore that possibility and essentially give garbage results as a consequence). Ada needs work in that area, but most other languages need more -- Ada at least detects most problems with dynamic checks.

Anyway, that's my 20 cents (inflation, you know). :-)

From: Paul Rubin
<no.email@nospam.invalid>
Date: Thu, 03 Feb 2022 21:19:14 -0800

> In any language with dynamic checks, one can easily reduce the problem of correctness down to simply proving that no check fails.

Well sure, but I'd take issue with the word "easily" there. The invariants for the red-black tree are fairly complicated. Enforcing them could be done with what I think SPARK calls a "ghost function" or something like that: a piece of code that is not actually executed, but is used only by the prover. But, what would the proof look like I think it would likely involve some manual work with an external proof assistant like ISABELLE. That takes quite a bit of effort and knowledge on the programmer's part.

On the other hand, the RB tree type declaration in that Haskell example is very natural and appealing even in that old implementation, and using newer GHC features that have appeared since then, it becomes even nicer.

[...]

From: Luke A. Guest
<laguest@archeia.com>
Date: Fri, 4 Feb 2022 10:28:33 +0000

> Ada 9x originally had a bunch of magic attributes [for finalization]

Now I want to know what these magic attributes were! Were they specific to a version of OO? Or were they to enable finalization?

From: Andreas Zeurcher
<zuercher_andreas@outlook.com>
Date: Fri, 4 Feb 2022 09:51:59 -0800

> Now I want to know what these magic attributes were! Were they specific to a version of OO? Or were they to enable finalization?

Randy, I agree with Luke: were these intermediate design proposals lost entirely or have they (as still extant) have simply not been released publicly? I suspect that at least some of these attributes have nowadays analogues in C++ smart pointer's & Objective-C/Swift's ARC {strong, weak, plain old data not needing finalization, pointed-to-object ownership, presence of finalization

subroutine/function/procedure a.k.a. finalizer/destructor, whether this finalizer in a subtype displaces its parent's finalizer versus this finalizer in a subtype chains its finalizer to all of its ancestors' finalizers unwound from most-derived to underived-root, … and so forth}. Or was Tucker's set of magic attributes going an entirely different direction? That intermediate proposal under consideration back in the 1st half of the 1990s might be a quite interesting read (especially by a reader with an interest in envisioning an Ada-esque analogue of Rust's borrow-checker algorithm).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 4 Feb 2022 22:31:40 -0600

> Now I want to know what these magic attributes were! Were they specific to a version of OO? Or were they to enable finalization?

They were specifically for finalization, and got called automatically in various places.

Re: Andreas. So far as I know, the documents existed only on paper - there never were any electronically distributed versions outside of the Ada 9x team (and possibly the printers). I still have a set of them on my bookshelf -- I look at them periodically to see where current oddities appeared in the language (and possibly to get some idea why). [But see below.]

Looking in the RM 3.0 (the final version was 6.0 for reference), it already had the tagged type version, but they were derived from an implementation defined type "Finalization_Implementation", and what became Adjust was named Duplicate.

Looking in ILS 1.2 (a working document full of ideas but not quite a complete RM, dated Dec 1992), I can't find any sign of finalization. It must have been gone by then.

I do have a large number of older documents somewhere in storage, but this isn't worth digging around in there to find out. Most of those were incomplete design documents.

You might be able to find something about that design in the Ada 9x mail archive or in the LSNs (Language Study Notes). You can find them in the AdaIC archives. Rooting around in there, there are some promising looking documents in the "history" section of the AdaIC archives. There is a directory of stuff called "9x-history"; there probably is interesting stuff there.

http://archive.adaic.com/pol-hist/history/9x-history/

LSNs are found in:
http://archive.adaic.com/standards/95lsn/

The Ada 9x mail archive (These were known as "MRT comments"):

http://archive.adaic.com/standards/95com/mrtcomments/

The comments of interest here are probably in the ZIPed comments rather than the more recent ones. (These are text files, I think, even though they don't have a usual extension.)

From: amo...@unizar.es <amosteo@unizar.es>
Date: Fri, 11 Feb 2022 09:40:14 -0800

> If possible please tell what Rust has to offer over Ada.

In my minimally informed opinion after going through parts of the official tutorial a couple of times, what Rust has to offer in general:

+ Memory safety (no leaks, double-free, race conditions*) by default.

- Terrible illegible syntax.

+ Safer/more expressive/more modern constructs than C.

+ Modern tooling shared by all the community.

[*] I guess in a protected-object sense, not in a high-level logic sense. But I don't really know.

The thing is that C is so basic and C++ so prone to shooting yourself in the foot, that Rust hits a middle ground that feels like the best thing since sliced bread to C/C++ programmers wishing for something better. Add to that the true novel contribution to a mainstream language that is memory safety (this is really a new way of thinking when you get into Rust), that if you don't know better (e.g., Ada) it really is overwhelmingly compelling. I'm not surprised at the cult-like following (I mean, we feel like that sometimes in the Ada world, right?) In a sense, Rust is the Ada of a younger generation, and without the baggage.

Of course you sometimes have to use "unsafe" programming evading the borrow checker, just like pointers are sometimes a necessity in Ada; and the legibility becomes truly awful IMHO really fast (to me, this is THE Achilles heel nobody seems to care too much about), but as I said, it has a couple of real selling points over the competition. Of course, if legibility is not your concern because you're used to C++ templating nightmares, you don't feel that particular pain. It's always the same story with Ada; most people don't know better to realize what they're missing.

The whole memory safety thing with the borrow checker goes beyond a gimmick, and it has a solid quality which goes beyond "in Ada you don't need pointers most of the time". It's a compile-time check, and it makes evident that runtime checks are a poor substitute. I'm more ashamed now of the whole anonymous pointers and accessibility surprises in Ada. Yes, SPARK added something similar for pointers, but in Rust it is for all variables. The equivalence in Ada would be not being able to use the same variable in two consecutive calls as an in-out parameter. So it's not the same, besides being only in SPARK.

Not having done anything of real import, I'm not sure how inevitable it is to go unsafe in Rust. My guess is that it will be hidden in libraries just like the Ada standard containers contain some scary pointer use (and I mean that I wouldn't like to have to understand what is going on there with the tampering checks etc.) At that point, obviously, you've lost the most solid selling point IMHO. Ada is safer not in a single sense, but as a whole design.

All in all, Rust has one big thing that Ada hasn't, which is the borrow checker.

And that is how I would summarize it: Rust is better in a single narrow sense, but Ada is better as a general language. Which is, not surprisingly, the consequence of the design motivations for each, which were precisely to have a memory-safe language and a high-integrity-oriented language. So the funny thing is that both have succeeded at their objective.

I really miss not having the time to become proficient in Rust at least to be able to properly compare. I think the memory safety is great to have (and if Ada were designed today, I guess it should play the same integral part, if practical), but Rust is demanding on the programmer in a way that C/C++ aren't, and the maintainability seems suspect, so I don't know how far it will carry Rust into the future. I guess it could absorb a big part of both new C and C++ development.

Boy, can I write a lot sometimes...

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 11 Feb 2022 19:24:02 +0000

> The thing is that C is so basic and C++ so prone to shooting yourself in the foot, that Rust hits a middle ground that feels

I'd say C++ is like a "backgun" (search images).

> like the best thing since sliced bread to C/C++ programmers wishing for something better. [...]

Exactly, people want something better, but for some reason CAN NOT accept anything that doesn't look like C/C++.

> [...] It's always the same story with Ada; most people don't know better to realize what they're missing.

And refuse to look for better, just accepting to continue using the same old, same old.

> The whole memory safety thing with the borrow checker goes beyond a gimmick, and it has a solid quality which goes beyond "in Ada you don't need pointers most of the time". It's a compile-time check, and it makes evident that runtime checks are a poor substitute.

So, you'd prefer, if Ada was designed now, it didn't do runtime checks (on pointers) and have compile-time checks?

> Yes, SPARK added something similar for pointers [...]

They added memory tracking at gnatprove time, much like the borrow checker afaik, which is an additional step.

[...]

> All in all, Rust has one big thing that Ada hasn't, which is the borrow checker.

I've not learnt any rust yet and that is my conclusion from what I've read. I need to do some tutorials at some point, but I also need eye bleach.

[...]

From: John Perry <devotus@yahoo.com>
Date: Fri, 11 Feb 2022 21:22:50 -0800

> I really miss not having the time to become proficient in Rust at least to be able to properly compare.

I've followed this thread with some interest. I've had to learn & use Rust at work; it has its ups and downs.

> + Memory safety (no leaks, double-free, race conditions*) by default.

Here's what Rust promises: https://doc.rust-lang.org/nomicon/races.html

"Safe Rust guarantees an absence of data races, which are defined as... [omitted] Rust does not prevent general race conditions. This is pretty fundamentally impossible, and probably honestly undesirable. ... So it's perfectly "fine" for a Safe Rust program to get deadlocked or do something nonsensical with incorrect synchronization. ... Still, a race condition can't violate memory safety in a Rust program on its own."

> In a sense, Rust is the Ada of a younger generation, and without the baggage.

Not quite. It's kind of discouraging to me how many of the older generation roll their eyes when you mention Ada, or relate stories of how it didn't work out for previous places of employment.

> Of course you sometimes have to use "unsafe" programming evading the borrow checker, just like pointers are sometimes a necessity in Ada...

The only time I've found it necessary (so far) to use the "unsafe" keyword is when interfacing with C or C++. There are people, and probably entire fields of IT, where "unsafe" may be much more common.

> and the legibility becomes truly awful IMHO really fast (to me, this is THE Achilles heel nobody seems to care too much about

I agree with this. It's not as bad as C++, not even as bad as C IMHO, but the braces get old. If not for IDEs that can help navigate them, I'd get lost pretty easily.

> The whole memory safety thing with the borrow checker goes beyond a gimmick [...] It's a compile-time check

In addition, the compiler's error messages are *very* useful, better than GNAT's for certain. The only time we have trouble making sense of them is, again, when we have to interface with C or C++ code.

(Well, except when I was learning. I got pretty frustrated with the compiler error messages at times. And I still haven't figured out Rust's manner of organizing modules; if I do understand it, then "lib.rs" is a much bigger deal than I think it should be. But I probably just don't understand well enough yet.)

> [...] The equivalence in Ada would be not being able to use the same variable in two consecutive calls as an in-out parameter.

Maybe I misunderstand, but I think the analogy's incorrect. When you designate a Rust variable as mutable, you can in fact have two consecutive calls in a manner akin to "in out", _so long as_ the function declares it wants to "borrow" the variable as mutable, *and* so long as the caller gives permission for both the borrow and the mutability. If it doesn't, the compiler gives a very clear error message.

I'm not sure Ada has anything comparable to that.

> Not having done anything of real import, I'm not sure how inevitable it is to go unsafe in Rust.

At work we have a fairly non-trivial Rust system that, as far as I know, goes unsafe only when... you can fill in the blank. :-)

> And that is what how I would summarize it: Rust is better in a single narrow sense, but Ada is better as a general language.

I haven't played with Ada's task & rendezvous mechanisms in a long time. Do they guarantee an absence of data races? If not, I'd say that's something else Rust has that Ada doesn't. I think SPARK does guarantee that, though. (If I understand correctly, the key is to disallow mutable objects being passed to multiple tasks / threads / etc.)

> Rust is demanding on the programmer in a way that C/C++ aren't...

Perhaps, C/C++ are demanding on the programmer in all kinds of ways that Rust isn't, and none of those ways is good. ;-) Whereas Rust's demands are pretty much all good (in comparison to C/C++).

I would also add that Rust has an amazing and effective ecosystem of libraries that are extremely easy to download and build, all as part of the generally-used Cargo build tool, which as far as I can tell is much easier to use and much more robust than ant, gradle, make, etc. I have the impression that alire is inspired by Cargo, but I haven't used alire at all yet, so I don't know how it compares beyond the ability to create projects and download libraries. I also don't know if alire is nearly as comprehensive as what Cargo offers (see, for instance, https://crates.io/, which offers tens of thousands of crates, and https://docs.rs/, which documents them -- alire has about 220 crates).

I have a feeling that abundance of crates, and the ease of incorporating and using them, has at least as much appeal as the guarantees on any safe code you may write.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Sat, 12 Feb 2022 02:08:11 -0800

> > and the legibility becomes truly awful IMHO really fast [...]

> I agree with this. It's not as bad as C++ [...]

Agree too, but only because they use K&R style. I find Allman style quite readable.

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Sat, 12 Feb 2022 18:34:04 +0100

> So, you'd prefer, if Ada was designed now, it didn't do runtime check (on pointers) and have compile-time checks?

I'd prefer that, as much as feasible, checks were moved (not removed!) to compile-time, yes. I know there are efforts in this direction at AdaCore to simplify the accessibility checks model.

>> I'm more ashamed now of the whole anonymous pointers and accessibility surprises in Ada.

> I'm not sure what you mean here.

My problem with runtime checks (which are undoubtedly better than no checks, sure), and in particular with accessibility checks, is that sometimes you get a failure much later during testing. By that time, understanding the problem may be 1) hard and 2) require painful redesign. At compile-time you get to deal with the problem immediately.

This is something in which Rust and Ada share the sentiment: "if it compiles, it works". So having something in another language found at compile-time makes me want to have it also in Ada at compile-time. It really spoils you against runtime checks. Much like I prefer the static elaboration model in GNAT instead of the dynamic one.

Also there are times in Ada where static checks are false positives that require some 'Unchecked_Access, and other times there is no failure yet you're doing something wrong. I find these from time to time in pretty obscure combinations not easy to provide a reproducer and frankly, I hate it. I'm never sure if I'm at fault, the compiler is at fault, or I've hit a corner case in the "heart of darkness". Nowadays I won't use a pointer even if it means obscene underperformance, until the thing is unavoidable.

There are also situations in which marking a parameter as aliased, even if you know it is already by reference (a limited/tagged type), will alter the things you can do with 'Access/'Unchecked_Access. There have been a couple of recent posts about that. Even if it's my fault, I find too hard to repeatably remember the finer details.

From: Alejandro R. Mosteo <alejandro@mosteo.com>
Date: Sat, 12 Feb 2022 19:24:02 +0100

> I agree with this. It's not as bad as C++, not even as bad as C IMHO, but the braces get old.

For me, it's not so much the braces as the reference/lifetime &'!<> soup.

> Maybe I misunderstand, but I think the analogy's incorrect. [...]

Yes, it is as you say. It is not a perfect analogy, and rethinking a bit more about it it's possible I was wrong including non-pointers. In Ada there's no trouble by default either; you have to mark things aliased, or take an 'Access/'Unchecked_Access to start to get into trouble.

With tasking involved is another matter, there Ada provides no safety when using global variables.

> I'm not sure Ada has anything comparable to that.

No, I think that's the novelty in Rust, the single-ownership model.

> I haven't played with Ada's task & rendezvous mechanisms in a long time. Do they guarantee an absence of data races?

Not for global variables, yes for task-local ones. For any decent design you'd encapsulate any shared data in a protected object or task, which would give the same assurance as the bit you quoted for Rust. Then there's Pragma Detect_Blocking, but that will only work for protected operations, and two tasks getting in a mutual deadlock needs not to involve protected operations.

> If not, I'd say that's something else Rust has that Ada doesn't. I think SPARK does guarantee that, though. (If I understand correctly, the key is to disallow mutable objects being passed to multiple tasks / threads / etc.)

I agree here. Rust prevents misuse of global variables at the low level of simultaneous access (from what you referenced before). This certainly can be useful in refactorings going from a single- to a multi-threaded design. In Ada you'd have to inspect every package for global state. SPARK deals with that, of course, but again: not Ada.

>> Rust is demanding on the programmer in a way that C/C++ aren't...

>

> Perhaps, C/C++ are demanding on the programmer in all kinds of ways that Rust isn't, and none of those ways is good. ;-) Whereas Rust's demands are pretty much all good (in comparison to C/C++).

Sure, that's a good point: it's not easy but it's for a good cause. That's another point I see in common with the Ada compiler. Still, I feel Ada is simpler for beginners. You don't need to face the harshness of the more complex aspects of the language, perhaps owing to simpler imperative roots. In Rust you must face the borrow checker head on.

> I would also add that Rust has an amazing and effective ecosystem of libraries that are extremely easy to download and build, all as part of the generally-used Cargo build tool,

The ecosystem certainly is a selling point. Look at Python; for quick and dirty there's nothing better simply because you know there's going to be the library you need.

> I have the impression that alire is inspired by Cargo, but I haven't used alire at all yet, so I don't know how it compares beyond the ability to create projects and download libraries.

The inspiration is there in a broad sense, but Alire is much younger and the manpower behind it is a fraction. For now we strive to cover the basics. There's also ideas from opam, virtualenvs, ... In some aspects Alire may be even more comprehensive (like crate configuration).

> I also don't know if alire is nearly as comprehensive as what Cargo offers [...] alire has about 220 crates).

No need to guess, given the difference in size of the communities.

> I have a feeling that abundance of crates, and the ease of incorporating and using them, has at least as much appeal as the guarantees on any safe code you may write.

Sure it's appealing. In many (most?) cases, as you say, it can tip the scales. Still, I don't think anyone that keeps on using Ada is doing it for the amount of ready-to-use libraries. It's a problem to attract new people, though. And it may disqualify Ada when a particular dependency is important and too costly to bind.

From: John Perry <devotus@yahoo.com>
Date: Sat, 12 Feb 2022 15:59:33 -0800

First, I agree with Alejandro about the reference/lifetime soup. The other day I saw an expression along the lines of &[&something] and thought, "what"? If I look & think hard enough, I can figure out what that means, but words would be so much nicer, and searching for the meaning of a word is a bit easier than searching for "[]".

On the other hand, again: the tooling and error messages are really very good. "cargo clippy" gives a lot of helpful advice/lint. I think one of the regular correspondents here sells or maintains one for Ada, but I can't remember the name [AdaControl —arm].

Anyway, I want to walk back part of what I said about Rust's safety, related to a post in a reddit thread where someone writes, "having students develop in Ada lead to them jumping from exception to exception until it worked, while other students writing code for the same problem in Rust lead to them swearing for 4 days until their code compiled and then being surprised that their code works, 100% as they expected and not ever producing a runtime error until the end of the two week practicum."

https://www.reddit.com/r/ada/comments/7wzrqi/why_rust_was_the_best_thing_that_could_have/

Maybe, but long term I'm not so sure.

Rust doesn't have a null in safe mode, so the idiomatic way to indicate an error is via a Result enum, which has two variants: Ok(result), where "result" is the desired result, or Err(msg), where "msg" is an error message.

https://doc.rust-lang.org/std/result/enum.Result.html

[...] In any case, handling the Result can be a little tedious (only a little but still!) so people often use the standard library's ".unwrap()" function instead. That means something akin to, "I'm confident the preceding expression had an Ok result, so just hand me the result. If it's an Err then go ahead and panic with Err's msg."

[...] But a lot of Rust users default to .unwrap() all the same, which makes me think that issue about Ada users jumping

from exception to exception may be a feature of a lot of Rust code, too. Depends on the self-discipline, I guess.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 13 Feb 2022 09:10:01 +0100

> In Ada you'd have to inspect every package for global state.

AdaControl has a rule for that: Global_References

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 14 Feb 2022 17:25:54 -0600

> Not for global variables, yes for task-local ones.

Ada 2022 has "conflict checking" to detect and reject bad uses of global variables. It's especially important for the parallel constructs (for which you don't have the syntactic guardrails that you get with tasks. It doesn't prevent every data race, but it eliminates most of them. (You can still get in trouble with accesses to multiple objects; that isn't necessarily safe even if accesses to the individual objects are.)

But, so far as I know, GNAT doesn't implement it yet.

From: Paul Rubin <no.email@nospam.invalid>
Date: Mon, 14 Feb 2022 20:29:34 -0800

> It doesn't prevent every data race, but it eliminates most of them

I wonder if we need software transactional memory: https://research.microsoft.com/en-us/um/people/simonpj/papers/stm/stm.pdf

https://research.microsoft.com/~simonpj/Papers/stm/beautiful.pdf

In the second paper, jump to the bottom of page 7 to skip a bunch of introductory stuff.

From: Kevin Chadwick <kevc3no4@gmail.com>
Date: Fri, 18 Feb 2022 05:24:15 -0800

> If possible please tell what Rust has to offer over Ada.

I haven't written a single line of Rust and do not intend to but I have done some research before and after choosing Ada, to confirm my choice due to Rusts popularity. My biggest preference is Ada's readability, of course some love brevity even when it adds complexity for some reason that I cannot understand.

Ada's type system has already been mentioned, but is fantastic.

Another is that Ada has a focus on stack and has tried to offer more heap tools in recent decades.

Rust has a focus on heap. I prefer the simpler stack default! Personally I avoided the heap, even with C.

I have heard that Rusts ownership model can cause problem with local/stack usage of owned memory (podcast interviewing a core maintainer "Rust with Steve Klabnik" but from 2015).

I have seen Rusts unsafe used even for simple things in embedded Rust whilst removing ALL of their 3 protections. Whereas with Ada you can more precisely permit pointer use and rarely need to.

https://docs.rs/svd2rust/0.19.0/svd2rust/#peripheral-api

```
struct PA0 { _0: () }
impl PA0 {
    fn is_high(&self) -> bool {
        // NOTE(unsafe) actually safe
        // because this is an atomic read with
       // no side effects
        unsafe { (*GPIOA::ptr()).idr.read().bits()
& 1 != 0 }
    }
    fn is_low(&self) -> bool {
        !self.is_high()
    }
}
```

Ada has been engineered to avoid pointer use, which I find appealing. Rust users would cite memory flexibility as appealing.

"Why Pascal is Not My Favorite Programming Language" by Kernighan is sometimes brought up, though much of it does not apply to Ada and no longer applies in any case and is clearly biased. Does he really promote the use of #include! Personally I blame flexibility points of view like his as the primary cause, as to why I have critical updates on my phone every single month and spend many days per year vulnerable to known exploits. Though really it is management at Vendors relentlessly pushing C. Maybe Rust can shift that closed point of view? I am aware that if my business does not succeed then the opportunity to write Ada, may go with it.

WRT compile time checks vs runtime: GO was written precisely because its authors were fed up waiting for C++ to compile. For me it is not important but worth bearing in mind. Personally I like the simplicity of runtime checks. I have much more faith in them than compile time checks! Though I confess to not knowing the details well enough to make that statement with complete confidence. It would also be nice to handle them more easily in a ZFP runtime.

SPARK sounds great and I like how it is intended to be applied where needed but I am dubious of any text that says it proves this or that, when it often depends on the tests implemented. I much prefer the language used by one AdaCore member in a podcast (Paul Butcher) along the lines of providing a high degree of confidence/assurance.

Ada versus Pascal

From: 711 Spooky Mart <711@spooky.mart>
Subject: Ada versus Pascal
Date: Thu, 21 Oct 2021 22:29:15 -0500
Newsgroups: comp.lang.ada

The little snippets of Ada code I've seen look _alot_ like Pascal.

What degree of learning curve is there to learn Ada, coming from a Pascal background? What kind of rough timeframes to get comfortable with programming without always looking into the manuals?

Where is the best starting point for a Pascal programmer to get up and running with Ada?

From: Ldries46 <bertus.dries@planet.nl>
Date: Fri, 22 Oct 2021 08:18:07 +0200

> Where is the best starting point for a Pascal programmer to get up and running with Ada?

I learned programming in 1966/1967 in Algol 60. As seen in the Algol report that can be found on the internet, Algol 60 is mostly a language for defining algorithms. It does not define Input and Output procedures. Pascal is one of the languages that have Algol 60 as a predecessor as is Ada. I did learn Pascal from some course and later on I did learn Ada, the latter by just reading the book "Software Engineering with Ada" by Grady Booch. That was Ada 85, the first version of Ada. Ada is stricter than other languages and is meant to have NO Operating system dependent items, so if you cannot go around something there must be a package on each operating system having the same interface everywhere.

From: Paul Rubin <no.email@nospam.invalid>
Date: Thu, 21 Oct 2021 23:40:51 -0700

> What degree of learning curve is there to learn Ada [...]

Ada is not that hard to get started with, but it has orders of magnitude more stuff in it than Pascal does, and getting familiar with all the intricacies is a big task. I've fooled with Ada a little, I consider myself a language geek, I've been following this newsgroup on and off for years, but I can't understand that many of the discussions I see here.

If I wanted to do something serious with Ada, I'd start by working through an in-depth textbook rather than just an intro or tutorial.

For just getting started, I used "Ada Distilled" (easy to find online). It is pretty good, but there is an enormous amount of material that it doesn't cover.

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Fri, 22 Oct 2021 11:57:17 +0300

> The little snippets of Ada code I've seen look _alot_ like Pascal.

Yes. Pascal syntax had a lot of influence on Ada syntax. But, as others have said, (current) Ada has a lot more features than (original) Pascal.

Very roughly speaking, and off the cuff, Ada has evolved as follows, which also gives you a list of the main things to learn, in addition to the Pascal base:

- Ada 83: Pascal + much improved type system + modules (packages) + exception handling + generic programming + concurrency (tasks)
- Ada 95: added modular (unsigned) integer types, object-oriented programming (classes = tagged types), package hierarchies (child packages), and asynchronous inter-task communication (protected objects)
- Ada 2005: added Java-like interface types (limited multiple inheritance), a standard container library, and further standard libraries
- Ada 2012: added support for program proof and logical run-time checks (preconditions, postconditions, type predicates), more forms of expressions (if expressions, case expressions, quantified expressions), and more standard libraries.
- Ada 2022 (upcoming): adds fine-grained parallel execution, extends the ability to do static (compile-time) computations, adds library packages for arbitrary-precision and arbitrary-range integer and real arithmetic ("bignums"), and makes lots of sundry improvements.

> What degree of learning curve is there to learn Ada?

As you can see from the list above, there is quite a lot to learn before you know _all_ of Ada. A Pascal-like subset should not be hard to learn, and if you learn the rest of the features in more or less the same order as they were added to Ada, you will pass from one consistent, working language subset to a larger such subset at each step.

The only point where I suggest to learn features in a different order is in inter-task communication: asynchronous communication via protected objects is much easier than was the original, synchronous rendez-vous method in Ada 83 (but which is of course still supported).

> Where is the best starting point for a Pascal programmer to get up and running with Ada?

I think this depends a lot on how you like to learn - by reading technical text (manuals) or by experimentation. I'm a "read the manual" type (I learned Ada from the Ada 83 Reference Manual) so perhaps I would start with the Ada Wikibook at https://en.m.wikibooks.org/wiki/Ada_Programming, which extends up to the 2012 Ada standard (or so it claims, I haven't really read it, but I've heard good reports of it).

Perhaps you could take one of your smaller Pascal programs and translate it to Ada as a first step? As the next step, you could divide that program into packages, then add exception handling. And then take a new problem and write an Ada program from scratch.

Ask for help here, or in some other Ada forum, whenever in doubt about something.

From: 711 Spooky Mart
<711@spooky.mart>
Date: Fri, 22 Oct 2021 04:59:25 -0500

> Ada is stricter than other languages and is meant to have NO Operating system dependent items, so if you cannot go around something there must be a package on each operating system having the same interface everywhere.

By this do you mean the same syntax and libs will run on all target systems without fiddling with {IFDEF} and architecture compiler switch woo foo for USES and repetitive cross-arch boilerplate?

One thing I can't stand about Pascal is the totally different functions and logic from several operating systems that MUST be re-written several times in the same code base to do the same job. This drives me mad. In fact it irks me so much I was thinking of writing some libraries for things I do that would handle this all automatically across arches. There would go a couple months of Sundays.

Think IPC with Pascal. Get a good IPC routine going for Linux in your app, then you have to re-write it for MAC and Windows, and even some other flavors of *nix.

So am I to understand that the Ada compiler has somehow eliminated this problem, by ensuring every target OS has a syntactically conformant package to execute its methods using the same statements?

If I'm understanding you rightly, even though Ada sounds like a much more complex language than Pascal, it also sounds like it would have less surprises across arches.

Please elaborate if I'm misunderstanding. And thanks to everyone else who has responded.

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 22 Oct 2021 13:49:11 +0200

> What degree of learning curve is there to learn Ada, coming from a Pascal background?

Pascal was the starting point for the Green language, which became Ada in 1980 (and also for the Blue, Red, and Yellow languages, which did not). Ada is firmly in the ALGOL family of languages.

There is a sequential subset of Ada that Pascal users can learn very quickly: the sequential language + packages (packages [modules] are fundamental to Ada, and you can't do anything useful without them). One should then quickly learn generics, as much of the standard library is generic. Ada's features are mostly orthogonal, so one can use this subset without surprises from the other aspects of the language.

One can then learn programming by type extension (tagged types and interfaces) and concurrent programming (tasks and their friends) to complete your understanding of the language.

I generally recommend “Ada Distilled”

(https://www.adaic.org/wp-content/uploads/2010/05/Ada-Distilled-24-January-2011-Ada-2005-Version.pdf)

to those familiar with another imperative language. It's ISO/IEC 8652:2007 Ada, but you can easily pick up the new Ada-12 features when you've finished. (There's also now an Ada-12 version available on Amazon.)

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Fri, 22 Oct 2021 18:12:49 +0300

> By this do you mean the same syntax and libs will run on all target systems without fiddling with {IFDEF} and architecture compiler switch woo foo for USES and repetitive cross-arch boilerplate?

I'm not ldries46, but here is an answer: Ada standardizes _some_ functions for which some other languages use "OS" services, principally threading, which in Ada is the "tasking" feature. Indeed Ada tasking works in the same way whichever OS is used, and also in the "bare board", no-OS situation. This is very useful for developing multi-threaded embedded SW, because the Ada tasking code can be tested on desk-top workstations and then executed on the target system unchanged. (and no "ifdefs").

But real operating systems (as opposed to simpler real-time kernels) provide many services that are not standardized in Ada, for example inter-process communication.

> Think IPC with Pascal. Get a good IPC routine going for Linux in your app, then you have to re-write it for MAC and Windows, and even some other flavors of *nix.

Indeed.

> So am I to understand that the Ada compiler has somehow eliminated this problem, by ensuring every target OS has a syntactically conformant package to execute its methods using the same statements?

Sadly no.

However, there are some rudiments:

- There is a standardized Ada interface (binding) to POSIX services. This is implemented in an Ada library called Florist. If you find or make a Florist implementation for the OSes you use, your Ada program can use the same OS service interfaces on all those OSes.
- The gcc-based Ada compiler GNAT comes with a GNAT-specific library that provides some OS services with the same Ada API on any OS that GNAT supports. This includes some IPC, but I don't know exactly how far that goes, and the library may of course change from one GNAT version to the next.
- There is an Ada library called Win32Ada that provides an extensive set of Microsoft Windows services, but it is a "thin binding" meaning that even the API is Windows-specific.

The Ada applications I have created and worked with have needed only a few OS services, basically some IPC: text in and out via pipes to and from a child process. We implemented our own binding to the required OS services (pipe and process creation and destruction). The interface consisted of a package declaration (.ads file) that was basically the same for all the supported OSes (Windows, Linux, Mac OS X) but had different OS-specific package body files (implementations, .adb files). In practice I think the Linux and Mac OS X implementations were the same and used direct binding to fork() and pipe() etc. The Windows implementation used Win32Ada.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 22 Oct 2021 17:47:59 +0200

> But real operating systems (as opposed to simpler real-time kernels) provide many services that are not standardized in Ada, for example inter-process communication.

Ada 83 predates threads. Initially a task meant to be either scheduled internally or mapped onto system processes. It is not late now. One could allow the pragma Import for tasks (and protected objects) in order to communicate to an external process using rendezvous and protected actions.

> - The gcc-based Ada compiler GNAT comes with a GNAT-specific library that provides some OS services with the same Ada API on any OS that GNAT supports.

It provides sockets and serial I/O, one or both are vital for many applications.

- There is the annex E providing RPC and shared objects. Unfortunately it is very vague and underdocumented. [...]
- The simple components library provides inter-process communication primitives: mutexes, events, streams, pools, RPCs etc.

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Fri, 22 Oct 2021 13:05:01 -0400

>The little snippets of Ada code I've seen look _alot_ like Pascal.

No surprise. The teams that took part in the DoD competition to design a language to replace the mish-mash of languages being used in the 70s tended to choose Pascal as the starting point (Modula-2 hadn't escaped ETH-Zurich yet <G>).

The main difference is that Ada incorporated block closing syntax at the base, finding Pascal (and C) [begin/end, {/} respectively] usage error-prone (dangling else, etc.) along with using ; as a terminator instead of separator. Oh, and using (/) for both function arguments and array indexing (back then, most US keypunches didn't support [/] or {/}).

Declarations do not have a defined sequence (type, constant, variable).

Also, Pascal of the era typically did not support separate compilation and/or include files -- programs were all single monolithic files, any change required recompiling the entire program.

Pascal also had a relatively limited I/O system -- with the bad quirk that it did "pre-reads" of files. Made interactive/console programs difficult (or required special handling by the run-time startup) -- starting a program would result in stdin reading at least one character, if not one line, into the file buffer variable... But the program may not want the data until after lots of initialization and prompts.

>What degree of learning curve is there to learn Ada, coming from a Pascal background?

If all one is writing is "Pascal" type applications, without using complex data types (ie; defining specific types for each "concept") -- it shouldn't take too long.

Tasking, rendezvous, protected objects (not to be confused with private objects), and generics, may take longer to get comfortable with.

The appendices of the LRM will tend to get lots of usage; there are many subtleties to the standard libraries.

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Fri, 22 Oct 2021 13:00:45 -0700

> What degree of learning curve is there to learn Ada, coming from a Pascal background?

You don't need to read more manuals than you were used to for Pascal (so, it can be a wide range, depending on your way of learning). The most outstanding difference between both languages is the degree of unification.

Here are a couple of links, with a Pascal-then-Ada perspective, that could be useful:

http://p2ada.sourceforge.net/pascada.htm

http://p2ada.sourceforge.net/

From: Paul Rubin <no.email@nospam.invalid>
Date: Fri, 22 Oct 2021 17:29:26 -0700

> Also, Pascal of the era typically did not support separate compilation and/or include files

I thought Ada was originally like that too. The program could be split into multiple files, but they were expected to all be compiled together.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Oct 2021 20:17:02 -0500

> I thought Ada was originally like that too.

No. Some implementations were like that, but most supported fully separate compilations from the beginning. Janus/Ada certainly did (once we got packages implemented, and Ada without packages really isn't Ada at all). You might have been thinking about the original permission to require generic bodies to be available when compiling an instantiation, but that only applied to generic units, never "regular" units. And some compilers (like Janus/Ada) never used that permission.

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Sat, 23 Oct 2021 13:24:19 -0400

>I thought Ada was originally like that too. The program could be split into multiple files, but they were expected to all be compiled together.

No... Pretty much every build system for Ada focused on only rebuilding the parts affected by a changed file -- by following WITH statements to find required units (see the LRM for what a "unit" comprises) /and/ determining if that unit requires compilation. Timestamps or intermediate files may be used in that determination. Changes in implementation (body) require the body to be recompiled, but if the specification did not change, then units WITHing the specification don't need to be compiled -- they just need relinking with the updated body.

GNAT's build system -- using the host OS filesystem as the "database" -- required that separate files are generated for each unit. (cf: GNATCHOP) All-in-One was the optional source file format accepted by some compilers -- but other than the early language reference manuals, I haven't encountered any text books that use that means of presenting code examples (unless it is discussing the use of GNATCHOP itself <G>).

https://en.wikisource.org/wiki/Stoneman_requirements [Link replaced. See below Rosen's comment. —arm] is the requirements document that DoD used to define the desired environment around Ada development.

"""

4.E APSE TOOLSET REQUIREMENTS

4.E.1 The tools in an APSE shall support the development of programs in the Ada language as defined by the Ada reference manual. In particular an APSE shall support the separate compilation features of the language.

"""

Note the last sentence

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 24 Oct 2021 09:04:29 +0200

> https://www.adahome.com/History/Stoneman/stoneman.htm is the requirements document that DoD used to define the desired environment around Ada development.

Please don't provide links to adahome, this site is frozen since 1998, and there are copyright issues with the owner.

Stoneman can be obtained from:

https://en.wikisource.org/wiki/Stoneman_requirements

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 24 Oct 2021 09:04:29 +0200

> https://www.adahome.com/History/Stoneman/stoneman.htm is the requirements document that DoD used to define the desired environment around Ada development.

Please don't provide links to adahome, this site is frozen since 1998, and there are copyright issues with the owner.

Stoneman can be obtained from:

https://en.wikisource.org/wiki/Stoneman_requirements

From: Jerry <list_email@icloud.com>
Date: Sat, 23 Oct 2021 21:33:14 -0700

This thread began as a comparison of Ada and the original Pascal. So how does Ada compare to Free Pascal Compiler and Delphi which have gone far past original Pascal?

From: Ldries46 <bertus.dries@planet.nl>
Date: Sun, 24 Oct 2021 08:32:01 +0200

> how does Ada compare to Free Pascal Compiler and Delphi which have gone far past original Pascal?

You can also ask can you compile a Free Pascal program in Delphi or in the other direction. Ada was intended to do so and to keep it that way

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sun, 24 Oct 2021 09:51:26 -0700

> how does Ada compare to Free Pascal Compiler and Delphi [...]?

You find a very partial answer in the comparison here:

http://p2ada.sourceforge.net/pascada.htm#tables

Made around year 2000, so ~30 after original Pascal but ~20 years ago.

Note that both FPC and Delphi descend from Turbo Pascal, which is itself completely different from other extensions like ISO Extended Pascal.

In a nutshell, Pascal is an extreme example of fragmentation of a language into dialects.

Ada is on the other extremity: you can build the same source sets (I mean exactly the same sources, without preprocessing gimmicks) on completely different compilers & OSes.

From: 711 Spooky Mart <711@spooky.mart>
Date: Sun, 24 Oct 2021 18:24:21 -0500

> Ada is on the other extremity: you can build the same source sets (I mean exactly the same sources, without preprocessing gimmicks) on completely different compilers & OSes.

I think that answers the important original query.

Does modern Ada have facility for writing boot loaders, inline Assembly, kernels, etc.?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 25 Oct 2021 11:23:49 +0300

> Does modern Ada have facility for writing boot loaders, inline Assembly, kernels, etc.?

In-line assembly is supported by most of the Ada compilers I have used, but the syntax may differ across compilers.

The run-time systems (real-time kernels) associated with Ada compilers for bare-board embedded systems are typically written in Ada, with minor amounts of assembly language inserted for the very HW-specific parts such as HW context saving and restoring.

I'm not very familiar with boot loaders, but I see no reason why a boot loader could not be written in Ada. However, usually (and as for other languages) there will be a small start-up routine in assembly language to initialize the processor, set up a stack, and so forth. The "Ada Bare Bones" project is doing something like this, I believe:

https://wiki.osdev.org/Ada_Bare_bones

From: Luke A. Guest <laguest@archeia.com>
Date: Mon, 25 Oct 2021 09:40:38 +0100

Boot loaders and kernels are just another application area any general purpose language can target, even Ada.

> I'm not very familiar with boot loaders

If you're talking x86 on PC's, then you'll need to read up on the x86 boot process in which x86 starts up in 16-bit (real) mode, then has to be taken into protected and then long modes. You would need a GCC that can target all those modes.

> The "Ada Bare Bones" project is doing something like this, I believe

Thanks for pointing out my project :) It's out of date and doesn't build as is any more, but others have written Ada pages on that site since.

IIRC, the changes that need to happen is that gprbuild

--target=arm-<whatever> be used instead of arm-<whatever>-gprbuild

Ada Practice

Custom Storage Pool Questions

From: Jere <jhb.chat@gmail.com>
Subject: Custom Storage Pool questions
Date: Sun, 12 Sep 2021 17:53:47 -0700
Newsgroups: comp.lang.ada

I was learning about making user defined storage pools when I came across an article that made me pause and wonder how portable storage pools actually can be. In particular, I assumed that the Size_In_Storage_Elements parameter in the Allocate operation actually indicated the total number of storage elements needed.

```
procedure Allocate(
    Pool : in out Root_Storage_Pool;
    Storage_Address : out Address;
    Size_In_Storage_Elements : in
    Storage_Elements.Storage_Count;
    Alignment : in
    Storage_Elements.Storage_Count)
is abstract;
```

But after reading the following AdaCore article, my assumption is now called into question: https://blog.adacore.com/header-storage-pools

In particular, the blog there advocates for separately counting for things like unconstrained array First/Last indices or the Prev/Next pointers used for Controlled objects. Normally I would have assumed that the Size_In_Storage_Elements parameter in Allocate would account for that, but the blog clearly shows that it doesn't

So that seems to mean to make a storage pool, I have to make it compiler specific or else risk someone creating a type like an array and my allocation size and address values will be off.

Is it intended not to be able to do portable Storage Pools or am I missing some Ada functionality that helps me out here. I scanned through the list of attributes but none seem to give any info about where the object's returned address is relative to the top of the memory actually allocated for the object. I saw the attribute Max_Size_In_Storage_Elements, but it doesn't seem to guarantee to include things like the array indices and it still doesn't solve the issue of knowing where the returned address needs to be relative to the top of allocated memory.

I can easily use a generic to ensure that the types I care about are portably made by the pool, but I can't prevent someone from using my pool to create other objects that I hadn't accounted for. Unless there is a way to restrict a pool from allocating objects of other types?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 13 Sep 2021 00:29:35 -0500

Not sure what you are expecting. There is no requirement that objects are allocated contiguously. Indeed, Janus/Ada will call Allocate as many times as needed for each object; for instance, unconstrained arrays are in two parts (descriptor and data area).

The only thing that you can assume in a portable library is that you get called the same number of times and sizes/alignment for Allocate and Deallocate; there's no assumptions about size or alignment that you can make.

If you want to build a pool around some specific allocated size, then if it needs to be portable, (A) you have to calculate the allocated size, and (B) you have to have a mechanism for what to do if some other size is requested. (Allocate a whole block for smaller sizes, fall back to built-in heap for too large is what I usually do).

More likely, you'll build a pool for a particular implementation. Pools are very low level by their nature, and useful ones are even more so (because they are using target facilities to allocate memory, or need to assume something about the allocations, or because they are doing icky things like address math, or ...).

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 13 Sep 2021 13:12:39 +0200

> I was learning about making user defined storage pools when I came across an article that made me pause and wonder how portable storage pools actually can be. [...]

That blog shows a special use for Storage_Pools, where you allocate /user/ data on top of the requested memory. When called by the compiler, it is up to the compiler to compute how much memory is needed, and your duty is to just allocate that.

From: Jere <jhb.chat@gmail.com>
Date: Mon, 13 Sep 2021 17:48:15 -0700

> That blog shows a special use for Storage_Pools

Yes, but if you look at that blog, they are allocating space for the /user/ data and for the Next/Prev for controlled types and First/Last for unconstrained arrays in addition to the size specified by allocate.

I agree I feel it is up to the compiler to provide the correct size to Allocate, but the blog would indicate that GNAT does not (or did not..old blog..so who knows?). Does the RM require that an implementation pass the full amount of memory needed to Allocate when new is called

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 14 Sep 2021 08:08:48 +0200

The RM says that an allocator allocates storage from the storage pool. You could argue that it does not say "allocates all needed storage...", but that would be a bit far fetched.

Anyway, a blog is not the proper place to get information from for that kind of issue. Look at the GNAT documentation.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 14 Sep 2021 08:23:08 +0200

> Yes, but if you look at that blog, they are allocating space for the /user/ data and for the Next/Prev for controlled types and First/Last for unconstrained arrays in addition to the size specified by allocate.

I do not understand your concern. The blog discusses how to add service data to the objects allocated in the pool.

I use such pools extensively in Simple Components. E.g. linked lists are implemented this way. The list links are allocated in front of list elements which can be of any type, unconstrained arrays included.

The problem with unconstrained arrays is not that the bounds are not allocated, they are, but the semantics of X'Address when applied to arrays.

A'Address is the address of the first array element, not of the array object. For a pool designer it constitutes a problem of getting the array object by address. This is what Emmanuel discusses in the blog.

[The motivation behind Ada choice was probably to keep the semantics implementation-independent.]

Consider for example a list of String elements. When Allocate is called with String, it returns the address of all String. But that is not the address you would get if you applied 'Address. You have to add/subtract some offset in order to get one from another.

In Simple Components this offset is determined at run-time for each generic instance.

Of course, a proper solution would be fixing Ada by adding another address attribute:

X'Object_Address

returning the first address of the object as allocated.

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 14 Sep 2021 08:42:52 +0200

> X'Object_Address

> returning the first address of the object as allocated.

But you cannot assume that the object is allocated as one big chunk. Bounds can be allocated at a different place. What would be X'Object_Address in that case?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 14 Sep 2021 09:00:13 +0200

The object address, without bounds, same as X'Address.

What Allocate returns is not what A'Address tells. The compiler always knows the difference, the programmer has to know it too. Nothing more.

From: Jere <jhb.chat@gmail.com>
Date: Tue, 14 Sep 2021 17:21:04 -0700

> I use such pools extensively in Simple Components. E.g. linked lists are implemented this way. The list links are allocated in front of list elements which can be of any type, unconstrained arrays included.

The blog I saw was old, so it is completely possible it no longer is true that GNAT does what the blog suggests. I'll take a look at your storage pools and see how they handle things like this.

From: Jere <jhb.chat@gmail.com>
Date: Tue, 14 Sep 2021 17:39:43 -0700

> Anyway, a blog is not the proper place to get information from for that kind of issue. Look at the GNAT documentation.

I'll take a look at the GNAT docs to see (and of course that blog is old, so GNAT may not do this anymore anyways), but I am mainly asking in the frame of what Ada allows and/or expects. I'd like to be able to allocate storage simply without worrying how the compiler does it under the hood and just assume that any calls to Allocate will ask for the full amount of memory.

Am I correct to assume that Ada doesn't provide any language means to restrict what types a pool can make objects of. The times that I have wanted to make a pool are generally for specific types and it is often simpler to design them if I can assume only those types are being generated

Thanks for the response. I'm sorry for all the questions. That's how I learn and I realize it isn't a popular way to learn in the community, but I have always learned very differently than most.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 15 Sep 2021 08:54:07 +0200

It seems that you are under the impression that Allocate must allocate more size than its Size parameter asks. The answer is no, unless *you* wanted to add something to each allocated object.

[...]

Again, if attributes are added, then it should be the object address as allocated. The compiler always knows the proper address because this address is passed to Free, not X'Address!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 15 Sep 2021 21:07:22 +0200

[Continuing on the subject of how to retrieve the object in the pool from its ‘Address, when there is hidden data like a fat pointer, Dmitry challenges readers to implement a function that needs to do so. —arm]

Now define and implement the following function:

```
type Type_Pointer is access Some_Type
     with Storage_Pool => Pool;
  function Get_Allocation_Time
       (Pointer : Type_Pointer) return Time;
```

The function returns the time when the pointed object was allocated in the pool.

[...]

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Thu, 16 Sep 2021 00:12:58 -0700

I am the original implementer of GNATCOLL.Storage_Pools.Headers, and I have been silent in this discussion because I must admit I forgot a lot of the details... To be sure, we did not add new attributes just for the sake of GNATCOLL, those existed previously so likely had already found other uses.

As has been mentioned several times in the discussion, the compiler is already passing the full size it needs to Allocate, and the storage pool only needs to allocate that exact amount in general. This applies for the usual kinds of storage pools, which would for instance preallocate a pool for objects of fixed sizes, or add stronger alignment requirements.

In the case of the GNATCOLL headers pool, we need to allocate more because the user wants to store extra data. For that data, we are left on our own to find the number of bytes we need, which is part of the computation we do: we of course need the number of bytes for the header's object_size, but also perhaps some extra bytes that are not returned by that object_size in particular for controlled types and arrays.

Note again that those additional bytes are for the header type, not for the type the user is allocating (for which, again, the compiler already passes the number of bytes it needs).

The next difficulty is then to convert from the object's 'Address back to your extra header data. This is when you need to know the size of the prefix added by the compiler (array bounds, tag,...) to skip them and then take into account the alignment, and finally the size of your header.

Dmitry's suggested exercise (storing the timestamp of the allocation) seems like a useful one indeed. It would be nice indeed if GNATCOLL's code wasn’t too complicated, but I am afraid this isn't the case. We had used those pools to implement reference counting for various complex types, and ended up with that complexity.

AdaCore (Olivier Hainque) has made a change to the implementation since the blog was published (https://github.com/AdaCore/gnatcoll-core/commits/master/src/gnatcoll-storage_pools-headers.adb), so I got some details wrong in the initial implementation apparently.

From: Jere <jhb.chat@gmail.com>
Date: Thu, 16 Sep 2021 16:21:58 -0700

> In the case of the GNATCOLL headers pool, we need to allocate more because the user wants to store extra data. For that data, we are left on our own to find the number of bytes we need, which is part of the computation we do: we of course need the number of bytes for the header's object_size, but also perhaps some extra bytes that are not returned by that object_size in particular for controlled types and arrays.

> Note again that those additional bytes are for the header type, not for the type the user is allocating (for which, again, the compiler already passes the number of bytes it needs).

Thanks for the response Emmanuel. That clears it up for me. I think the confusion for me came from the terminology used then. In the blog, that extra space for First/Last and Prev/Next was mentioned as if it were for the element, which I mistook was the user's object being allocated and not the header portion. I didn't catch that as the generic formal's name, so that is my mistake. I guess in my head, I would have expected the formal name to be Header_Type or similar so I misread it in my haste.

I appreciate the clarity and apologize if I caused too much of a stir. I was asking the question because I didn't understand, so I hope you don't think too poorly of me for it, despite my mistake.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 17 Sep 2021 15:56:20 +0200

> I was asking the question because I didn't understand, so I hope you don't think too poorly of me for it, despite my mistake.

Nope, especially because the issue with X'Address being unusable for memory pool developers is a long standing painful problem that needs to be resolved. That will never happen until a measurable group of people start asking questions. So you are doubly welcome.

From: Simon Wright <simon@pushface.org>
Date: Fri, 17 Sep 2021 20:46:26 +0100

> the issue with X'Address being unusable for memory pool developers is a long standing painful problem that needs to be resolved.

There are two attributes that we should all have known about, Descriptor_Size[1] (bits, introduced in 2011) and Finalization_Size[2] (storage units, I think, introduced in 2017)

[1] https://docs.adacore.com/live/wave/gnat_rm/html/gnat_rm/gnat_rm/implementation_defined_attributes.html#attribute-descriptor-size

[2] https://docs.adacore.com/live/wave/gnat_rm/html/gnat_rm/gnat_rm/implementation_defined_attributes.html#attribute-finalization-size

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 17 Sep 2021 22:39:05 +0200

They are non-standard and have murky semantics I doubt anybody really cares about. What is needed is the address passed to Deallocate should the object be freed = the address returned by Allocate. Is that too much to ask?

BTW, finalization lists (#2) should have been removed from the language long ago. They have absolutely no use, except maybe for debugging, and introduce huge overhead. The semantics should have been either Unchecked_Deallocation or compiler allocated objects/components may call Finalize, nothing else.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 18 Sep 2021 00:17:50 +0300

> What is needed is the address passed to Deallocate should the object be freed = the address returned by Allocate. Is that too much to ask?

That is already required by RM 13.11(21.7/3): "The value of the Storage_Address parameter for a call to Deallocate is the value returned in the Storage_Address parameter of the corresponding successful call to Allocate."

The "size" parameters are also required to be the same in the calls to Deallocate and to Allocate.

> BTW, finalization lists (#2) should have been removed from the language long ago.

Huh? Where does the RM _require_ finalization lists? I see them mentioned here and there as a _possible_ implementation technique, and an alternative "PC-map" technique is described in RM 7.6.1 (24.r .. 24.t).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Sep 2021 09:49:16 +0200

> That is already required by RM 13.11(21.7/3): "The value of the Storage_Address parameter for a call to Deallocate is the value returned in the Storage_Address parameter of the corresponding successful call to Allocate."

You missed the discussion totally. It is about the X'Address attribute.

The challenge: write pool with a function returning object allocation time by its pool-specific access type.

> Huh? Where does the RM _require_ finalization lists?

7.6.1 (11 1/3)

> I see them mentioned here and there as a _possible_ implementation technique, and an alternative "PC-map" technique is described in RM 7.6.1 (24.r .. 24.t).

I don't care about techniques to implement meaningless stuff. It should be out, at least there must be a representation aspect for turning this mess off.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 18 Sep 2021 12:03:03 +0300

> You missed the discussion totally. It is about X'Address attribute.

Sure, I understand that the address returned by Allocate, and passed to Deallocate, for an object X, is not always X'Address, and that you would like some means to get the Allocate/Deallocate address from (an access to) X. But what you stated as not "too much to ask" is specifically required in the RM paragraph I quoted. Perhaps you meant to state something else, about X'Address or some other attribute, but that was not what you wrote.

Given that an object can be allocated in multiple independent pieces, it seems unlikely that what you want will be provided. You would need some way of iterating over all the pieces allocated for a given object, or some way of defining a "main" or "prime" piece and a means to get the Allocate/Deallocate address for that piece.

>> Huh? Where does the RM _require_ finalization lists?

> 7.6.1 (11 1/3)

RM (2012) 7.6.1 (11.1/3) says only that objects must be finalized in reverse order of their creation. There is no mention of "list".

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Sep 2021 12:22:45 +0200

> Perhaps you meant to state something else, about X'Address or some other attribute, but that was not what you wrote.

I wrote about attributes, specifically GNAT-specific ones used in the blog to calculate the correct address. "Too much to ask" was about an attribute that would return the object address directly.

> Given that an object can be allocated in multiple independent pieces, it seems unlikely that what you want will be provided.

Such implementations would automatically disqualify the compiler. Compiler-generated piecewise allocation is OK for the stack, not for user storage pools.

And anyway, all this equally applies to X'Address.

> RM (2012) 7.6.1 (11.1/3) says only that objects must be finalized in reverse order of their creation. There is no mention of "list".

It talks about "collection."

> Then your complaint seems to be about something specified for the order of finalization, but you haven't said clearly what that something is.

No, it is about the overhead of maintaining "collections" associated with an access type in order to call Finalization for all members of the collection.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 18 Sep 2021 18:59:27 +0300

>> Given that an object can be allocated in multiple independent pieces, it seems unlikely that what you want will be provided.

> Such implementations would automatically disqualify the compiler.

That is your opinion (or need), to which you are entitled, of course, but it is not an RM requirement on compilers -- see Randy's posts about what Janus/Ada does.

[...]

> No, it is about the overhead of maintaining "collections" associated with an access type in order to call Finalization for all members of the collection.

So you want a way to specify that for a given access type, although the accessed object type has a Finalize operation or needs finalization, the objects left over in the (at least conceptually) associated collection should _not_ be finalized when the scope of the access type is left? Have you proposed this to the ARG?

To me it seems a risky thing to do, subverting the normal semantics of initialization and finalization. Perhaps it could be motivated for library-level collections in programs that are known to never terminate (that is, to not need any clean-up when they do stop or are stopped).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Sep 2021 18:19:23 +0200

> So you want a way to specify that for a given access type, although the accessed object type has a Finalize operation or needs finalization, the objects left over in the (at least conceptually) associated collection should _not_ be finalized when the scope of the access type is left?

Exactly, especially because these objects are not deallocated, as you say they are left over. If they wanted GC they should do that. If they do not, then they should keep their hands off the objects maintained by the programmer.

> To me it seems a risky thing to do, subverting the normal semantics of initialization and finalization.

Quite the opposite, it is the collection rule that subverts semantics because objects are not freed, yet mangled.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sun, 19 Sep 2021 13:36:11 +0300

> Quite the opposite, it is the collection rule that subverts semantics because objects are not freed, yet mangled.

Local variables declared in a subprogram are also not explicitly freed (deallocated), yet they are automatically finalized when the subprogram returns.

My understanding of Ada semantic principles is that any object that is initialized should also be finalized. Since the objects left in a collection associated with a local access type become inaccessible when the scope of the access type is left, finalizing them automatically, even without an explicit free, makes sense to me. If you disagree, suggest a change to the ARG and see if you can find supporters.

Has this feature of Ada caused you real problems in real applications, or is it only a point of principle for you?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 19 Sep 2021 13:41:00 +0200

> Local variables declared in a subprogram are also not explicitly freed (deallocated), yet they are automatically finalized when the subprogram returns.

Local objects are certainly freed. Explicit or not, aggregated or not, is irrelevant.

> My understanding of Ada semantic principles is that any object that is initialized should also be finalized.

IFF deallocated.

An application that runs continuously will never deallocate, HENCE finalize certain objects.

> Has this feature of Ada caused you real problems in real applications, or is it only a point of principle for you?

1. It is a massive overhead in both memory and performance terms with no purpose whatsoever. I fail to see where that sort of thing might be even marginally useful. Specialized pools, e.g. mark-and-release will deploy their own bookkeeping, not rely on this.

2. What is worse is that a collection is not bound to the pool. It is to an access type, which may have a narrower scope. So the user could declare an unfortunate access type, which would corrupt objects in the pool and the pool designer has no means to prevent that.

From: Jere <jhb.chat@gmail.com>
Date: Sun, 19 Sep 2021 17:31:26 -0700

> Not sure what you are expecting. There is no requirement that objects are allocated contiguously. Indeed, Janus/Ada will call Allocate as many times as needed for each object; for instance, unconstrained arrays are in two parts (descriptor and data area).

Followup question cause Randy's statement got me thinking: If a compiler is allowed to break up an allocation into multiple calls to Allocate (and of course Deallocate), how does one go about enforcing that the user's header is only created once In the example Randy gave (unconstrained arrays), in Janus there is an allocation for the descriptor and a separate allocation for the data. If I am making a storage pool that is intending to create a hidden header for my objects, this means in Janus Ada (and potentially other compilers) I would instead create two headers, one for the descriptor and one for the data, when I might intend to have one header for the entire object.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 20 Sep 2021 09:34:47 +0300

I think one cannot enforce that, because the calls to Allocate do not indicate (with parameters) which set of calls concern the same object allocation.

This is probably why Dmitry said that such compiler behaviour would "disqualify the compiler" for his uses.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 19 Sep 2021 23:48:20 -0700

> I think one cannot enforce that, because the calls to Allocate do not indicate (with parameters) which set of calls concern the same object allocation.

I think the only solution would be for this compiler to have another attribute similar to 'Storage_Pool, but that would define the pool for the descriptor:

```
for X'Storage_Pool use Pool;
for X'Descriptor_Storage_Pool use
Other_Pool;
```

That way the user can decide when to add (or not) extra headers.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 20 Sep 2021 10:05:14 +0300

> Local objects are certainly freed. Explicit or not, aggregated or not, is irrelevant.

Objects left over in a local collection may certainly be freed automatically, if the implementation has created a local pool for them. See ARM 13.11 (2.a): "Alternatively, [the implementation] might choose to create a new pool at each accessibility level, which might mean that storage is reclaimed for an access type when leaving the appropriate scope."

> An application that runs continuously will never deallocate, HENCE finalize certain objects.

And I agreed that in this case it could be nice to let the programmer specify that keeping collections is not needed.

> 1. It is a massive overhead in both memory and performance terms with no purpose whatsoever. [...]

Have you actually measured or observed that overhead in some application?

> 2. What is worse is that a collection is not bound to the pool. It is to an access type, which may have a narrower scope. So the user could declare an unfortunate access type, which would corrupt objects in the pool and the pool designer has no means to prevent that.

So there is a possibility of programmer mistake, leading to unintended finalization of those (now inaccessible) objects.

However, your semantic argument (as opposed to the overhead argument) seems to be based on an assumption that the objects "left over" in a local collection, and which thus are inaccessible, will still, somehow, participate in the later execution of the program, which is why you say that finalizing those objects would "corrupt" them.

It seems to me that such continued participation is possible only if the objects contain tasks or are accessed through some kind of unchecked programming. Do you agree?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 20 Sep 2021 09:35:35 +0200

> Objects left over in a local collection may certainly be freed automatically, if the implementation has created a local pool for them. See ARM 13.11 (2.a): "Alternatively, [the implementation] might choose to create a new pool at each accessibility level, which might mean that storage is reclaimed for an access type when leaving the appropriate scope."

May or may not. The feature which correctness depends on scopes and lots of further assumptions has no place in a language like Ada.

> Have you actually measured or observed that overhead in some application?

How?

However you could estimate it from the most likely implementation as a doubly-linked list. You add new element for each allocation and remove one for each deallocation. The elements are allocated in the same pool or in some other pool. Finalization is not time bounded, BTW. Nice?

> It seems to me that such continued participation is possible only if the objects contain tasks or are accessed through some kind of unchecked programming. Do you agree?

No. You can have them accessible over other access types with wider scopes:

```
Collection_Pointer := new X;
Global_Pointer :=
Collection_Pointer.all'Unchecked_Access;
```

access discriminants etc. As I said, hands off!

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Mon, 20 Sep 2021 11:08:52 +0300

> Global_Pointer := Collection_Pointer.all'Unchecked_Access;

So, unchecked programming, as I said.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 20 Sep 2021 10:28:28 +0200

> So, unchecked programming, as I said.

Right, working with pools is all that thing. Maybe "new" should be named "unchecked_new" (:-))

Finalize and Initialize certainly should have been Unchecked_Finalize and Unchecked_Initialize as they are not enforced. You can override the parent's Initialize and never call it. It is a plain primitive operation anybody can call any time any place. You can even call it before the object is fully initialized!

So, why bother with objects the user manually allocates (and forgets to free)?

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 18:48:02 -0500

> The problem with unconstrained arrays is not that the bounds are not allocated, they are, but the semantics of X'Address when applied to arrays.

> A'Address is the address of the first array element, not of the array object. For a pool designer it constitutes a problem of getting the array object by address. This is what Emmanuel discusses in the blog.

Right, this is why Janus/Ada never "fixed" 'Address to follow the Ada requirement. (Our Ada 83 compiler treats the "object" as whatever the top-level piece is, for an unconstrained array, that's the bounds -- the data lies elsewhere and is separately allocated -- something that follows from slice semantics.)

I suppose your suggestion of implementing yet-another-attribute is probably the right way to go, and then finding every use of 'Address in existing RRS Janus/Ada code and changing it to use the new attribute that works "right".

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 18:51:15 -0500

Sorry about that, I didn't understand what you were asking. And I get defensive about people who think that a pool should get some specific Size (and only that size), so I leapt to a conclusion and answered accordingly.

The compiler requests all of the memory IT needs, but if the pool needs some additional memory for its purposes (pretty common), it will need to add that space itself. It's hard to imagine how it could be otherwise, I guess I would have thought that goes without saying. (And that rather proves that there is nothing that goes without saying.)

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 18:58:34 -0500

> But you cannot assume that the object is allocated as one big chunk. Bounds can be allocated at a different place. What would be X'Object_Address in that case?

The address of the real object, which is the bounds. (I'm using "object" in the Janus/Ada compiler sense and not in the Ada sense.) The only way I could make sense of the implementation requirements for Janus/Ada was to have every object be statically sized. If the Ada object is *not* statically sized, then the Janus/Ada object is a descriptor that provides access to that Ada object data.

Generally, one wants access to the statically sized object, as that provides access to everything else (there may be multiple levels if discriminant-dependent arrays are involved). That's not what 'Address is supposed to provide, so the address used internally to the compiler is the wrong answer in Ada terms, but it is the right answer for most uses (storage pools being an obvious example).

When one specifies 'Address in Janus/Ada, you are specifying the address of the statically allocated data. The rest of the object lives in some storage pool and it makes absolutely no sense to try to force that somewhere.

There's no sensible reason to expect 'Address to be implementation-independent; specifying the address of unconstrained arrays is nonsense unless you know that the same Ada compiler is creating the object you are accessing -- hardly a common case.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:19:38 -0500

> I don't care about techniques to implement meaningless stuff. It should be out, at least there must be a representation aspect for turning this mess off.

There is: Restriction No_Controlled_Types. - Randy

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:26:19 -0500

> Such implementations would automatically disqualify the compiler. Compiler-generated piecewise allocation is OK for the stack, not for user storage pools.

If someone wants to require contiguous allocation of objects, there should be a representation attribute to specify it. And there should not be nonsense restrictions on records with defaulted discriminants unless you specify that you require contiguous allocation. There is no good reason to need that for 99% of objects, why insist on a very expensive implementation of slices/unconstrained arrays unless it's required??

> No, it is about the overhead of maintaining "collections" associated with an access type in order to call Finalization for all members of the collection.

How else would you ensure that Finalize is always called on an allocated object? Unchecked_Deallocation need not be called on an allocated object. The Ada model is that Finalize will ALWAYS be called on every controlled object before the program ends; there are no "leaks" of finalization. Otherwise, one cannot depend on finalization for anything important; you would often leak resources (especially for simple kernels that don't try to free unreleased resources themselves).

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:37:56 -0500

> 1. It is a massive overhead in both memory and performance terms with no purpose whatsoever. I fail to see where that sort of thing might be even marginally useful.

The classic example of Finalization is file management on a simple kernel (I use CP/M as the example in my head). CP/M did not try to recover any resources on program exit, it was the program's responsibility to recover them all (or reboot after every run). If you had holes in finalization, you would easily leak files and since you could only open a limited number of them at a time, you could easily make a system non-responsive.

You get similar things on some embedded kernels.

If you only write programs that live in ROM and never, ever terminate, then you probably have different requirements. Most likely, you shouldn't be using Finalization at all (or at least not putting such objects in allocated things).

> 2. What is worse is that a collection is not bound to the pool. It is to an access type, which may have a narrower

> scope. So the user could declare an unfortunate access type, which would corrupt objects in the pool and the pool designer has no means to prevent that.

Pools are extremely low-level things that cannot be safe in any sense of the word. A badly designed pool will corrupt everything. Using a pool with the "wrong" access type generally has to be programmed for (as I answered earlier, if I assume anything about allocations, I check for violations and do something else.) And a pool can be used with many access types; many useful ones are.

Some of what you want is provided by the subpool mechanism, but it is even more complex at runtime, so it probably doesn't help you.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Sep 2021 19:45:28 -0500

> You can override the parent's Initialize and never call it. It is a plain primitive operation anybody can call any time any place. You can even call it before the object is fully initialized!

User calls on Initialize and Finalize have no special meaning; they're ignored for the purposes of language-defined finalization. The fact that they're normal routines and can be called by someone else means that some defensive programming is needed. That all happened because of the "scope reduction" of Ada 9x; the dedicated creation/finalization mechanism got dumped. Finalization was too important to lose completely, so Tucker cooked up the current much simpler mechanism in order to avoid the objections. It's not ideal for that reason -- but Finalize would still have been a normal subprogram that anyone could call (what else could it have been -- the mechanism of stream attributes could have been used instead). I don't think there is a way that one could have prevented user-defined calls to these routines (even if they had a special name, you still could have renamed an existing subprogram to the special name).

From: Simon Wright <simon@pushface.org>
Date: Tue, 28 Sep 2021 08:52:31 +0100

>> Deallocation is irrelevant. Finalization is called when objects are about to be destroyed, by any method.

> And no object may be destroyed unless deallocated.

Well, if it's important that an allocated object not be destroyed, don't allocate it from a storage pool that can go out of scope!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 28 Sep 2021 10:07:52 +0200

> Well, if it's important that an allocated object not be destroyed, don't allocate it from a storage pool that can go out of scope!

That was never the case.

The case is that an object allocated in a pool gets finalized because the access type (not the pool!) used to allocate the object goes out of the scope.

This makes no sense whatsoever.

Again, finalization must be tied with [logical] deallocation. Just like initialization is with allocation.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 28 Sep 2021 17:04:05 -0500

> Again, finalization must be tied with [logical] deallocation. Just like initialization is with allocation.

But it is. All of the objects allocated from an access type are logically deallocated when the access type goes out of scope (and the memory can be recovered). Remember that Ada was designed so that one never needs to use Unchecked_Deallocation.

I could see an unsafe language (like C) doing the sort of thing you suggest, but not Ada. Every object in Ada has a specific declaration point, initialization point, finalization point, and destruction point. There are no exceptions.

Code Flow Control

From: Kevin Chadwick <kevc3no4@gmail.com>
Subject: Code flow control
Date: Fri, 15 Oct 2021 08:08:42 -0700
Newsgroups: comp.lang.ada

Although surprised that pragma No_Exception_Propagation seems to prevent access to some exception information, I am happy with Ada's exception mechanism. I have read that exceptions should not be used for code flow.

For Ada after perusing various threads on this mailing list around best practice I am considering using exceptions locally but also have an in out variable for code flow control at the point of use. Is that the way with the caveat that it all depends on the task at hand?

In Go with vscode a static checker will warn if an error type variable is returned without a following if error utilisation (check usually of the form if err /= nil).

I have read that Spark has some kind of static analysis to achieve something similar as it forbids exceptions.

It is not the end of the world but is there any static analyser that could do similar for Ada. IOW save me some time or perhaps worse whenever I have simply omitted the check by accident, in haste or distraction.

I'm sure I could quickly write a shell script easily enough for a specific design as in if keyword appears once but not again within x lines then shout at me but I am wondering if I am missing a tool or better practice before I do so?

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 15 Oct 2021 19:48:06 +0200

> I have read that exceptions should not be used for code flow.

Some people reserve exceptions for signalling errors. I regard them as a way to handle "exceptional" situations, i.e. when the normal flow of control cannot continue. For example, in a deep recursive search, they are handy to stop the recursion and go back to top level when you have found what you were looking for. Some would disagree with that.

> I am considering using exceptions locally but also have an in out variable for code flow control at the point of use.

I definitely would prefer an exception, on the ground that you can omit the check, but you cannot ignore an exception.

> In Go with vscode a static checker will warn if an error type variable is returned without a following if error utilisation (check usually of the form if err /= nil).

An interesting idea for AdaControl, especially if you have some funding for it ;-)

From: Jeffrey R. Carter
Date: Fri, 15 Oct 2021 19:53:06 +0200

What you're talking about is result codes. Exceptions exist because of problems people encountered with result codes. Using result codes when you have exceptions is like using conditional go-tos when you have if statements.

So, yes, there is better practice. It's called exceptions.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 15 Oct 2021 20:03:16 +0200

[...]

What I do wish is carefully designed exception contracts in Ada.

Named vs Anonymous Pointer Types

From: Simon Belmont <sbelmont700@gmail.com>
Subject: Is this legal?
Date: Sat, 16 Oct 2021 12:00:18 -0700
Newsgroups: comp.lang.ada

I'm trying to learn the 2012 changes to accessibility rules, e.g. aliased parameters, additional dynamics checks, and some

eliminated unnecessary typecasts. But I am also aware of the… fluid nature of GNATs correctness of implementing them, and the following situation seems dubious. In particular, when 'current' is an anonymous access type, it compiles without issue, but not when it's a named access type (or when explicitly converted to one). Does anyone know off hand which is the correct behavior?

Thanks

-sb

```
procedure Main is

   subtype str5 is string(1..5);
   type s5_ptr is access all str5;

   type T is
      record
         current : access str5;
         --current : s5_ptr; -- "aliased actual has
                            -- wrong accessibility"
         foo : aliased str5;
      end record;

   function F (y : aliased in out str5)
   return access str5 is
   begin
      return y'Access;
   end F;


   procedure P (x : in out T) is
   begin
      x.current := F(x.foo);
   end P;

  o : T := (current => null,  foo => "Hello");

begin
   P(o);
end Main;
```

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sun, 17 Oct 2021 01:35:22 -0700

For information, ObjectAda v.10.2 accepts both variants (in Ada 2012 mode).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 18 Oct 2021 22:50:52 -0500

>I'm trying to learn the 2012 changes to accessibility rules [...] Does anyone know off hand which is the correct behavior?

I can assure you that no one anywhere will *ever* know "off-hand" the correct behavior. :-) It takes quite a bit of looking to be sure.

...

...

> function F (y : aliased in out str5) return access str5 is

> begin

> return y'Access;

> end F;

This is always legal (we hope :-). There should be a static (or dynamic) check on Y when F is called that Y has an appropriate lifetime for the result. (I can't grok "accessibility", either. I always think in terms of lifetimes, and then try to translate to the wording.)

> procedure P (x : in out T) is

> begin

> x.current := F(x.foo);

> end P;

This should always be statically illegal. X here has the lifetime of P (as the actual lifetime is unknown). That's not long enough regardless of how you declare Current (since it's type is necessarily outside of P). There is no special accessibility rules for anonymous access components (unlike most other cases); they always have the accessibility (think lifetime) of the enclosing type.

[...]

Read a long UTF-8 File, Incrementally

From: Marius Amado-Alves <amado.alves@gmail.com>
Subject: How to read in a (long) UTF-8 file, incrementally?
Date: Tue, 2 Nov 2021 10:42:37 -0700
Newsgroups: comp.lang.ada

As I understand it, to work with Unicode text inside the program it is better to use the Wide_Wide (UTF-32) variants of everything.

Now, Unicode files usually are in UTF-8.

One solution is to read the entire file in one gulp to a String, then convert to Wide_Wide. This solution is not memory efficient, and it may not be possible in some tasks e.g. real time processing of lines of text.

If the files has lines, I guess we can also work line by line (Text_IO). But the text may not have lines. Can be a long XML object, for example.

So it should be possible to read a single UTF-8 character, right? Which might be 1, 2, 3, or 4 bytes long, so it must be read into a String, right? Or directly to Wide_Wide. Are there such functions?

Thanks a lot.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 2 Nov 2021 19:17:58 +0100

> So it should be possible to read a single UTF-8 character, right? Which might be 1, 2, 3, or 4 bytes long, so it must be read into a String, right? Or directly to Wide_Wide. Are there such functions?

You simply read a stream of Characters into a buffer. Never ever use Wide or Wide_Wide, they are useless. Inside the buffer you must have 4 Characters ahead unless the file end is reached. Usually you read until some separator like line end.

Then you call this:

http://www.dmitry-kazakov.de/ada/strings_edit.htm#Strings_Edit.UTF8.Get

That will give you a code point and advance the cursor to the next UTF-8 character.

However, normally, no text processing task needs that. Whatever you want to do, you can accomplish it using normal String operations and normal String-based data structures like maps and tables. You need not to care about any UTF-8 character boundaries ever.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Wed, 3 Nov 2021 00:43:02 -0700

> [...] Are there such functions?

There is a special library to process Unicode text, see https://github.com/AdaCore/VSS; 'contrib' directory contains VSS.Utils.File_IO package to load file into Virtual_String. However, attempting to load a whole file into the memory is usually a bad decision.

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 3 Nov 2021 08:48:58 +0000

> As I understand it, to work with Unicode text inside the program it is better to use the Wide_Wide (UTF-32) variants of everything.

You can take a look at my simple lib: https://github.com/Lucretia/uca

It can read into a large string buffer. And can break it up into lines. There's no Unicode consistency checks.

The lib is a bit hacky, but seems to work for now. There's nothing more than what I've mentioned so far.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Thu, 4 Nov 2021 04:43:22 -0700

Great libraries, thanks.

It still seems to me that Wide_Wide_Character is useful. It allows to represent the character directly in the source code e.g.

```
if C = '±' then ...
```

And Wide_Wide_Character'Pos should give the codepoint.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 4 Nov 2021 13:13:12 +0100

> if C = '±' then ...

If the source supports Unicode, it should do UTF-8 as well. So, you would write

if C = "±" then ...

where C is String.

> And Wide_Wide_Character'Pos should give the codepoint.

Yes, but you need no Wide_Wide to get an integer value and if your objective is Unicode categorization, that is too complicated for manual comparisons. Use a library function [generated from UnicodeData.txt] instead.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 4 Nov 2021 14:30:25 +0000

> And Wide_Wide_Character'Pos should give the codepoint.

Characters no longer exist as a thing as one can even be represented as multiple utf-32 code points.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Fri, 5 Nov 2021 03:56:42 -0700

You're alluding to combining characters?

From: Simon Wright <simon@pushface.org>
Date: Fri, 05 Nov 2021 19:55:33 +0000

> You're alluding to combining characters?

Fun & games on macOS[1]:

> $ GNAT_FILE_NAME_CASE_SENSITIVE=1 gnatmake -c p*.ads

> gcc -c pack3.ads

> pack3.ads:1:10: warning: file name does not match unit name, should be "pack3.ads"

>

> The reason for this apparently-bizarre message is that macOS takes the composed form (lowercase a acute) and converts it under the hood to what HFS+ insists on, the fully decomposed form (lowercase a, combining acute); thus the names are actually different even though they _look_ the same.

[1] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=81114#c1

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Tue, 16 Nov 2021 03:55:05 -0800

I'm worried. I need the concept of character, for proper text processing. For example, I want to reference characters in a text file by their position. Any tips/references on how to deal with combining characters, or any other perturbing feature of Unicode, greatly appreciated.

(For me, a combining character is not a character, the combination is. Unicode agrees, right?)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 16 Nov 2021 13:36:00 +0100

> I'm worried. I need the concept of character, for proper text processing.

Simply ignore or reject decomposed characters.

> For example, I want to reference characters in a text file by their position.

That is no problem either. There are two alternatives:

1. Fixed font representation. Reduce everything to normal glyphs, use string position corresponding to the beginning of an UTF-8 sequence.
2. Proportional font. Use a graphical user interface like GTK. The GTK text buffer has a data type (iterator) to indicate a place in the buffer, e.g. when a selection happens. These iterators are consistent with the glyphs as rendered on the screen and you can convert between them and string position.

> (For me, a combining character is not a character, the combination is. Unicode agrees, right?)

No, Unicode disagrees, e.g. É can be composed from E and acute accent. But it is advised just to ignore all this nonsense and consider:

code point = character

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Tue, 16 Nov 2021 05:52:59 -0800

> Simply ignore or reject decomposed characters.

Brilliant!

> 1. Fixed font representation. Reduce everything to normal glyphs, use string position corresponding to the beginning of an UTF-8 sequence.

I am indeed resorting to byte position in UTF-8 files as the character position. Treating UTF-8 entities as the strings that they are:-)

(Not dealing with fonts nor graphics yet, just plain text.)

Thanks a lot.

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 16 Nov 2021 15:25:10 +0000

> [...] I want to reference characters in a text file by their position. [...]

You can't. The concept of character is dead, the new concept are grapheme clusters.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Tue, 16 Nov 2021 09:38:13 -0800

> (For me, a combining character is not a character, the combination is. Unicode agrees, right?)

You can use VSS and Grapheme_Cluster_Iterator to lookup for grapheme cluster at given position and to obtain position of the grapheme cluster in the string (as well as UTF-8/UTF-16 code units).

However, the concept of grapheme clusters doesn't handle special cases like tabulation stops; TAB is just a single grapheme cluster.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 16 Nov 2021 14:23:28 -0600

> Simply ignore or reject decomposed characters.

Unicode calls that "requiring Normalization Form C". ("Form D" is all decomposed characters.) You'll note that what Ada compilers do with text not in Normalization Form C is implementation-defined; in particular, a compiler could reject such text.

My understanding is that various Internet standards also require Normalization Form C. For instance, web pages are supposed to always be in that format. Whether browsers actually enforce that is unknown (they should enforce a lot of stuff about web pages, but generally just try to muddle through, which causes all kinds of security issues).

Happy Birthday, Ada!

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Subject: Happy Birthday, Ada!
Date: Fri, 10 Dec 2021 10:37:41 +0100
Newsgroups: comp.lang.ada

Born this day in 1815 and 1980. New version on this date in 2012.

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 10 Dec 2021 11:30:11 +0000

Are new revisions always released on this date?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 10 Dec 2021 13:36:32 +0100

No. 83, 95, and ISO/IEC 8652:2007 were not published on Dec 10. Probably 2X won't be, either.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Fri, 10 Dec 2021 06:23:44 -0800

A list of dates Ada 93 .. Ada 2012 can be found here:
https://www.ada-deutschland.de/sites/default/files/AdaTourCD/AdaTourCD2004/Ada%20Magica/Contents.html

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 10 Dec 2021 23:28:05 -0600

> Are new revisions always released on this date?

No. It was mostly a coincidence. We were told that the Ada 2012 Standard would be published in the December 15th batch of Standards. Joyce Tokar made a request to date it December 10th in honor of Ada's birthday; she got a response that ISO couldn't do that. When it actually was published (around December 15th), we found they had dated it December 10th as requested. Not sure when/where/how they changed their mind.

If it had been scheduled for, say, February 15th, such a request wouldn't make any sense. We wouldn't want to delay the Standard for months just so it can have a cool date.

Big_Integer Has a Limit of 300 Digits?

From: Michael Ferguson <michaelblakeferguson@gmail.com>
Subject: Ada.Numerics.Big_Numbers.Big_Integer has a limit of 300 digits?
Date: Tue, 21 Dec 2021 21:57:08 -0800
Newsgroups: comp.lang.ada

I just started using the Big_Integer library that is a part of the 202X version of Ada.

It is repeatedly described as an "arbitrary precision library" that has user defined implementation.

I was under the impression that this library would be able to infinitely calculate numbers of any length, but there is clearly a default limit of 300 digits.

Is there any way to get rid of this problem?

[Example code omitted, as it is not referenced in later discussion. —arm]

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Date: Wed, 22 Dec 2021 00:25:26 -0800

How did you determine the limit of 300 digits? I see nothing in your example, that would imply such a limit. Are you sure that you are not reaching a line length limit in Text_IO or maybe a limit in the Image attribute?

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Dec 2021 03:14:12 -0800

There is a limit

Bignum_Limit : constant := 200;

in System.Generic_Bignums body, function Normalize, lines 1100ff.

I do not see an implementation advice, implementation permission or implementation requirement about such limits in A.5.5, A.5.6.

But there is somewhere in the RM a paragraph stating that implementation may pose limits on certain features. I just cannot find the place in the RM.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Dec 2021 03:32:23 -0800

See RM 1.1.3(2).

I guess this limit is just a transient arbitrary limit until Ada 2022 is standardized.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 22 Dec 2021 08:04:29 -0800

> Bignum_Limit : constant := 200;

RM 2.2(14) limits the line length and the length of lexical elements to 200.

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 22 Dec 2021 17:01:56 +0000

What are you doing that requires that number of digits?

From: Michael Ferguson <michaelblakeferguson@gmail.com>
Date: Wed, 22 Dec 2021 09:27:56 -0800

> What are you doing that requires that number of digits?

I am working on ProjectEuler.net problem number 48.

The questions asks you to sum the numbers n^n for (2 <= n <= 1000) and determine what the last ten digits of this number are.

Obviously, this is quite a trivial problem when using any arbitrary precision library.

I had incorrectly determined that 700^700 had 300 digits, in fact 700^700 = 3.7E1991.

However, my code strictly breaks when the loop range is set to 683, which 683^683 = 8.12E1935.

So, it is interesting that the Big_Integer type works with numbers of just under 2000 digit length despite Bignum_Limit : constant := 200.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 22 Dec 2021 19:37:45 +0200

>> Bignum_Limit : constant := 200;

> RM 2.2(14) limits the line length and the length of lexical elements to 200.

To express it more clearly, RM 2.2(14) requires implementations to support lines and lexical elements of /at least/ 200 characters, but /allows/ implementations to support longer lines and lexical elements.

I'm not sure if GNAT supports more than 200 characters, though. And of course an Ada program that uses more than 200 characters may not be portable to compilers that support only 200.

But I don't see any direct logical connection to the number of digits that Big_Integers can support. While one cannot write a big-number literal longer than a line or longer than the maximum length of a lexical element, that should not directly limit the size of big-number values in computations.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 22 Dec 2021 19:48:42 +0200

> I was under the impression that this library would be able to infinitely calculate numbers of any length,

I have the same impression (up to Storage_Error, of course).

How does it break? Some exception, or something else?

Mark Lorenzen suggested in an earlier post that the limit might be in the Big_Integer'Image function. The package Ada.Numerics.Big_Numbers.Big_Integers has some other output operations that you could try:

```
function To_String
(Arg : Valid_Big_Integer; ...) return String;
procedure Put_Image
(Buffer : ... ; Arg: in Valid_Big_Integer);
```

Of course those might be internally linked to the 'Image function and have the same possible limitation.

From: Michael Ferguson <michaelblakeferguson@gmail.com>
Date: Wed, 22 Dec 2021 10:02:41 -0800

[...]

Niklas also gave me an epiphany as the exact error my program gives for the upper limit is

raised STORAGE_ERROR : Ada.Numerics.Big_Numbers.Big_Integers.Bignums.Normalize: big integer limit exceeded

I had thought that since the end of this error said big integer limit exceeded it was a problem with the library, but now I'm starting to think I need to get GNAT to allocated more memory for the program.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Wed, 22 Dec 2021 17:43:46 +0000

Does Ada's Big_Integer type work with modular ranged types?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Wed, 22 Dec 2021 21:05:18 +0200

[...]

Modular types are not connected to Big_Integers, except that the particular problem you are trying to solve could be computed "mod 10**10" because it asks for only the last 10 digits. However, the Big_Integers package does not directly support computations "mod" something (perhaps this should be an extension in a later Ada standard, because such computations are quite common).

Using "mod 10**10" operations in solving the problem would limit the number of digits in all intermediate results drastically.

> Niklas also gave me an epiphany as the exact error my program gives for the upper limit is

>
Ada.Numerics.Big_Numbers.Big_Integers.Bignums.Normalize: big integer limit exceeded

[...] the very specific exception message ("big integer limit exceeded") suggests that this exception is not a typical Storage_Error (say, heap or stack exhaustion) but may indeed stem from exceeding some specific limit in the current Big_Integer implementation in GNAT.

The size of your problem, with only a few thousand digits, suggests that heap exhaustion is unlikely to happen. However, if the Big_Integer computations are internally recursive, and use stack-allocated local variables, stack overflow could happen, so the first thing to try would be to increase the stack size. Unfortunately, for the main subprogram that has to be done with some compiler or linker options which I don't recall now. (We should really extend pragma Storage_Size to apply also to the environment task, by specifying it for the main subprogram!)

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 22 Dec 2021 12:31:32 -0800

> I am working on ProjectEuler.net problem number 48. ...

The thing about Euler problems is they usually want you to figure out a clever math trick to get to the solution, rather than using brute calculation. In the case of this problem, you want to reduce all the intermediate results mod 1e10 (which fits in an int64 easily, though not quite in an int32). That gets rid of the need for bignums.

From: Simon Wright <simon@pushface.org>
Date: Wed, 22 Dec 2021 20:34:20 +0000

> There is a limit

> Bignum_Limit : constant := 200;

> in System.Generic_Bignums body, function Normalize, lines 1100ff.

This is the maximum length of a Digit_Vector, where

```
subtype SD is Interfaces.Unsigned_32;
-- Single length digit
type Digit_Vector is array
     (Length range <>) of SD;
-- Represent digits of a number
   (most significant digit first)
```

I think this should give a maximum value of ~10**2000.

I printed out sum'image'length; the last value before the exception was 1937.

From: Luke A. Guest <laguest@archeia.com>
Date: Thu, 23 Dec 2021 08:31:11 +0000

Is mod overloadable?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 23 Dec 2021 09:54:34 +0100

It is as any operator.

P.S. For large numbers one needs rather full division than separate /, mod, rem.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Thu, 23 Dec 2021 03:41:13 -0800

> However, the Big_Integers package does not directly support computations "mod"

It does A.5.6(18/5).

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 23 Dec 2021 14:18:34 +0200

Yes, there is a "mod" operator for Big_Integer. My point was that there are no Big_Integer operations, such as multiplication, that are intrinsically modular in the same way as the operations for modular types are. So the only way to perform a modular multiplication of Big_Integers is to first multiply the numbers in the usual way, producing a possibly very large product, and then apply "mod" to reduce that product.

In my imperfect understanding, intrinsically modular big-number computations can be much more efficient than such post-computation applications of "mod", at least if the modulus is not itself a big number.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Thu, 23 Dec 2021 14:01:47 +0000

Yes, there are efficient algorithms for "x * y mod n" so almost all "big num" libraries provide a function to do it. Ada has the type system for the mod operation to be explicit in the type.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 23 Dec 2021 16:48:17 +0100

As it appears from the rest of the discussion that there is a limit in the implementation of the pkg, you could try using PragmARC.Unbounded_Numbers.Integers

https://github.com/jrcarter/PragmARC/blob/Ada-12/pragmarc-unbounded_numbers-integers.ad

where the implementation restricts a number to Integer'Last "digits" or the available heap memory, whichever is less.

Note that with recent versions of GNAT for 64-bit processors, the "digits" will be 64 bits.

Pure vs Preelaborate

From: Simon Belmont <sbelmont700@gmail.com>
Subject: Re: Ada Pure or Preelaborate or ? in Adare_net
Date: Mon, 3 Jan 2022 17:11:36 -0800
Newsgroups: comp.lang.ada

> I and a friend created an Ada network lib where, from the begining, we tried very hard to make It a Ada Pure.

> From the examples dir, the lib worked as expected (in gcc-10.2 gcc-11.2 and gcc-12). To our surprise, what most caught the attention of the group's friends was the fact that the lib was Ada Pure and if that was correct.

> For this reason, if really 'is' pure, not pure, preelaborate or what (?), pleeeeeeaaase, we ask the group's Ada Language Lawyers to help analyze and suggest modifications if necessary.

> link: https://gitlab.com/daresoft/network/adare_net/-/tree/202x

> for Ada version use 2012 and or 202x.

It seems to be mostly just a thin binding to a bunch of C functions, so the applicability of any Ada feature is mostly a moot point. The Ada compiler has no control or visibility into the C domain, so while on the one hand your packages are technically Pure, on the other hand the C functions can violate those "purity rules" all they want, which might be misleading to users expecting otherwise. You don't use 'Unchecked_Access either, but obviously that doesn't mean the C functions are somehow prevented from creating dangling pointers. Personally, I would have the interfaces reflect the reality of the actual behavior (which in the case of C code you don't control, is usually assume-the-worst).

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Tue, 4 Jan 2022 05:52:41 -0800

> For this reason, if really 'is' pure, not pure, preelaborate or what (?)

I recommend reading this: https://stackoverflow.com/questions/19353228/when-to-use-pragma-pure-preelaborate

If your units are declared as Pure, the compiler considers that they have no side effects and can decide to call the sub-programs only once and cache the result, or even not call the sub-program if the result is not used after.

From: Daniel Norte Moraes <danielcheagle@gmail.com>
Date: Wed, 5 Jan 2022 08:11:52 -0800

> It seems to be mostly just a thin binding to a bunch of C functions, so the applicability of any Ada feature is mostly a moot point. [...]

The C pointers are only created in c_initialize_socket.c (c_init_address()) and data pointed copied to an Ada array, and then immediately free by c part. This is the only time there is a dynamic allocation.

We managed to make libadare_net very close to 100% static allocation!

Because of this, there aren't dangling pointers.

There is still the problem of omitting the execution of subprograms by the compiler by pure packages.

Would 'preelaborate' solve this?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 5 Jan 2022 17:40:04 -0600

Yes. The permission to omit calls only applies to Pure (see 10.2.1(18/3)). http://www.ada-auth.org/standards/2xaarm/html/AA-10-2-1.html#p18. (I gave a reference to the Ada 2022 AARM, but this rule hasn't changed in spirit since it was introduced in Ada 95.)

From: Daniel Norte Moraes <danielcheagle@gmail.com>
Date: Thu, 6 Jan 2022 12:39:21 -0800

> Yes. The permission to omit calls only applies to Pure (see 10.2.1(18/3)).

Thanks!

We will change the packages in LibAdare_Net to 'preelaborate'. :-) and continue from here.

Conference Calendar

Dirk Craeynest

KU Leuven, Belgium. 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.

The COVID-19 pandemic had a catastrophic impact on conferences world-wide. Where available, the status of events is indicated with the following markers: "(v)" = event is held online, "(h)"= event is held in a hybrid form (i.e. partially online).

2021

April 02-06 (v)
IEEE/ACM **International Symposium on Code Generation and Optimization** (CGO'2022), Seoul, South Korea. Co-located with PPoPP, CC and HPCA. Topics include: a wide range of optimization and code generation techniques and related issues at the interface of hardware and software; from purely static to fully dynamic approaches, from pure software-based methods to specific architectural features and support for code generation and optimization.

☺ April 02-06 (v)
27th ACM SIGPLAN **Symposium on Principles and Practice of Parallel Programming** (PPoPP'2022), Seoul, South Korea. PPoPP'2022 was postponed from 12-16 February to 2-6 April. Topics include: all aspects of parallel programming, including theoretical foundations, techniques, languages, compilers, runtime systems, tools, and practical experience; techniques and tools that improve the productivity of parallel programming.

April 02-07
25th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2022). Munich, Germany. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems).

☺ April 02-03 **VerifyThis Verification Competition 2022**. Topics include: no restrictions on programming language and verification technology used.

April 04-13 (v)
15th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2022). Valencia, Spain.

April 05-06 (v)
31st ACM SIGPLAN **International Conference on Compiler Construction** (CC'2022), Seoul, South Korea. Co-located with CGO, HPCA, and PPoPP. Topics include: processing programs in the most general sense (analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques (including program representation, analysis, and transformation; code generation, optimization, and synthesis; the verification thereof); run-time techniques (including memory management, virtual machines, and dynamic and just-in-time compilation); programming tools (including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers); techniques, ranging from programming languages to micro-architectural support, for specific domains such as secure, parallel, distributed, embedded or mobile environments; design and implementation of novel language constructs, programming models, and domain-specific languages.

April 09-13
13th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2022). Beijing, China.

April 23 (h)
2nd **International Conference on Code Quality** (ICCQ'2022). Innopolis, Kazan, Russia. Topics include: static analysis, program verification, bug detection, and software maintenance.

April 25-29 (v)
37th ACM **Symposium on Applied Computing** (SAC'2022). Brno, Czech Republic.

April 25-29 (v) 17th **Track on Dependable, Adaptive, and Secure Distributed Systems** (DADS'2022). Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.

April 25-29 (v) **Embedded Systems Track** (EMBS'2022). Topics include: the application of both novel and well-known techniques to the embedded systems development.

April 25-29 (v) **Track on Software Engineering** (SE'2022)**.** Topics include: developing high-quality software applications more effectively and efficiently; architecture, framework, and design patterns; standards; software maintenance and evolution; software testing and verification; mining software repositories; quality assurance; verification, validation, testing, and analysis; safety, security, privacy, and risk management; dependability and reliability; fault tolerance and availability; formal methods and theories; component-based development and reuse; empirical studies and industrial best practices; applications and tools; etc.

May 03-06 (v) 15th **Cyber-Physical Systems and Internet of Things Week** (CPS Week'2022). Milan, Italy. Event includes: 5 top conferences, HSCC, ICCPS, IPSN, RTAS, and IoTDI, multiple workshops, tutorials, competitions and various exhibitions from both industry and academia.

☺ May 04-06 (v) 28th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2022). Topics include: systems research related to embedded systems and time-sensitive systems; original systems, applications, case studies, methodologies, and algorithms that contribute to the state of practice in design, implementation, verification, and validation of embedded systems or time-sensitive systems.

May 04-06 (v) 13th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2022). Topics include: software platforms and systems for CPS; specification languages and requirements; design, optimization, and synthesis; testing, verification, certification; security, trust, and privacy in CPS; applications of CPS technologies; tools, testbeds, demonstrations, and deployments; etc.

☺ May 17-18 (h) 25th IEEE **International Symposium On Real-Time Distributed Computing** (ISORC'2022). Västerås, Sweden. Topics include: all aspects of object/component/service-oriented real-time distributed computing (ORC) technology, such as distributed computing, internet of things (IoT), real-time scheduling theory, resilient cyber-physical systems, autonomous systems (e.g., autonomous driving), optimization of time-sensitive applications, applications based on ORC technology (e.g., medical devices, intelligent transportation systems, industrial automation systems and industry 4.0, smart grids, ...), etc.

May 21 (v) 28th **International Symposium on Model Checking of Software** (SPIN'2022), Chicago, Illinois, USA. Topics include: automated tool-based techniques for the analysis of software as well as models of software for the purpose of verification and validation, formal specification languages, design-by-contract, model checking, verifying compilers, abstraction and symbolic execution techniques, static analysis and abstract interpretation, combination of verification techniques, modular and compositional verification techniques, combination of static and dynamic analyses, case studies of interesting systems or with interesting results, engineering and implementation of software verification and analysis tools, formal methods of education and training, etc.

May 21-29 44th **International Conference on Software Engineering** (ICSE'2022). Pittsburgh, Pennsylvania, USA. Topics include: the full spectrum of Software Engineering.

May 9 (v) 4th **International Workshop on Robotics Software Engineering** (RoSE'2022).

May 22-23 (h) 5th **International Conference on Technical Debt** (TechDebt'2022).

May 22-23 (h) 10th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2022). Topics include: approaches, methods and tools for verification and validation; correctness-by-construction approaches for software and systems engineering; application of formal methods to specific domains, e.g., autonomous, cyber-physical, intelligent, and IoT systems; scalability of formal methods applications;

integration of formal methods within the software development lifecycle; model-based software engineering approaches; formal methods in a certification context; formal approaches for safety and security; usability of formal methods; guidelines to use formal methods in practice; case studies developed/analyzed with formal approaches; experience reports on the application of formal methods to real-world problems; etc.

May 24-27 (h)
14th **NASA Formal Methods Symposium** (NFM'2022). Pasadena, California, USA. Topics include: challenges and solutions for achieving assurance for critical systems, such as advances in formal methods (interactive and automated theorem proving, model checking, static analysis, runtime verification, automated testing, design for verification and correct-by-design techniques, ...), integration of formal methods techniques, formal methods in practice (experience reports of application of formal methods on real systems, such as autonomous systems, safety-critical systems, concurrent and distributed systems, cyber-physical, embedded, and hybrid systems, ...; use of formal methods in education; reports on negative results in the development and the application for formal methods in practice; usability of formal method tools, and their infusion into industrial contexts; ...).

☺ June 01-02
International Conference on Reliability, Safety and Security of Railway Systems (RSSRail'2022). Paris, France. Topics include: building critical railway applications and systems; safety in development processes and safety management; system and software safety analysis; formal modelling and verification techniques; system reliability; validation according to the standards; tool and model integration, toolchains; domain-specific languages and modelling frameworks; model reuse for reliability, safety and security; etc.

June 4 (v)
Ada-Belgium 2022 General Assembly. Belgium. Deadline for registration: June 3, 2022, 16:00.

☺ June 06-10 (h)
36th **European Conference on Object-Oriented Programming** (ECOOP'2022). Berlin, Germany. Topics include: all practical and theoretical investigations of programming languages, systems and environments; innovative solutions to real problems as well as evaluations of existing solutions. Deadline for submissions: April 1-30, 2022 (workshop papers).

☺ June 07-08 (h)
30th **International Conference on Real-Time Networks and Systems** (RTNS'2022). Paris, France. Topics include: real-time application design and evaluation (automotive, avionics, space, railways, telecommunications, process control, ...), 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, security, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), ...), formal specification and verification, real-time distributed systems, etc.

June 07-10 (h)
17th **International Conference on integrated Formal Methods** (iFM'2022). Lugano, Switzerland. . Topics include: recent research advances in the development of integrated approaches to formal modelling and analysis; all aspects of the design of integrated techniques, including language design, verification and validation, automated tool support and the use of such techniques in software engineering practice.

June 13-15 (h)
26th **International Conference on Empirical Assessment and Evaluation in Software Engineering** (EASE'2022). Gothenburg, Sweden. Topics include: assessing the benefits/costs associated with using chosen development technologies; empirical studies using qualitative, quantitative, and mixed methods; evaluation and comparison of techniques and models; modeling, measuring, and assessing product and/or process quality; replication of empirical studies and families of studies; software technology transfer to industry; etc.

June 13-17 (v)
25th **Ibero-American Conference on Software Engineering** (CIbSE'2022). Cordoba, Argentina. Topics include: formal methods applied to software engineering (SE), mining software repositories and software analytics, model-driven SE, software architecture, software dependability, SE education and training, SE for emerging application domains (e.g., cyber-physical systems, IoT, ...), SE in industry, software maintenance and evolution, software process, software product lines, software quality and quality models, software reuse, software testing, technical debt management, etc.

June 14-17 (h)

26th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC 2022 aka Ada-Europe 2022), Ghent, Belgium. Sponsored by Ada-Europe. In cooperation with ACM SIGAda, SIGBED, SIGPLAN, the Ada Resource Association (ARA), and Ghent University. Deadline for early registration: May 20, 2022.

☺ June 17 — 7th **Workshop on Challenges and New Approaches for Dependable and Cyber-Physical System Engineering** (De-CPS'2022). Topics include: artificial intelligence for CPS; model-based system engineering; transport and mobility, vehicle of the future; Industry 4.0 / 5.0; IoT, edge and cloud Continuum; safety and (cyber)security; advanced platforms; human/machine interaction; timing sensitive networks; 5G networks. Deadline for submissions: April 29, 2022 (papers).

June 17 — **ADEPT workshop, AADL by its practitioners** (ADEPT'2022). Topics include: current projects in the field of design, implementation and verification of critical systems where AADL is a first citizen technology. Deadline for submissions: May 20, 2022 (abstracts).

June 15-17 (h) — 20th **International Conference on Software and Systems Reuse** (ICSR'2022). Montpellier, France. Theme: "Reuse and Software Quality". Topics include: new and innovative research results and industrial experience reports dealing with all aspects of software reuse within the context of the modern software development landscape, such as technical aspects of reuse (model-driven development, variability management and software product lines, domain-specific languages, new language abstractions for software reuse, software composition and modularization, ...), software reuse in industry (reuse success stories, reuse failures and lessons learned, reuse obstacles and success factors, return on Investment studies), etc. Deadline for submissions: April 15, 2022 (Industry and Tool-Demo track), April 19, 2022 (doctoral symposium).

June 20-22 — 17th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2022), Budapest, Hungary. HiPEAC'2022 was postponed from 17-19 January to 20-22 June. Topics include: computer architecture, programming models, compilers and operating systems for embedded and general-purpose systems. Deadline for early registration: May 15, 2022.

July 05-08 — **Software Technologies: Applications and Foundations** (STAF'2022), Nantes, France. Deadline for submissions: May 14, 2022 (workshop papers). Deadline for early registration: May 31, 2022.

☺ July 07-08 — 15th **International Symposium on High-Level Parallel Programming and applications** (HLPP'2022), Porto, Portugal. Topics include: high-level parallel programming, its tools and applications, such as high-level programming and tools, automatic code generation for parallel programming, model-driven software engineering with parallel programs, high-level programming models for heterogeneous/hierarchical platforms, applications of parallel systems using high-level languages and tools, formal models of timing and real-time verification for parallel systems, etc. Deadline for submissions: May 6, 2022. Deadline for early registration: June 8, 2022.

☺ August 22-26 — 28th **International European Conference on Parallel and Distributed Computing** (Euro-Par'2022). Glasgow, Scotland, UK. Topics include: all flavors of parallel and distributed processing, such as compilers, tools and environments, scheduling and load balancing, theory and algorithms for parallel and distributed processing, parallel and distributed programming, interfaces, and languages, multicore and manycore parallelism, etc. Deadline for submissions: May 20, 2022 (workshop papers).

☺ August 23-25 (h) — 28th IEEE **International Conference on Embedded and Real-Time Computing Systems and Applications** (RTCSA'2022). Taiwan. Deadline for submissions: April 8, 2022 (abstracts), April 15, 2022 (papers).

September 04-07 (h) — 17th **Federated Conference on Computer Science and Information Systems** (FedCSIS'2022). Sofia, Bulgaria. Deadline for submissions: November 21, 2022 (technical sessions), May 10, 2022 (papers), June 7, 2022 (position papers).

September 12-14 — 15th **International Conference on the Quality of Information and Communications Technology** (QUATIC'2022). Talavera de la Reina, Spain. Topics include: all quality aspects in ICT systems engineering and management; related to the specification, design, development, operation, maintenance and evolution of ITC systems; quality in ICT process, product, and applications domains; practical studies; etc.

☺ September 14-16 — 27th **International Conference on Formal Methods for Industrial Critical Systems** (FMICS'2022). Warschaw, Poland. Part of CONFEST'2022. Topics include: case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; methods, techniques and tools to support automated analysis, certification, debugging, descriptions, learning, optimisation and transformation of complex, distributed, real-time, embedded, mobile and autonomous systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues); impact of the adoption of formal methods on the development process and associated costs; application of formal methods in standardisation and industrial forums. Deadline for submissions: May 5, 2022 (abstracts), May 26, 2022 (papers).

September 19-23 — 16th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2022). Helsinki, Finland. Deadline for submissions: April 25, 2022 (abstracts of technical papers and emerging results and vision papers), May 2, 2022 (technical papers and emerging results and vision papers), June 22, 2022 (Doctoral Symposium), June 30, 2022 (Registered Reports track), July 7, 2022 (Journal-First papers), August 13, 2022 (Industry Forum).

October 03-06 (v) — 29th IEEE **Software Technology Conference** (STC'2022). Internet. Topics include: software engineering for emerging systems; software testing, testability, and assurance; cybersecurity and information assurance; agile software development; challenges and opportunities in SW & systems development processes; etc. Deadline for submissions: June 3, 2022 (abstracts, full papers).

October 07-14 (h) — **Embedded Systems Week 2022** (ESWEEK'2022), Shanghai, China. Includes CASES'2022 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2022 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2022 (International Conference on Embedded Software). Deadline for submissions: April 7, 2022 (Journal Track full papers), April 16, 2022 (workshops, tutorials, education class, brainstorming sessions, breakout sessions), April 30, 2022 (special sessions), May 27, 2022 (MEMOCODE abstracts, i.e. International Conference on Formal Methods and Models for System Design), June 1, 2022 (student competition proposals), June 3, 2022 (MEMOCODE papers), June 11, 2022 (industry papers, Industry Challenge pitches, Work-in-Progress (WiP) papers, Ph.D. Forum).

♦ October 14 — ACM SIGAda **High Integrity Language Technology International Workshop on Supporting a Rigorous Approach to Software Development** (HILT'2022), Ann Arbor, Michigan, USA. Co-located with ASE'2022. Sponsored by ACM SIGAda. Topics include: practical use of High Integrity languages, technologies, and methodologies that enable expedited design and development of software-intensive systems; practical use of formal methods at industrial scale; IDE-support for formal methods; model-level analysis tools for systems like SysML, AADL, Lustre, or Simulink; continuous integration and deployment based on advanced static analysis tools; safety-oriented programming language features; qualification of language tools for critical systems use; etc. Deadline for submissions: July 1, 2022.

November 15-17 — 24th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2022), Clermont-Ferrand, France. Topics include: concurrent and distributed computing (foundations, fault-tolerance, and security); distributed, concurrent, and fault-tolerant algorithms; synchronization protocols; formal methods, validation, verification, and synthesis; etc. Deadline for paper submissions: April 15, 2022 (1st deadline), August 5, 2022 (2nd deadline).

☺ December 05-08 — 43rd IEEE **Real-Time Systems Symposium** (RTSS'2022), Houston, Texas, USA. Topics include: addressing some form of real-time requirements such as deadlines, response times or delay/latency. Deadline for submissions: May 15, 2022 (TCRTS award nominations), May 26, 2022 (papers).

☺ December 05-10 — ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2022), Auckland, New Zealand. Deadline for submissions: May 1, 2022 (workshops), September 1, 2022 (workshop papers). Deadline for early registration: September 18, 2022.

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

2023

January 16-18 — 18th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2023), Toulouse, France. Topics include: software development for high

performance parallel systems; tools for compilation, evaluation, optimization of high performance parallel systems (compiler support, tracing, and debugging for parallel architectures, ...); embedded real-time systems, mixed criticality system support, dependable systems, ...; software support for embedded architectures (tracing and real-time analysis of embedded applications, runtime software); etc. Deadline for submissions: June 1, 2022 (workshops, tutorials).

March 06-10 — 25th **International Symposium on Formal Methods** (FM'2023), Lübeck, Germany. Deadline for submissions: June 10, 2022 (workshops), July 1, 2022 (tutorials).

April 22-27 — 26th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2023), Paris, France. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems). Deadline for submissions: May 20, 2022 (satellite events).

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

HILT'22 CALL FOR PAPERS

About the conference

This is the seventh 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 highly complex critical software systems. HILT is organized by ACM SIGAda, in cooperation with Ada-Europe.
High Integrity Language Technologies have been tackling the challenges of building efficient, safe, reliable software for decades. Critical software as a domain is quickly expanding beyond embedded real-time control applications to the increasing reliance on complex software for the basic functioning of businesses, governments, and society in general.
For its 2022 edition, HILT will be a workshop of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE'2022. The workshop will be held on October 14th 2022.

Topics

HILT 2022 will focus on the increasing synergies between formal methods (theorem provers, SAT, SMT, etc.), advanced static analysis (model checking, abstract interpretation), software design and modeling, and safety-oriented languages. From separate fields of research, we now observe a stronger interconnection between formal methods, advanced analytics, modeling and design of software, and safety features in programming languages. Programming languages for safety-critical systems now routinely integrate theorem proving capabilities like C/ACSL or Ada/SPARK2014. Theorem provers such as Coq, Lean, or Isabelle have established themselves as a viable strategy to implement compilers or properly define the semantics of domain-specific languages. Tools for verifying modeling languages such as AADL, Lustre, and Simulink are becoming more widely available, and with the emergence of the Rust language and the release of Ada 2022, safety is rising to the top of concerns for critical systems developers.

The HILT'2022 workshop seeks to explore ways High Integrity Language Technologies leverage recent advances in practical formal methods and language design to deliver the next generation of safety-critical systems.

Call for Papers and Extended Abstract

This workshop is focused on the practical use of High Integrity languages, technologies, and methodologies that enable expedited design and development of software-intensive systems.
Key areas of interest include experience and research into:

- Practical use of formal methods at industrial scale
- IDE-support for formal methods
- Model-level analysis tools for systems like SysML, AADL, Lustre, or Simulink
- Continuous Integration and Deployment based on advanced static analysis tools
- Safety-Oriented Programming Language features
- Qualification of Language Tools for critical systems use

The workshop accepts either short abstracts (2-3 pages) for presentation, or full papers (up to 8 pages). Submissions should conform, at time of submission, to the ACM Proceedings Template: https://www.acm.org/publications/proceedings-template.

The workshop proceedings will be jointly published in the ACM Ada Letters and Ada-Europe's Ada User Journal. Authors of accepted papers will be invited to contribute to a special issue of the Springer Journal on Software and Tools for Technology Transfer (STTT).

Conference Website: https://conf.researchr.org/home/hilt-2022
Submission Deadline: July, 1 2022
Notification to authors: August, 1 2022
Workshop date: October, 14th 2022

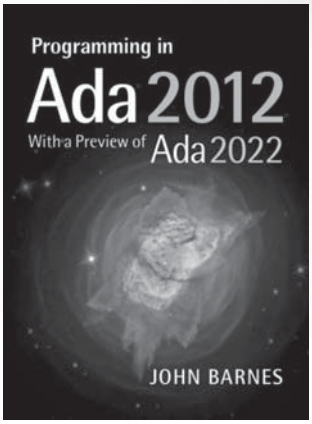
Tentative list of Program Committee members

- Robert Bocchino (NASA JPL, US)
- Claire Dross (AdaCore, FR)
- Florian Gilcher (Ferrous Systems, DE)
- John Hatcliff (KSU, US)
- Laura Humphrey (AFRL, US)
- John Kassie (Collins Aerospace, US)
- Nikolai Kosmatov (CEA List, FR)
- Nicholas Matsakis (AWS, US)
- Sara Royuela, (Barcelona SC, ES)
- Ina Schaefer, (TU Braunschweig, DE)
- Alok Srivastava, (SAIC Inc.m US)
- Cesare Tinelli, (Univ. Iowa, US)
- Joyce Tokar, (Raytheon, US)
- Uwe Zimmer (ANU, AU)

Programming in Ada 2012 with a Preview of Ada 2022

2nd Edition

John Barnes *(John Barnes Informatics)*



The latest edition of the definitive guide to the Ada language covers the full details of the core language Ada 2012 as updated by the 2016 ISO Corrigendum and introduces the key new features in Ada 2022. The book is in four parts. It begins by introducing the fundamental concepts for newcomers, before moving onto algorithmic aspects and then structural features such as OOP and multitasking. The fourth part gives details of the standard library and interaction with the external environment. Six complete executable programs illustrate the core features of the language in action. The book concludes with an appendix focussing on the new features in Ada 2022. These new features aid program proof and the efficient use of multicore architectures.

19 May 2022

244 x 188 mm 992pp

Paperback 978-1-009-18134-1

Original Price	*Discount Price*
£84.99	£63.74
$110.00	$82.50

'As he has done similarly before, in this special edition of his classic 'Programming in Ada' text, John Barnes introduces the reader to a new version of Ada, namely 'Ada 2022', with both rigour and vigor. John is unique in his ability to explain new Ada features in a way that makes them both immediately understandable and immediately usable, because of the helpful context and witty examples he provides. As a member of the group that maintains the Ada language standard, John knows the ins and outs of why the features ended up as they did, but he seems to always be the best at rising above the minutiae the typical language design team member worries about, to see the big picture and how the new feature fits into the overall language, and how it can benefit the programmers building critical applications in Ada.'
- S. Tucker Taft, *Director of Language Research, Ada Core*

11th Ada Developer Room at FOSDEM 2022

Dirk Craeynest

KU Leuven, Dept. of Computer Science, B-3001 Leuven, Belgium; email: Dirk.Craeynest@cs.kuleuven.be

Abstract

FOSDEM is a huge open source event organized each year in Brussels, Belgium. Among others, it features dozens of tracks on specific topics: the Developer Rooms. This year, for the 11th time, there was a track about the Ada programming language and related technologies. A brief overview of the full-day "Ada DevRoom" program is given, followed by reports of most presentations by their respective authors.

Keywords: Free and Open Source, FOSDEM, Ada.

1 Introduction

FOSDEM [1], the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organized early each year in Brussels. It is highly developer-oriented and brings together 8000+ participants from all over the world. It is free to attend and no registration is necessary. The 2022 edition took place on Saturday 5 and Sunday 6 February. As last year, due to the COVID-19 pandemic, it had been turned into an online event.

At FOSDEM, Ada-Belgium organized 10 Ada DevRooms in the years 2006-2020. In this edition, the Ada FOSDEM community organized once more 8 hours of presentations related to Ada and Free or Open Software. The "Ada DevRoom" at FOSDEM 2022, for the first time in an online format, was held on the 2nd day of the event, and offered introductory presentations on the Ada programming language, as well as more specialised presentations on focused topics, tools and projects: a total of 13 Ada-related presentations by 12 authors from 8 countries!

This year the coordination was done for the first time by Fernando Oleo Blanco, and done very well! The event was organized in cooperation with Ada-Belgium and Ada-Europe, and saw a good level of participation and great discussions.

2 Program overview

Each slot in the Ada DevRoom program consisted of a pre-recorded video of the presentation, played on the virtual "main stage" of the DevRoom; a Q&A session, where participants could interact live with the presenter on that "main stage"; and when the next video started, participants had the option to continue discussions with the presenter in a dedicated "chat room" for each presentation.

The Ada DevRoom featured the following program:

- *Introduction to the Ada DevRoom*, by Fernando Oleo Blanco, Germany;
- *Introduction to Ada for Beginning and Experienced Programmers*, by Jean-Pierre Rosen, France;
- *Ada Looks Good, Now Program a Game Without Knowing Anything*, by Stefan Hild, Germany;
- *The Ada Numerics Model*, by Jean-Pierre Rosen, France;
- *2022 Alire Update*, by Fabien Chouteau, France, and Alejandro Mosteo, Spain;
- *SweetAda: Lightweight Development Framework for Ada-Based Software Systems*, by Gabriele Galeotti, Italy;
- *Use (and Abuse?) of Ada 2022 Features to Design a JSON-Like Data Structure*, by Alejandro Mosteo, Spain;
- *Getting Started with AdaWebPack*, by Max Reznik, Ukraine;
- *Overview of Ada GUI*, by Jeffrey Carter, Belgium;
- *SPARKNaCl: a Verified, Fast Re-implementation of TweetNaCl*, by Roderick Chapman, UK;
- *The Outsider's Guide to Ada: Lessons from Learning Ada in 2021*, by Paul Jarrett, USA;
- *Proving the Correctness of the GNAT Light Runtime Library*, by Yannick Moy, France;
- *Implementing a Build Manager in Ada*, by Stephane Carrez, France;
- *Exporting Ada Software to Python and Julia*, by Jan Verschelde, USA;
- *Closing of the Ada DevRoom*, by Dirk Craeynest, Belgium, and Fernando Oleo Blanco, Germany.

The Ada at FOSDEM 2022 web page has all details, see:

https://fosdem.org/2022/schedule/track/ada/

It provides the full schedule, abstracts of presentations, biographies of speakers, and pointers to more info. Recordings of each presentation, i.e. the pre-recorded video as well as the Q&A session, plus copies of the slides, are available there as well.

3 Informal proceedings in the AUJ

All authors were invited to prepare short papers based on their presentations, to be included in the AUJ, and most of them accepted. The idea was to make it easier for authors and to be as inclusive as possible, hence the Journal accepted papers with variable size, from 1 to 8 pages.

The collection of papers is provided in this AUJ issue, in what we may call the informal proceedings of the Ada DevRoom at FOSDEM 2022.

[1] https://fosdem.org/2022/

Ada Looks Good, Now Program a Game Without Knowing Anything

Stefan Hild

Abstract

In 2020 I started live streaming the development of a turn-based strategy game. At that time I had little idea about Ada, programming or game development (nothing has changed about that to this day). But by September 2020 it had taken the early form of a Civilization clone. After more than a year of development, it has become almost a real game with its own features. And now I'm going to talk a little bit about some experiences and weirdnesses with game development in Ada.

Keywords: Game, Beginner, Ada.

1 Introduction

I discovered Ada in early 2019, I can't remember exactly how, but I directly liked the language, unlike several other programming languages I've looked at or tried to learn in the past. Some of the things I liked were readable syntax, usable error messages, and that not any junk just compiled through.

So I started learning the language with the help of the tutorials from Wikibooks [1] and Rosetta Code [2] and after understanding some of the basics, of course I immediately started programming a game. The result [3] is exactly on the level you would expect when someone programs an RPG with little knowledge of the programming language used or game development in general.

The time to really get more involved with Ada was at the beginning of 2020, when I also started developing a new game [4]. By September 2020 it had taken the form of an early Civilization clone and after more than a year of development it's now almost a real game with its own ideas. As I live stream almost all of the development on YouTube [5] and Twitch [6], I was "discovered" and asked to give a presentation at FOSDEM 2022 [7].

But since I don't know anything, I just talked about some basic things and problems in Ada that I noticed during my game development. Because sometimes it seems as if Ada wasn't developed primarily for game programming.

2 General

First of all I would like to clarify that all statements refer to game development with Ada 2012 under a reasonably modern computer system and there will probably be significant differences if you develop for old or embedded systems (or you just know what you are doing). So let's start with a few basic recommendations.

Ada offers various additional validity checks (-gnatVx), which should all be activated. The same applies to the additional warnings, with the exception of -gnatw.y, which only provides information about why a package needs a body, which may be useful for very complex packages. You should also have all warnings treated as errors. All of this may sound a bit exaggerated at first, but once you understand what you have to do to avoid the messages that appear as a result, it hardly takes any more time and you can identify problems much earlier. Which then makes it possible to fix corresponding errors in five minutes of work and not drag it around for a while and then take three days to adjust everything. Which of course has never happened to me.

There are also various style checks that you should take a look at. Among other things, you should think about limiting the nesting depth, just because Ada allows things like nested packages doesn't mean you should overdo it. It should also be noted that Ada guarantees a length of 200 characters for names, this should also be used for a more understandable code, there is no advantage in limiting all names to three characters.

3 String Handling

As with many programs, games need text, so string handling is an important thing. Luckily, Ada has a variety of different types of strings, but all but three variants can be practically ignored. These three types are Unbounded_Wide_Wide_String, Wide_Wide_String, and String.

Since the texts in games are often available in several languages and/or you want to enable modding by the players, the exact text length is unknown. Therefore, the simplest solution to this problem is an Unbounded_String. In addition, other languages often use characters outside of the ASCII character set, for example the German language contains the special characters ä, ö, ü and ß, a problem that can be easily solved by using the Wide_Wide_Version. Of course one could also use a different string variant and adjust the length of the string dynamically or use the UTF8 version of the standard string. Not only is this a significantly greater effort to save text, it also does not result in any significant advantages. Certainly, an Unbounded_Wide_Wide_String requires more memory than a normal string, but this is only theoretically relevant, since this consumption is completely lost in the background noise compared to other parts, for example the game world map. In addition, an Unbounded_String can also be used in records without further adjustments.

Occasionally, especially if you want to interact with the standard or external libraries, you need a string that is not Unbounded. In these cases, it is best to use Wide_Wide_String whenever possible. This way you don't have to worry about possible problems with texts containing

special characters and by using Ada.Strings.Wide_Wide_Unbounded.To_Wide_Wide_String and Ada.Strings.Wide_Wide_Unbounded.To_Unbounded_Wide_Wide_String an easy conversion option is available. Again, the theoretically saveable memory is irrelevant and it just creates a bunch of extra work.

Finally there is the normal string, which is especially important when interacting with a library that only uses string. An example here is the standard library, which with Ada.Directories has offered the possibility of reading in directories and file names since Ada 2005, but only in a string.

4 Standard library

Which brings us directly to one of the problems with the Ada standard library, Ada.Directories only uses String. Ada 2005 introduced Wide_Wide_ for complete UTF8 support and many packages have a corresponding Wide_Wide_ Variant, but not Ada.Directories. The revision in Ada 2012 hasn't changed that either. This must be taken into account when naming files and folders, and additional conversions must be built in.

Also for Ada.Float_Text_IO there is no Wide_Wide_Version and if you need one you have to create a derived package based on Ada.Wide_Wide_Text_IO yourself. Not an insurmountable obstacle, but sometimes quite inconsistent, especially since an Integer_Wide_Wide_Text_IO exists.

Another problem is that there are often no comments when naming different procedures or functions in the same way. Yes, Ada allows multiple procedures/functions with the same name in the same file, even if the transfer parameter names are the same, as long as the data types are different. However, creating 5000 procedures with the name Put and then distributing them in 300 files without any comments is not very user-friendly. When you're learning Ada and you're looking for a solution to a problem and you find the solution "just use Put for that", it's a bit difficult to find out which Put is meant. Of course, this does not only apply to Put, there are also plenty of Get, despite the possible name length of 200 characters in Ada.

5 Minor things

Lastly, a few little things. The string that (Wide_Wide_)Image returns has a minus as the first character for negative numbers, and a space for positive numbers, possibly this can chop up the text positioning. (Wide_Wide_)Value allows you to convert a string to a number, but doesn't check if the string is all numbers. You have to check this yourself, because if the string contains a character that is not a number, a program error occurs. Converting float to a string puts it in scientific view. If you want it to be a normal decimal number, then you have to adjust it accordingly, but there is a put in the standard library for that, have fun finding the right one. Just kidding, what you are looking for is in Ada.Wide_Wide_Text_IO.Float_IO and is called:

```
procedure Put
   (To   : out Wide_Wide_String;
    Item : Num;
    Aft  : Field := Default_Aft;
    Exp  : Field := Default_Exp);
```

Then there is the reading/writing of data. The simplest solutions are Text_IO for text and Stream_IO for everything else. This is easy and while other variants may use less disk space, it's not worth the extra work as the graphics, sounds and music will take up more space anyway.

6 End

That was roughly the summary of my FOSDEM contribution and I'm now going to continue working on my game. I still hope to be able to sell it successfully in the future, after all the number of commercial games written in Ada is far too low. In addition, you are of course welcome to my live streams, in addition to German I also speak English and if you still don't understand me then don't worry, I usually don't understand myself either.

Citations and references

[1] Wikibooks, Ada Programming, https://en.wikibooks.org/wiki/Ada_Programming

[2] Rosetta Code, https://rosettacode.org/wiki/Category:Ada

[3] GitHub, https://github.com/HonkiTonk/Test-Rollenspiel

[4] GitHub, https://github.com/HonkiTonk/Civ-Klon

[5] YouTube, https://www.youtube.com/user/tpHonkiTonk

[6] Twitch, https://www.twitch.tv/tphonkitonk

[7] Stefan Hild, Ada Looks Good, Now Program a Game Without Knowing Anything, https://fosdem.org/2022/schedule/event/ada_looks_good_game/

The Ada Numerics Model

Jean-Pierre Rosen

Adalog, 2 rue du Docteur Lombard, 92130 Issy-Les-Moulineaux, France.; email: rosen@adalog.fr

Abstract

This paper describes the challenges of making portable calculations across different architectures, and how the Ada model addresses the issues.

Keywords: Ada, numerics, floating points, fixed points.

1 What is numerical analysis?

All programming languages feature so-called "real" types. However, these types are very different from mathematical reals. The mathematical set $\Re$ cannot be represented on a computer: it has an infinite number of values, even for a bounded segment. A computer can represent only types with a finite number of values, and these values can be (at most) rational numbers, since there is no finite representation of irrational numbers.

However, computers are intended to perform computations for the real world, and if you want to compute the circumference of a circle given its radius, you will need Π, which *is* irrational!

Therefore, we can define numerical analysis as the art of making *not too wrong* computations in the real world, using only the finite subset of rational numbers that a computer can handle.

Moreover, since the result is not exact, it is important to be able to compute how wrong (or more precisely uncertain) the result is.

2 Hardware formats and languages

There are many ways of representing real numbers on a computer For example, [1] describes 76 different floating point formats! Moreover, there are often several available formats on a given computer, allowing various trade-offs between range, accuracy, and memory space. For example, the popular IEEE-754 standard [2,3] features 5 standard formats, (3 binary, 2 decimal), + extensions. The old DEC/Vax architecture is another interesting case, as pictured in figure 1:

Size	Exponent	Mantissa
32 bits	8	23
64 bits	8	55
	11	52
128 bits	15	112

Figure 1 VAX floating point formats

Note that there are two different 64 bits format, one with more accuracy (longer mantissa) and the other one with more range (longer exponent). Talking about "short" or "long" floats cannot describe this situation.

Actually, the notion of short or long floats dates back to the early times of Fortran, when most computers had only two floating point formats. Most today's languages still define various floating point types just by the size of the type, without any idea of the actual accuracy (or range) implied by the type, and no definition of the accuracy of computations.

The IEEE-754 standard tried to address these issues by defining a number of standard formats and the associated accuracy, including a precise definition of arithmetic: two computers implementing the standard will give exactly the same results. However, the relation to programming languages is delegated to the programming language standard, including the means to adjust a certain number of features (like exceptions). And from a programmer's point of view, the issue of portability for computers that do not implement the standard remains.

3 Ada model and real types

3.2 Model of arithmetic

The Steelman requirements [4] called for both "approximate" and "exact" computations whose accuracy could be chosen by the user, independently of the underlying architecture.

The solution offered by Ada is inspired by the notion of approximate values in physics: a value does not stand only for itself, but represents a small range of values corresponding to an uncertainty around the value. Based on this, a range arithmetic can be defined, the so-called Brown's model [5].

Physicists use two kind of approximations: absolute approximations, where the uncertainty is the same for the whole range of values (i.e. value is 5V±0.1V), and relative approximations where the uncertainty is proportional to the value (i.e. value is 5V±1%). Similarly, Ada offers two kind of real types: fixed point types corresponding to absolute approximation, and floating point types corresponding to relative approximation.

The syntax of the definition of a fixed point type is as follows:

```
-- Binary fixed
type name is delta step range min .. max;
type Volts is delta 0.01 range 0.0 .. 100.0;
```

```
-- Decimal fixed
type name is delta step digits number_of_digits
  [ range  min .. max ];
type Euros is delta 0.01 digits 11;
```

The syntax of the definition of a floating point type is as follows:

```
type name is digits number_of_digits
  [ range min..max ];
type Length is digits 5 range 0.0..40.0E6;
```

Unlike most other languages, the programmer does not choose one of the types provided by the computer, but expresses the requirements on the mathematical properties of the type. It is up to the compiler to choose an appropriate machine type that fulfils the requirements.

The definition of an Ada real[1] type defines a set of values, called *model numbers* that must be represented exactly on every implementation. If no machine type is available that satisfies the declaration (i.e. that can represent exactly all model numbers), the declaration is rejected by the compiler. The machine type chosen by the compiler may include more values than the model numbers: these extra values are called *machine numbers* and provide more accuracy than the minimum guaranteed by the declaration.

Since the programmer specifies the requirements for accuracy and range, the compiler can choose the most appropriate among all available machine types.

This defines the accuracy of the definition of data. In addition, if the compiler implements the numerics annex, there are additional requirements on the accuracy of operations, including for the functions provided by the numerical libraries: elementary functions, linear algebra, and random number generators.

The principle of these requirements, following Brown's model, is as follows:

- If both operands are model numbers and the mathematical result of the operation is a model number, then the computed result must be that model number, exactly.
- Otherwise, if the mathematical result lies between two model numbers, the computed result can be any value belonging to the *model interval* bounded by the nearest model numbers that surround the mathematical result. This means that the compiler can keep more accuracy if hardware permits, but that the inaccuracy is bounded independently of the hardware.
- The above principle is extended when both operands are only known to belong to some intervals: the mathematical operation is (formally) performed between all values of operands in their respective model intervals, thus defining a *result interval* that can extend over several model intervals, and is not necessarily bound by model numbers; the computed result must belong to this result interval, extended to the nearest model numbers.

This is basically like computing approximations in physics, except for the extra "digitalization noise" due to extending the intervals to the nearest model numbers.

Note that with this model, there is no notion of *underflow*. Some arithmetic models make a special case when a mathematical result is not strictly zero, but the computed result is an exact zero. In Ada, this situation means that zero belongs to the result interval and is an acceptable result; it is not a special condition.

As far as portability of computations is concerned, a program running on two different computers will *not* give the exact same result; however, both results will belong to an interval whose range can be computed independently of the implementation. What Ada guarantees is not an absolute result (which is meaningless anyway, since there is always some uncertainty), but portable bounds on the maximum error of computations.

In addition, the standard requires that all static expressions be evaluated *exactly*; no error can be introduced at compile time by differences of evaluations within the compiler.

3.3 Fixed point vs. floating point types

All languages provide floating point types, and programmers are used to them. But few languages provide fixed point types, and people who are not familiar with them often do not consider their use. They constitute however an additional possibility of Ada that can often better match the problem domain than floating point types. Figure 2 shows a comparison of the respective model numbers of a fixed point type and of a floating point type:

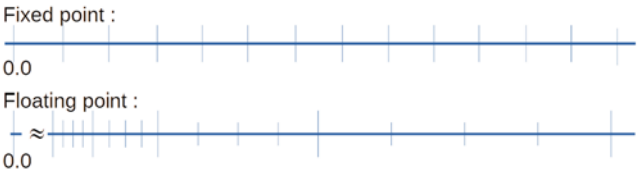


Figure 2 Floating points vs. fixed points model numbers

Fixed point model numbers are evenly spaced, while floating point model numbers are very tight when close to zero, but lose (absolute) accuracy when away from zero. Fixed point types are often more appropriate to represent money or physical values, like readings from measurements devices. This is especially the case for time, where the zero value is arbitrary[2], and there is no reason to provide more accuracy when getting closer to zero[3].

3.2 Numerical portabilities

Portability of numerical computations is not a single goal. There are actually several kinds of numerical portabilities, as allowed by the Ada model.

[1] Remember that the term "real" in Ada encompasses both floating point and fixed point types.

[2] In Ada, the type duration is a fixed point type.

[3] Except for astronomers who model the big bang, of course…

For example, you may want to benefit from the “natural” types of your computer, and be able to determine the confidence range of your result. You will use predefined types, like Float or Long_Float. Knowing your algorithm and given the various attributes provided by Ada, a numerical analyst is able to determine the accuracy of the result. This can be called *a posteriori* portability: the same program run on machine A will print “PI=3.14+-0.05”, while on machine B it will print “PI=3.1415+-0.0005”. Both results are correct, although not identical.

In other cases, you may have requirements on the accuracy of the result; for example, the maximum error on the computed speed of a vehicle must not exceed 5 km/h. Like any requirements, it should be provably met! Given the Ada model, your numerical analyst is able to determine the required accuracy (and range) of the data that are part of the computation. These can be expressed in Ada terms, and guaranteed by the implementation, independently of any architecture. This can be called *a priori* portability: different machines may give different results, but they will all be within the stated requirements.

Conclusion

For many programming languages, as soon as a numeric value includes a decimal point, the only choice is between short and long floats, which usually boils down to short floats if memory space is an issue, or long floats “just to be on the safe side” otherwise. Not exactly an engineered approach…

Ada offers a rich range of real types that can provably match stated requirements and guarantee the maximum uncertainty of the computed results, independently of the underlying architecture. When it comes to choosing a programming language for a development, this aspect of Ada should really be more known to all those with requirements on the accuracy of computations.

References

[1] http://www.quadibloc.com/comp/cp0201.htm

[2] Wikipedia, IEEE 754, https://en.wikipedia.org/wiki/IEEE_754

[3] "IEEE Standard for Binary Floating-Point Arithmetic". ANSI/IEEE Std 754-1985.

[4] Department of Defense, *Requirements for High Order Computer Programming Languages "STEELMAN"*, June 1978. https://en.wikisource.org/wiki/Steelman_language_requirements

[5] W. S. Brown, *A Simple but Realistic Model of Floating-Point Computation*, ACM Transactions on Mathematical Software, Volume 7 Issue 4, Dec. 1981, pp. 445-480.

Alire 2022 Update

Alejandro R. Mosteo
Centro Universitario de la Defensa, ctra. de Huesca s/n, 50090, Zaragoza, Spain; email: amosteo@unizar.es
Fabien Chouteau
AdaCore, 46 rue d'Amsterdam, 75009, Paris, France; email: chouteau@adacore.com

Abstract

The Alire Package Manager released its first stable version in 2021 and, since then, it has seen continued improvement and new features. Herein we present the latest major features that have been added for the benefit of Ada developers, which include toolchain installation, a publishing assistant, generation of configuration code, and a so-called pin system for concurrent development of multiple projects. We also take a look at the status of its ecosystem of available libraries.

Keywords: Alire, Package Manager, Dependency Management, Open Source.

1 Introduction

It is nowadays a common expectation of open source developers that some kind of package manager will exist for a language, easing the retrieval of dependencies, reuse of libraries, and publishing of projects (*crates* in Alire parlance). Alire [1] positioned as such a community project for Ada and SPARK and saw the release of its first stable release [2] in 2020. Since then, the project has seen steady progress, having seen the release of its 1.1 version, which contains several major new features.

These features are the focus of this writing, and they take the project's command-line tool, `alr`, to a new level of usefulness for developers, in some instances surpassing the capabilities of package managers for other languages. A video presentation of these features is also available on-line [3].

The main features that are included in this update are as follows. Installation of toolchains is covered in Section 2. The publishing assistant is described in Section 3. The solution provided to develop several crates in tandem is the subject of Section 4. Configuration of crates based on code generation is given in Section 5. A brief look at the status of the ecosystem of crates in the project is presented in Section 6. Finally, concluding remarks are given in Section 7.

2 Installation of toolchains

Through the `alr toolchain` command, it is now possible to download and select a variety of compilers based on the FSF source tree [4], with a corresponding `gprbuild` tool for build management. Although some Linux distributions already are able to provide such a toolchain, they have their own release life cycles that might make their compilers somewhat older. Alire makes its compilers, hosted on x86_64 at the moment, available generically for Linux, but also for macOS and Windows, including cross-compilers for ARM, AVR, and RISC-V. These compilers are distributed as binaries, although their build process is open and can be perused at [5].

In Alire, one GNAT and `gprbuild` can be selected as a default (there is also the option of not selecting any, and relying in a user-provided toolchain existing in the environment where `alr` is run), via the `-select` switch. However, the user can, if so desired, install several toolchains. The default toolchain is used normally for compilation, unless some dependency specifically requires a particular version or cross-compiler, in which case the appropriate toolchain is automatically used.

3 Publishing of releases

In Alire, we refer generically to a packaged project as a *crate*, which in turn can offer one or more releases, each identified by their semantic version [6]. For authors, publishing and updating their work should be as easy as possible, which is the motive behind the `alr publish` feature.

Alire relies on indexes to provide releases. Indexes may be hosted anywhere, even privately, but the Alire project hosts the *community index* [7], which is the default in use and the one to which open source authors are encouraged to contribute. An index is composed by individual *manifest* files, one per release, in which their metadata is detailed in TOML format [8]. The community index itself is just a public git repository containing these files.

Thus, in essence, the process of publishing a release consists in adding its manifest to the appropriate location of this repository. In usual GitHub methodology, the repository itself is read-only except to its maintainers, so all modifications by third parties are achieved through a *pull request* [9], which is automatically vetted by GitHub Actions, and then manually reviewed, approved and merged by Alire maintainers.

At present, the `alr publish` command helps with the following steps:

- Verifying that the user has properly forked the community index in their own GitHub account.
- Verifying that all necessary metadata is already defined in the working manifest[1] of the project.

[1] The working manifest contains a subset of the information that goes into the index manifest, and is always present inside an Alire working release.

```
[[pins]]
subdir = { path = "../my/bar" }
# Use a local folder to override a dependency

remote = { url = "https://github.com/baz.git" }
# No commit, will track HEAD, will update on 'alr update'

branch = { url = "https://gitrepo.com/wip.git"
           branch="feature" }
# Track a remote branch, will update on 'alr update'

commit = { url = "https://gitlab.com/gru.git"
           commit="..." }
# Explicit commit, will not be updated
```

Figure 1: Different pin kinds.

- Fetching and compilation of sources from their public on-line publishing location.
- Generation of the index manifest, and providing of an URL to which the manifest file can be uploaded to initiate the pull request.

4 Dependency overriding with pins

An usual occurrence during development is to have to update concurrently a few interrelated projects. Likewise, when working with bleeding-edge code, it may be necessary to use developing versions of code from third parties, not yet published to the community index.

By default, dependencies in Alire are always satisfied by releases in the community index, or some other index provided by the user. However, publishing versions very often, although relatively straightforward, is too unwieldy, and will not help with code maintained by other developers. The new pinning mechanism in Alire is designed to tackle these situations.

A pin is a dependency override in which an alternative source is specified for the code of a dependency, bypassing any indexes. By design, a pin will always fulfill the dependency on a crate, so the responsibility of making sure that a pin is proper falls on the developer. As pins are not intended for final publishing, but only during transitory development stages, this is a reasonable and even expected requisite.

Pins are defined, as all other information, in the working manifest of the releases being developed. Their precise syntax is described in the project documentation [10], boiling down to a few fields of information per dependency (see Fig. 1).

```
my_lib
├── alire.toml
├── examples
│   └── alire.toml
└── tests
    └── alire.toml
```

Figure 2: A crate with two nested subcrates.

4.1 Local pins

Local pins require only a relative or absolute path to a directory, which will contain the code for the dependency. This is practical when the overridden dependency is being modified at the same time than its dependent crates.

Another use case is to have nested crates inside a root project, that provide for example testing projects or demonstration executables (see Fig. 2). Since these nested crates are not intended to be built during the normal use of the library, and may furthermore bring in additional dependencies themselves, in this way they remain separate yet always in sync with their parent project, for local use.

An example of the manifest in such a nested crate is as follows, in this case for a testing crate that uses `aunit` to verify the main library. Note how the `my_lib` dependency, which would be the library intended to be published, is simply pinned through a relative path to the parent folder, where the root manifest is located, as seen in Fig. 2.

```
[[pins]]
my_lib = { path="../" }   # Use parent folder for dependency

[[depends-on]]
aunit = "*"               # Extra dependency, only for testing
```

Listing 1: Manifest of nested testing crate.

When the target of a pin contains itself an Alire manifest, it will be processed as a regular Alire crate and its own dependencies and pins added to the closure of sources required for building. Otherwise, it is assumed that a GPR project file will exist at the root that will enable the building of these sources.

4.2 Remote pins

Sometimes, it is necessary to use bleeding edge code from third parties that still has not yet made into the community index. In that case, Alire can save the manual steps of fetching the code and pinning to it with a path, as just shown. This is achieved with a pin that provides a URL to a git repository and, optionally, a branch or commit.
When no branch is supplied, the repository default branch is tracked, so an `alr update` will not only update regular dependencies, but fetch the latest changes from such a pin. Likewise, when a branch name is given, the working is the same but for the given branch. Finally, when a commit hash is specified instead, this is an immutable reference that will not result in updates.

5 Crate configuration

The Ada designers consciously made a decision to avoid preprocessor macros. Likewise, the `gprbuild` tool does not allow launching other tools before the compilation step. There are circumstances, however, in which some kind of code tailoring is necessary that goes beyond selecting among different preexisting source files. In embedded development, in particular, not all features of Ada may be available due to restricted run-times, further complicating the self-configuration of data structures based on run-time gathered information. Also, code size in small boards may be a critical factor.
Traditionally, one would address these issues with some other build tool, like perhaps `make`. However, for developers using

```
[configuration.variables]
Device_Name   = {type = "String",
                 default = "no device name"}
Print_Debug   = {type = "Boolean", default = false}
Debug_Level   = {type = "Enum",
                  values = ["Debug", "Warn", "Error"],
                  default = "Warn"}
Buffer_Size   = {type = "Integer",
                  first = 0, last = 1024,
                 default = 256}
Max_Power     = {type = "Real",
                  first = 0.0, last = 100.0,
                 default = 50.0}
```

Figure 3: Definitions in a configurable crate.

Alire as their driving tool, there is now a simpler alternative, which is the new crate configuration feature.

This feature enables the creator of a crate to supply a number of variables, of a given type, and their bounds and optional default value (see Fig. 3). The usual predefined types of Ada are supported, and also enumerated types.

These values can be set in a different, dependent crate, with any conflict being detected and reported (Fig. 4). In this way, there is a flow of information from the dependent crate into the dependency, which at the time of its compilation is taking into account the settings requested by the dependent crate.

More concretely, before the compilation run, Alire generates static files containing definitions for these variables as Ada, C and GPR sources (Fig. 5). This way, the values are readily available for mixed-language programming, and also to be used from other GPR project files. Once files are generated, the compilation process follows the usual topological order of the dependency tree.

6 Ecosystem overview

At the time of this writing, the Alire project indexes 243 crates, of which 14 are in SPARK or SPARK-friendly. Notably, the most popular tag is “embedded” with 32 crates, which is a testimony to the community of embedded developers that are taking advantage of the project. Other popular tags completing the top 10 are “nostd” (26 hits), “gnatcoll” (14), “web” (12), “database” (10), “rp2040” (9), “bindings” (9), “sql” (8), and “game” (8).

7 Conclusion

The Alire project continues its evolution incorporating more advanced features. In this update, four new main features have been reported. First, Alire is now able to install binary toolchains, hosted in x86_64 Linux, macOS, and Windows, including cross-compilers for ARM, AVR, and RISC-V. Publication of releases is now simpler too, thanks to the publishing assistant that verifies information and generates the single manifest file required for a new release. The release process

```
[configuration.values]
my_crate.Device_Name = "Custom␣device␣name"
my_crate.Print_Debug = true
my_crate.Debug_Level = "Error"
my_crate.Buffer_Size = 42
my_crate.Max_Power = 42.0
```

Figure 4: Configuration settings in a dependent crate.

```
package my_crate_Config is
   Crate_Version : constant String := "0.1.0";

   Buffer_Size_First : constant := 0;
   Buffer_Size_Last : constant := 1024;
   Buffer_Size : constant := 42;

   type Debug_Level_Kind is (Debug, Warn, Error);
   Debug_Level : constant Debug_Level_Kind := Error;

   Print_Debug : constant Boolean := True;

   Max_Power_First : constant := 0.00000000000000E+00;
   Max_Power_Last : constant := 1.00000000000000E+02;
   Max_Power : constant := 4.20000000000000E+01;

   Device_Name : constant String := "Custom␣device␣name";
end my_crate_Config;
```

Figure 5: Generated Ada package for static compilation.

is based on opening a GitHub pull request that adds this file to the community index.

For more advanced developer workflows, there is a new “pinning” feature in which dependencies can be overridden by local folders or remote repositories, enabling the concurrent development of several projects, and also the creation of nested testing and demonstration projects without interfering with a main library intended for publishing.

Finally, generation of source files prior to compilation is now available, containing configurable variables of scalar, string and enumerated types, with ranges where applicable and optional default values. These values can be configured by any dependent crate, enabling powerful configuration of libraries even for embedded contexts in which static compilation is a priority due to run-time or board constraints.

The Alire project documentation, downloads and source code are available on-line at `https://alire.ada.dev`.

References

[1] A. R. Mosteo, “Alire: a library repository manager for the open source Ada ecosystem.,” *Ada User Journal*, vol. 39, no. 3, 2018.

[2] F. Chouteau, P.-M. de Rodat, and A. R. Mosteo, “Alire: Ada has a package manager.” `https://archive.fosdem.org/2020/schedule/event/ada_alire`, 2020. FOSDEM.

[3] A. R. Mosteo and F. Chouteau, “Alire 2022 update.” `https://fosdem.org/2022/schedule/event/2022_alire_update`, 2022. FOSDEM.

[4] Free Software Foundation, “GNAT GNU Ada.” `https://www.gnu.org/software/gnat`, 2022.

[5] Alire Project, “GNAT FSF builds.” `https://github.com/alire-project/GNAT-FSF-builds`, 2022.

[6] T. Preston-Werner, “Semantic versioning 2.0.” `https://semver.org`, 2022.

[7] Alire Project, “Community index.” `https://github.com/alire-project/alire-index`, 2022.

[8] T. Preston-Werner, “Tom’s obvious minimal language.” `https://toml.io`, 2022.

[9] GitHub, “Collaborating with pull requests.” `https://docs.github.com/en/pull-requests/collaborating-with-pull-requests`, 2022.

[10] Alire Project, “Work-in-progress dependency overrides.” `https://alire.ada.dev/docs`, 2022.

SweetAda: A Lightweight Ada-Based Framework

Gabriele Galeotti

SweetAda home, 50019 Sesto Fiorentino (FI), Italy; email: gabriele.galeotti@sweetada.org, gabriele.galeotti.xyz@gmail.com; website: https://www.sweetada.org, https://github.com/gabriele-galeotti/SweetAda

Abstract

This article tries to describe what is SweetAda, how it was developed, its uses, and its possible future routes.

Keywords: Ada, SweetAda.

Introduction

SweetAda could be described, at a first glance, like something resembling a build system to produce a working, albeit simple, Ada code on various devices.

The origin of SweetAda can be found on the classical scheme involving a software project whose purpose is running a primitive kernel (maybe neither a "classical" kernel at all) on some CPU device. Following the combined GNU/Linux [1] open-source revolution, thousands of projects like that have been developed in the last decades, so the idea is, after all, not so newsworthy.

Trying to spend time on something different and not reinvent the wheel once more, SweetAda slowly born as a radical departure from the basic concept of using a low-level language for its implementation (mostly C or one of its dialects), propelled with a "naive" approach towards an experimental Ada-based one, and the wish to see it run on every possible piece of hardware available.

Despite these loose initial criteria, several key points were anyway stated during the inception of such a system, e.g., the complete absence of an underlying OS, the use of standard GNU FSF toolchains, a Makefile-based build engine, and avoiding the use of everything not provided in the form of source code. The expected result is to produce a thing like a binary image that can bring a CPU out of reset and execute Ada code able to manipulate some I/Os and peripheral devices.

On the other hand, CPU computing devices are increasingly becoming complex and hard to manage. A user view limited to objects exposed by the RTS could be too much coarse in some context, so SweetAda tries to stay more close to hardware, possibly without disregarding other things.

That being said, in order to obtain this result, SweetAda freely chooses to not provide a full RTS, and lacks the fundamental OS-dependent primitives, like tasking and protected objects, which imply the presence of a rather advanced, already-existent support behind the scenes.

Looking forward, high-level Ada language constructs will eventually have to be mapped on top of the more primitive objects exposed by SweetAda.

1 Initial development

Unfortunately, this kind of approach is not a simple one.

Problems are the availability of useful tools needed in order to be proficient during the development. To take a CPU out of reset and make it able to finally execute a single line of high-level language can be daunting, and once you enter the domain of Ada, things do not improve. Assembling a stable sandbox is not straightforward and you cannot write every CPU specific low-level fragment of code in pure assembly, since this could soon divert from the original intent of the project.

If you want to put Ada code on the bare metal, then, you will soon find yourself debugging a hostile emulator, or diving into the inners of a JTAG driver which refuses to load your binary image. As a matter of fact, I had to rewrite almost from scratch a JTAG driver for a MIPS platform (besides buying a DECstation 5000 [1] and burn EPROMs to test real code). Furthermore, even if you have a test bench based on an emulator, chances are that you cannot see anything until you correctly setup the (emulated) UART to successfully send something out. So I also wrote a sort of plugin for QEMU [1] that shows some I/O ports in a graphic window to check that the program is alive (incidentally this could be very useful not only at the development stage, but also for users).

In the end, in order to develop SweetAda for the maximum possible range of CPUs, appropriate hardware devices are instrumental. But some targets could appear rather exotic, because sometimes is very difficult to find an affordable development platform. For example, in the case of Hitachi SH4 CPU [1], the popular Dreamcast [1] game console is being used as a target.

Toolchains, then, are another big issue to solve. If you want to provide a friendly environment, you have obviously to provide them for at least the most common machines, like a Linux [1] machine or a Windows [1] one. To create a single GNAT toolchain is a process involving four different (cross-) compilers, and invariably some sort of issue shows up, maybe because the build crashes with some form of error due to bad configure options, or maybe because the compiler doesn't run on the target OS due to a missing library. To create toolchains and utilities became soon almost a project in itself.

To make the long story short: you start writing Ada, but less than 5% of your development time will be exactly aimed at this business.

2 Layout, RTS, low-level code, bits and bytes

A choice made rather early during the development of SweetAda was to not follow the classical scheme of having everything suited around an existing RTS.

Almost always (in accordance with Ada guidelines), every piece of software is depending on an RTS that transparently provides a foundation base where your code will live. For practical (and commodity) reasons, I avoid this kind of layout, since it would have prevented me developing for more than one architecture, putting too much effort on something that would end up as a trivial software port and nothing more.

So, starting from a "pragma No_Runtime" base that was hardly able to be even referenced by the context, I create a series of low-level units whose purpose was to deal with the various kinds of hardware. Since in any case a ZFP/SFP RTS would have been mandatory, I started patching the original GNAT sources to not include heavy features, like the secondary stack (which is now partially implemented), or the floating-point support (also in the development stage). It was a long and tedious process but it finally worked out quite satisfactory.

Complete control about LibGCC was another point that I liked to implement. In this context, this does not mean an alternative implementation, but I felt that the opportunity of having exactly every single bit of code in source form (except clearly the code generated by the compiler) would have been a good thing.

To keep up with this fact, SweetAda has a copy of the last current LibGCC, kept synchronized together with the last toolchain version. LibGCC sources are only cosmetically cleaned up and carry no patches whatsoever, and when you build SweetAda for some platform/CPU, a LibGCC copy is automatically incorporated in the CPU base library that will be linked into the final executable. Besides, this allows also a more fine-grained control on the object components.

This proved fine, anyway there is also the more "conservative" choice to link in the standard copy that comes as a component of the compiler toolchain (also because the in-source LibGCC was a later addiction to the SweetAda system).

Still speaking about LibGCC, this is why does exist an homonym subdirectory in the core component of SweetAda: here you will find Ada implementations for the very basic functions that the compiler uses for doing low-level arithmetic. If, for example, you run out of memory in a resource-limited microcontroller, maybe you can just use a 32x32 bitwise multiplication without neither being concerned about using LibGCC.

2.1 Bits&Bytes

Handling of physical hardware is clearly one of most difficult things to do. Although GNAT has a bunch of pragmas/aspects, CPU internals and I/O registers are always extremely complicated to describe correctly, and their management is often unclear, maybe because registers overlap at identical addresses. Description by records is thus generally limited to the single register (and still this is not ideal because the same register has different meanings, depending on some "mode" determined by a flag in another register). The whole peripherals should then be described by means of specific offsets mapping and helpers.

An approach used, but now abandoned, was to create a bunch of generics, on top of which to create a long string of dedicated access helpers, but I found that unpleasant, unnecessarily verbose and error-prone, besides being inefficient. Furthermore, this prevents the treatment of some corner case or highly-specialized behaviour. So, disregarding style and elegance, a raw approach proved to pay better. Besides that, the new approach, although partially using unchecked conversions, operates in a very controlled context, and does not hurt safety nor readability.

To ease the handling of all these low-level things, units like Bits, LLutils and MMIO are provided in core directory of the system, and are completely overridable and customizable.

2.2 System layout

The "unorthodox" nature of SweetAda can be immediately ascertained when you look at the organization of the filesystem.

Instead that having a giant RTS-dictated structure, we have a more free organization of the various components, like in mainstream C-based project. There is a CPU-dedicated tree for low-level dependent code, a core tree for basic CPU-independent code, a devices tree for peripheral handling, and a modules hierarchy for yet independent units that implement not-mandatory features. There is also a C nano-library (whose functions are actually calling Ada subprograms) which could be used to incorporate C code into a project, perhaps to ease a software port.

Then finally comes the platforms tree, where the intended target device BSP code is implemented.

2.3 Configuration

The configuration of SweetAda is carried out by a chain of Makefile-style variable declaration files that the build engine includes back-to-back, in order to create a detailed description of what it is going to build. The configuration files are written in simple Makefile syntax, so the more complicated thing to write down is an "ifdef" conditional to suppress a GNAT warning switch, or to change one of the default core units in favour of a customized one. Configuration variables can be selectively suppressed or augmented during the various phases of this process.

The first file sits in the top-level directory and describes basic informations, like paths to toolchains and their triplet names, then, once the system knows which is the intended target platform, the corresponding configuration files of both the target CPU and target platform will be parsed.

Anyway, SweetAda provides as an option also a GPRbuild-style configuration inside the main Makefile build engine

(except for the link phase which, is always executed by the latter).

3 What is able to do SweetAda?

Being in a development phase, SweetAda is not fullfilled with every possible feature, but in some cases is able to perform, generally speaking, some basic activities, like program an Arduino UNO [1] board and make some kind of I/O digital processing.

But you can also respond to an UDP datagram received by your ancient x86 motherboard equipped with an NE2000 network card, or you could send a message from your emulated S/390 IBM [1] mainframe to an X3270 terminal once SweetAda is coming out from IPL.

Let's bring a Raspberry Pi 3 [1], and see how is possible to blink a LED, and send a message out of serial port when the temperature of the SoC rises too much (refer to examples in the application directory and connect GPIO serial pins through a simple FTDI232 USB converter).

Here is a cutted-down example:

```
with RPI3; use RPI3;
with Console; use Console;
-- called by BSP_Setup
procedure Run is
  T : Integer;
begin
  -- wire up some GPIOs to LEDs and USB serialport
  GPFSEL0.FSEL5 := GPIO_OUTPUT;
  GPFSEL0.FSEL6 := GPIO_OUTPUT;
  loop
    T := Temperature_Get (Temperature_ID);
    if T > 70 then
      GPSET0 := (SET6 => True, others => False);
      GPCLR0 := (CLR5 => True, others => False);
      Print ("T is too high.", NL => True);
    else
      GPSET0 := (SET5 => True, others => False);
      GPCLR0 := (CLR6 => True, others => False);
    end if;
  end loop;
end;
```

This example is clearly convoluted because the Raspberry Pi 3 support hasn't yet great notions about time and CPU management, so the CPU sits in a loop, wasting time only for the pleasure to see a LED turn on.

Many other examples can be found in the application directory, where every platform has a little loop to perform some activity or to test the software environment.

Yet as another example, the following is a screenshot showing SweetAda running on a Malta MIPS [1] board under QEMU, and has just executed:

- an initial bootstrap, out of reset

- detection of PCI interfaces available

- filling of the VGA LCD with a banner

- directory listing of a FAT virtual disk attached to the IDE interface,

and is now looping through an I/O port count, while the interrupt timer is running at 1 kHz, blinking the LED once per second.

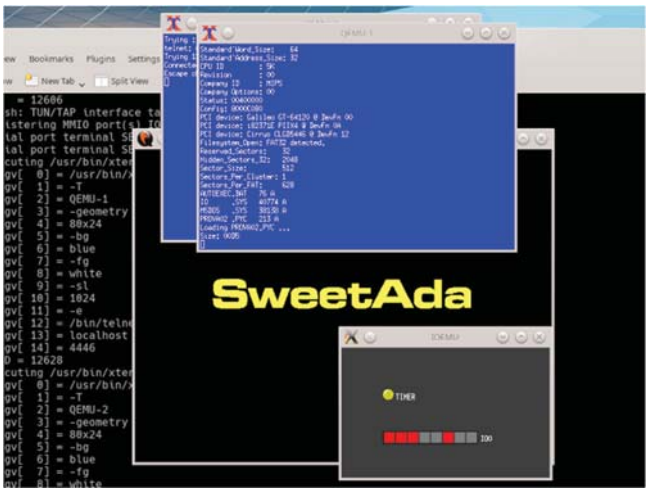


Figure 1 SweetAda running on a Malta MIPS in QEMU.

Conclusion

SweetAda is currently in a deep phase of development.

After initial releases, much work was done in order to keep the project clean and to reduce dependencies from host machine environments and toolchains. For many CPUs support is not completely deployed yet, and various components like interrupt handling and ABI interfaces have yet to be fully implemented up to an operational stage. The SFP RTS is to be refined, while the ZFP is highly stable, like the whole build engine.

The future of SweetAda could be to follow and expand the current approach, or to converge towards a standard RTS-based layout, depending on which will be the better route.

Anyway, the system so far have proved to be modular and flexible, is working fine for every platform/CPU tested, and some contexts are already powerful enough to perform very interesting activities.

References

[1] Trademarks and trade names are properties of their respective owners.

Use (and Abuse?) of Ada 2022 Features in Designing a JSON-Like Data Structure

Alejandro R. Mosteo
Centro Universitario de la Defensa, ctra. de Huesca s/n, 50090, Zaragoza, Spain; email: amosteo@unizar.es

Abstract

Ada 2022 introduces new features that enable more natural-looking initializations. On the one hand, it is now possible to use universal numbers and strings to initialize any private type; on the other hand, containers can be initialized directly without the explicit use of converting functions. Both features are enabled through new aspects that associate user-defined subprograms with the necessary initialization calls.

In this work, the possibility of using these new features is explored to define a container type that allows initializations using heterogeneous types, as is usual in textual formats for structured data such as JSON, TOML and YAML, and still without the need to resort to explicit conversion calls or "crutch" functions like the often-seen trick of overloading the "+" operator. Although this has proven ultimately avoidable, there is still the need to use qualifications in certain circumstances.

Keywords: Ada 2022, JSON, Yeison, user-defined initialization.

1 Introduction

The Ada 2022 revision of the language added a few new aspects [1, 2] that enable two orthogonal features: initializing any variable of a user-defined type with a universal number or string, and initializing a collection with a list of values or key-value pairs. User-defined initialization is used to great effect, for example, in the new big-number packages in the standard library, by enabling to write a number as long as necessary to initialize a `Big_Integer`. Previously, an equivalent package would have been forced to use something like

```
Int : Big_Integer := To_Big_Integer ("1234567890123456789");
```

whereas now one can simply write

```
Int : Big_Integer := 1234567890123456789;
```

Likewise, initialization of containers is intended to reduce boilerplate when creating a collection. These are two examples of now valid constructs:

```
type User_List is tagged private
  with ...;  -- Aspects omitted for now
type User_Map is tagged private
  with ...;  -- Aspects omitted for now

List : constant User_List := (1, 2, 3, 4, 5);
Map : constant User_Map := ("key1" => "value1",
                            "key2" => "value2");
```

Based on those examples, one would be right in thinking that they can be combined and nothing precludes, in good Ada orthogonality fashion, having a container of big numbers initialized by combining both features.

From this starting point, the objective of this work was to see whether, by exploiting these new aspects and perhaps some other features, it would be possible to initialize a JSON-like data structure without resorting to helper conversion functions. For example:

```
X : Yeison.Map :=
    ("array"   => (1, "2", 3.0),
     "boolean" => True,
     "color"   => "gold",
     "number"  => 123,
     "object"  => ("a" => "b",
                   "c" => "d"));
```

Listing 1: Desired Ada initialization.

which would be equivalent to this JSON-formatted [3] information:

```
{
"array"   : [1, "2", 3.0],
"boolean" : true,
"color"   : "gold",
"number"  : 123,
"object"  : { "a": "b",
              "c": "d" }
}
```

In both examples we can find heterogeneous elements in arrays and maps, which poses an obvious difficulty in traditional Ada.
The efforts discussed in this work are fully available in source code form [4], and have been packaged under the in-jest name of YEISON. A video presentation is also on-line [5].
In the following, the necessary new aspects are presented in Section 2. Difficulties found in trying to apply them are described in Section 3. The best solution identified to date is detailed in Section 4, and concluding remarks are given in Section 5.

2 Aspects involved

The feature found in the Ada Reference Manual as *User-defined literals* [1] defines three new aspects that connect a function with the values being created. For conciseness, only the integer and string cases are shown in the following listing, although a similar `Real_Literal` aspect exists for real types:

```
type My_Integer is private
    with Integer_Literal => Init ;
function Init (Img : String) return My_Integer;
-- Create a number from its ASCII image

type My_String is private
    with String_Literal => Init ;
function Init (Str : Wide_Wide_String) return My_String;
-- Store the string with whatever representation we prefer
```

In regard to container initializations, the pertinent topic is *Container Aggregates* [2], which describes aspects used to create and modify a container. Only those relevant to lists and maps (presuming that an array would be always indexed from 0, like in JSON, and thus being somewhat equivalent to a list) are shown:

```
type My_List is private with
    Aggregate => (Empty => Empty,
                  Add_Unnamed => Append);
function Empty return My_List;
procedure Append (List : in out My_List; Val : Value;
-- E.g., List := (E1, E2, E3) will call Empty and then
-- thrice the Append procedure.

type My_Map is private with
    Aggregate => (Empty => Empty,
                  Add_Named => Insert);
function Empty return My_Map;
procedure Insert (Map : in out My_Map; K : Key; V : Value);
-- In this case, for Map := ("a" => "b", "c" => "d"),
-- Insert would be called twice with the "a"/"b", "c"/"d"
-- argument pairs.

-- Naturally, these aspects require matching types for
-- key and element types of the containers, which usually
-- would be defined in a generic package with those types
-- as generic formal parameters.
```

Note how the `Aggregate` aspect is itself composed by an aggregation of differently named components, depending on the kind of the container (a list, a map, or a discretely indexed vector (omitted here, as it is not used in this library).
The Reference Manual defines a few rules and limitations that will be discussed when relevant in the next section. However, it should be clear by inspecting these aspects how they are the building blocks that will allow to attempt to create the intended data structure.

3 Difficulties found

Among all the attempts at achieving the code depicted in Listing 1, there where roughly two main approaches: one in which the datatype is unique, and to which all aspects would be applied; and another approach in which a family of interrelated classes are used for each data kind: elementary, list and map.

3.1 Monolithic type

In this approach, the ideas boiled down to apply all relevant aspects to a single type:

```
type Any is tagged private with
    Integer_Literal  => To_Int,
    Real_Literal     => To_Real,
    String_Literal   => To_String,
    Aggregate =>
        (Empty           => Empty,
         Add_Named       => Insert,
         Add_Unnamed => Append)
```

It turns out that initializing a same type with both numeric and string literals is allowed, so the first three aspects pose no problem. However, there is a explicit rule (ARM 202x 4.3.5, 5/5 [2]) forbidding using both named and unnamed aggregates:

> *If Add_Named is specified, neither Add_Unnamed nor Assign_Indexed shall be specified.*

This idea, then, is a no-go that would require the use of explicit conversion functions and auxiliary types to work around that rule.

3.2 Family of classes

Given the previous finding, another approach is to apply the incompatible aspects to two different types. Here, again, several options open. One possibility is to try to have Insert/Append procedures with several profiles, so a mixture of types can be stored in a container. An example showing only integers and strings is as follows (omitting unrelated aspects):

```
type List is tagged private with
    Aggregate => (Add_Unnamed => Append);
procedure Append (L : in out List; I : Integer);
procedure Append (L : in out List; S : String);
```

This, unfortunately, once again does not work, at least with GNAT 11.2.4, as only the last Append prototype is associated to the aspect, with others being silently ignored. At the time of this writing, the author has not pinpointed whether this is intended behavior, or a bug where either it should be rejected or all procedures should be recognized during aggregate construction.
This finding prompted the use of a solution based on class-wide types, which is described in the following section.

4 Current solution

The best solution found until date consists in a container where elements belong to a same class:

```
type Any is tagged private with
    Integer_Literal  => To_Int,
    Real_Literal     => To_Real,
    String_Literal   => To_Str;
```

Listing 2: Base type for literal initializations.

This base type serves to allow initialization with elementary values. Although later there are derived wrapper types defined for integers, reals, and strings, these aspects must be also present in the base class, or initializations fail to recognize values as appropriate for aggregate components.
The derived classes are (with some details omitted) thus:

```
type Bool is new Any with private;
function False return Bool;
function True return Bool;

type Int is new Any with private with Integer_Literal => To_Int;
type Real is new Any with private with Real_Literal => To_Real;
type Str is new Any with private with String_Literal => To_Str;

type Map is new Any with private with
    Aggregate =>  (Empty          => Empty,
                   Add_Named      => Insert);
procedure Insert (M : in out Map; Key : String; Val : Any'Class);
type Vec is new Any with private with
    Aggregate =>  (Empty          => Empty,
                   Add_Unnamed    => Append);
procedure Append (V : in out Vec; Val : Any'Class);
```

Here, the type `Vec` fulfills here the role of a simple list or 0-indexed vector. Note that both `Map` and `Vec` expect elements to be of type `Any'Class`, which enables heterogeneous initializations. Now, these definitions[1] allow the writing of initializations such as (knowing that they are inside a root package named `Yeison`):

```
A1 : constant Yeison.Int  := 1;
A2 : constant Yeison.Str  := "string";
A3 : constant Yeison.Bool := Yeison.True;
A4 : constant Yeison.Real := 3.14;

M1 : constant Yeison.Map := ("one" => 1,
                             "two" => "two");
M2 : constant Yeison.Map := ("one" => A1,
                             "two" => "two",
                             "three" => M1);
V1 : constant Yeison.Vec := (1, M2, "three", 4.0);
V2 : constant Yeison.Vec := (A1, 2, M2, V1, "five");
```

Listing 3: Some initialization examples.

That is, we truly can have heterogeneous initializations, mixing both literals or all kinds and variables, and still without using any adapter function.

4.1 The trouble with class-wide values

What then about the objective we set out for with Listing 1? It turns out we face and old "problem" (actually a logical imposition of the language, even if in some cases it would be unambiguous): when a literal of an aggregate type is used for a class-wide placeholder, one must qualify the value with its type. An example that could already arise in Ada 2012 is:

```
type Base is tagged null record;
type Int  is new Base with record
   I : Integer;
end record;

function Make (I : Integer) return Base'Class;
```

Now, in the body of `Make` we are forced to write as follows, even if there is no possible ambiguity:

```
function Make (I : Integer) return Base'Class is
begin
   return Int'(I => I);      -- Valid
   return (I => I);          -- Invalid
end Make;
```

The second, invalid, return statement in the previous listing fails with an error *"Type of aggregate cannot be class-wide"*. The same problem afflicts the presented design in examples such as

```
V : Yeison.Vec := (1, "2", 3.0, ("four" => 4));
M : Yeison.Map := ("a" => 1,
                   "b" => "2",
                   "c" => 3.0,
                   "d" => (1, 2, 3, 4));
```

with the four element of both initialization expressions.
It is a pity that the user-defined literals, that result in a precise value of type `Any` (Listing 2) are accepted, whereas an expression that can only be interpreted as an initialization of a `Map` or a `Vec`, respectively, is instead "hijacked" by the aggregate initialization, resulting in the aforementioned error.

[1] The complete code is available on-line [4].

Again, we are back to the problem of being unable to define both `Add_Named` and `Add_Unnamed` aspects to the base `Any` type.

4.2 Final outlook

With these limitations in mind, the closest code to Listing 1 still requires some "excess" qualifications, yet still not involving auxiliary functions:

```
X : Yeison.Map :=
   ("array"   => Yeison.Vec'(1, "2", 3.0),
    "boolean" => True,
    "color"   => "gold",
    "number"  => 123,
    "object"  => Yeison.Map'("a" => "b",
                             "c" => "d"));
```

Listing 4: Final result with qualifications.

These qualifications, as seen in Listing 3, were not necessary when actual variables of a concrete type were used instead of in-place aggregate initialization expressions. Here, the qualifications resolve the ambiguity and ensure these values have the required `Map` or `Vec` type, which fulfills the `Any'Class` elements expected by `Map` and `Vec`.

5 Conclusion

This work described current attempts to achieve initialization of a containers with heterogeneous data such as can be found in JSON representations, without resorting to helper conversion functions. This was largely achieved thanks to a combination of new aspects for user-defined initialization from literals, container aggregate initialization, and tagged types that provide the necessary heterogeneity.
This solution fails at being entirely free of boilerplate, as the use of class-wide types requires the use of qualification expressions in nested elements of a collection. Apparent limitations of the current GNAT implementation hint at a possible alternative solution if eventually more than one function profile is accepted for container initializations. Nonetheless, the reduction in boilerplate achieved is remarkable and is a testimony to how old and new orthogonal features combine to facilitate the stated goal.
Finally, since the new user-defined initialization aspect only applies to literals, there is still no risk of marring the language with the unintended appearance of implied or automatic conversions that make other languages such as C++ much harder to unravel for a human reader.

References

[1] "User-defined literals." `http://www.ada-auth.org/standards/2xrm/html/RM-4-2-1.html`. Ada Reference Manual (Ada 202x Draft 32).

[2] "Container aggregates." `http://www.ada-auth.org/standards/2xrm/html/RM-4-3-5.html`. Ada Reference Manual (Ada 202x Draft 32).

[3] L. Bassett, *Introduction to JavaScript object notation: a to-the-point guide to JSON*. " O'Reilly Media, Inc.", 2015.

[4] A. R. Mosteo, "YEISON repository." `https://github.com/mosteo/yeison`, May 2022.

[5] A. R. Mosteo, "Use (and abuse?) of Ada 2022 features in designing a JSON-like data structure." `https://fosdem.org/2022/schedule/event/ada_2022_json_like/`, 2022. FOSDEM.

Getting Started with AdaWebPack

Max Reznik
Private entrepreneur, Zaporizhzhia, Ukraine; Tel: +38 050 106 6648; email: reznikmm@gmail.com

Abstract

This article introduces the AdaWebPack project recently presented at the 12th Ada Developer Room at FOSDEM 2022 [3]. The AdaWebPack project aims for providing a toolchain and Ada libraries to enable developing of web applications to be executed in a web browser.

Keywords: Ada, WebAssembly, LLVM.

1 Introduction

A web application is application software that is accessed by the user through a web browser mainly with an active network connection. As the Internet spreads and the capabilities of web browsers develop, web applications become more and more powerful and attractive. Web applications have several advantages. Web applications are very portabe, because they run on a wide range of devices and operating systems. It's easy to keep web applications uptodate as they served from a web server. They are easy to use because may not require installation. Web applications can reach anyone, anywhere, on any device with a single codebase.

Web browsers are quite capable right today. They provides reach APIs such as WebGL for 3D visualisation, WebVR for virtual reality, WebRTC for network communications and so on. Web browsers execute WebAssembly, so web applications could be developed in many languages, not just in JavaScript.

WebAssembly is a portable binary-code format as well as software interfaces to the host environment. The main goal of WebAssembly is to enable high-performance web applications. Many languages (including C/C++, Rust, Python, Java, Ruby, Go) have WebAssembly as a compilation target. The AdaWebPack project aims for adding Ada to this list.

In Oct 2019 AdaCore presented the GNAT LLVM project [1]. This project combines the GNAT Ada front-end with a LLVM code generator. The LLVM supports many instruction sets for real CPU/GPU, but it is able to generate WebAssembly also. This combination opens the way for Ada to the Web.

Besides GNAT LLVM based toolchain, AdaWebPack contains a customised GNAT Run Time library and Web API bindings.

2 Compiling to WebAssembly

GNAT LLVM toolchain includes llvm-gcc compiler. To compile an Ada code to WebAssembly you need

- specify wasm32 as a target to LLVM backend,
- point GNAT to a target dependent information, because it differs from the native one. This information is provided by AdaWebPack in the wasm32.atp file.

Corresponding command looks like this:

```
llvm–gcc –c ––target=wasm32 –gnateT=wasm32.atp ...
```

Since GNAT LLVM release, gprbuild is aware of it, so you can compile a whole project like this:

```
gprbuild ––target=llvm –P hello.gpr –cargs
   ––target=wasm32 –gnateT=wasm32.atp
```

3 Customized Ada Run-Time Library

At the current state WebAssembly doesn't allow any stack manipulation, so some Ada features are not supported yet. Given that the Posix API is not available in the WebAssembly environment, it is clear why the embedded GNAT runtime library was chosen to be adapted as the WebAssembly runtime library. It imposes the following restrictions:

- No exception handling (but local)
- No nested subprogram access values
- No tasks and protected objects

To allow dynamic memory allocation a simple implementation of TLSF allocatior was written. If the default memory region is exhausted then allocatior requests new memory region from the host environment.

4 Provided Ada Libraries

4.1 WASM auxiliary packages

JavaScript object space is securely isolated from the WebAssembly program, so no JavaScript object can be accessed directly. To simplify JavaScript integration AdaWebPack provides several auxiliary packages under WASM hierarchy (See Figure 1).

They include WASM.Objects package to map JavaScript objects to integer identifiers as they are passed to Ada space. WASM.Methods lets user call a method on a such object, while WASM.Attributes provides read and write access to object's attributes.

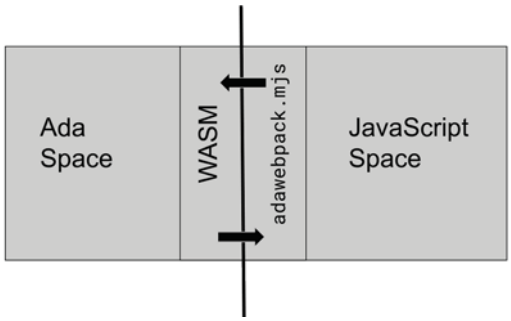


Figure 1: JavaScript and Ada space separation

4.2 WebAPI: Web.Strings

It seems that the string is the most used type for Web applications. To simplify exchange strings with JavaScript world AdaWebPack provides a new string type (Web_String) with primitives to contert it to and from Wide_Wide_String.

```
function "+" (X : Wide_Wide_String)
  return Web.Strings.Web_String
    renames Web.Strings.To_Web_String;

X : Web.HTML.Elements.HTML_Element :=
  Web.Window.Document.Get_Element_By_Id
    (+"toggle_label");
```

4.3 WebAPI: Web.DOM

The Web.DOM binding let the user access to a Web document in a usual way. This interface provides Documents, Nodes, Elements, Events types and so on. The user can search for an element with Get_Element_By_Id function, traverse element tree, create, update and delete nodes, subscribe to DOM events. To implement an event handler the user provides an implementation of Event_Listener interface.

```
type Listener is limited new
  Web.DOM.Event_Listeners.Event_Listener
    with null record;

overriding procedure Handle_Event
 (Self  : in out Listener;
  Event : in out Web.DOM.Events.Event'Class);

L : aliased Listener;
```

4.4 WebAPI: Web.HTML

This package hierarchy provides a binding to HTML elements such as Buttons, Inputs, Windows and so on.

```
procedure Initialize_Demo is
  B : Web.HTML.Buttons.HTML_Button_Element
    := Web.Window.Document.Get_Element_By_Id
        (+"toggle_button").As_HTML_Button;

begin
  B.Add_Event_Listener (+"click", L'Access);
  B.Set_Disabled (False);
end Initialize_Demo;
```

4.5 WebAPI: Web.GL

The WebGL is an API for rendering interactive 2D and 3D graphics within a web application. This very impressive technology could be used to create games, 3D visualisation, content creation and so on.

AdaWebPack provides a basic binding for a subset of WebGL API.

4.6 WebAPI: Web.Sockets, Web.XHR

The other two packages are for network communication. The WebSocket protocol provides full-duplex communication channels over a single TCP connection and lets the browser interact with the server and vice versa.

XMLHttpRequest is an API to make HTTP requests such as GET, POST, DELETE, etc. This is widely used to get data from the server, send a user input to the server, check user's permissions and so on.

5 Web Application architecture

Just as an any Ada program, Ada Web application is executed in two phases - elaboration and main procedure execution. During elaboration the application initializes internal data, creates event listeners and subscribes them to required events. To emphasize that the browser controls the application execution, the body of the main program is left empty, so it does nothing. After than the browser drives event loop and web application gets events and reacts on them.

The AdaWebPack includes several examples to help newcomers understand basic ideas.

6 AdaWebPack distribution

The AdaWebPack sources are published on GitHub under an open source license. Everyone can build it from sources, but this process could be complicated because it involves sources from several other projects: GNAT frontend sources (part of GCC), GNAT LLVM, LLVM. All these projects evolve on their-own pace, so it could be hard to get consistent sources.

The AdaWebPack repository provides a binary release to help with this issue. Currently, there are binaries for Fedora Linux, Ubuntu and MSYS2 package for Windows.

7 Future work

Currently, AdaWebPack is at an early stage of development. It capable to build a rather simple web applications, but writing them requires a lot of manual coding. It's hard because it involves synchronous update of Ada, HTML, CSS, WebGL shaders code. We are looking a way to simplify this by developing a higher-level widget toolkit, but this requires more expirience, ideas and experiments than we have now.

Current WebAPI binding is rather small. Extending it it's trivial, but requires a lot of work for a massive API. So we are developing a WebIDL-to-Ada converter to automate this work.

Another interesting idea is to use Ada for a multitier development. Suppose you write a single Ada program and a tool then splits it to client and server parts. These parts communicate each other with Distributed System Annex primitives implemented over WebSocket, WebRTC or XMLHttpRequest API.

We invite everyone intrested in Web application development give AdaWebPack a try, and let us know via Issues and Pull Requests on GitHub how it works for you.

Repository link: https://github.com/godunko/adawebpack

References

[1] A. Charlet, "Combining gnat with llvm." https://blog.adacore.com/combining-gnat-with-llvm, 2019.

Overview of Ada GUI

Jeffrey R. Carter
PragmAda Software Engineering, https://github.com/jrcarter/

Abstract

This is a summary of the presentation of the same title made in the Ada devroom at FOSDEM 2022. Traditional GUIs require registering callbacks and then giving up the program's thread of control to the GUI. This results in an unnatural programming style that runs counter to the way people typically learn to read programs. Ada GUI uses an alternative approach suited to a concurrent language.

Keywords: Ada, GUI.

1 Introduction

Traditional GUI frameworks require the program to create widgets, register callbacks, and then give up the thread of control. This is a response to the inherent parallelism of a GUI by sequential languages. Programs are unintuitive to design, write, and understand. In a concurrent language such as Ada, the GUI can have its own task. This allows the GUI to preserve the intuitive way of writing and understanding programs. Ada GUI provides an interface for such a GUI. There could be multiple implementations of this interface. There is an all-Ada example implementation, that uses a browser as a platform, supplied with the interface on Github [1]. The use of a browser as the platform makes the implementation very portable.

2 Traditional GUI Framework

In a traditional GUI framework, the program creates widgets, registers callbacks, and gives up its thread of control. This is due to the fact that a GUI is inherently parallel: the user and the program run on completely separate processors. The GUI needs a thread of control in order to be able to respond to user actions. In a sequential language, that thread must come from the program. But there must be a mechanism for the program's code to execute in response to the user's actions. This is provided by the GUI calling subprograms that the program has registered with the GUI.

This approach has a number of consequences. What of program code that needs to run independently of the GUI? There is the question of what happens if there is a GUI event while a callback is executing. Nor is this purely an academic consideration: the Gnoga version of Mine Detector [2] has to take action to deal with the possibility of a GUI event occuring during a callback.

There is also an effect on program design, implementation, and understanding. People learn to think of programs as sequential things: First it does X, then Y, and then Z. Consider a typical program that does not use a GUI:

```
with Ada.Text_IO;

procedure No_GUI is
   -- Empty
begin -- No_GUI
   All_Commands : loop
      Ada.Text_IO.Put(Item => "Enter g, h, or q:  ");
      One_Command : declare
         Command : constant String :=
            Ada.Text_IO.Get_Line;
      begin -- One_Command
         if Command'Length > 0 then
            case Command (Command'First) is
            when 'g' =>
               Ada.Text_IO.Put_Line(Item => "Greetings");
            when 'h' =>
               Ada.Text_IO.Put_Line(Item => "Hello");
            when 'q' =>
               Ada.Text_IO.Put_Line(Item => "Quitting");
                exit All_Commands;
             when others =>
                null;
             end case;
         end if;
      end One_Command;
   end loop All_Commands;
end No_GUI;
```

This program is designed, written, and read top-to-bottom, with X, Y, and Z corresponding to outputting a prompt, getting a line, and processing the line.

Compare No_GUI to a similar program that uses a GUI:

```
with TGF;

procedure With_GUI is
   G      : TGF.Button;
   H      : TGF.Button;
   Q      : TGF.Button;
   Output : TGF.Text_Box;

   procedure G_Click is
      -- Empty
   begin -- G_Click
      Output.Set_Text (Text => "Greetings");
   end G_Click;

   procedure H_Click is
      -- Empty
   begin -- H_Click
      Output.Set_Text (Text => "Hello");
   end H_Click;
```

```
   procedure Q_Click is
     -- Empty
   begin -- Q_Click
     Output.Set_Text(Text => "Quitting");
     TGF.End_GUI;
   end Q_Click;
 begin -- With_GUI
   TGF.Set_Up(Title => "Silly Example");
   G.Create
     (Text         => "g",
      Click_Action => G_Click'Access);
   H.Create
     (Text         => "h",
      Click_Action => H_Click'Access);
   Q.Create
     (Text         => "q",
      Click_Action => Q_Click'Access);
   Output.Create;
   TGF.Main_Loop;
 end With_GUI;
```

(TGF stands for Traditional GUI Framework; it is imaginary.) This can be read top-to-bottom, but even understanding how package TGF works, that doesn't give much information. Instead, one must think in terms of "when the user does this, the GUI calls that". The callbacks normally have to modify the internal state of the program, and in many cases modify the GUI as well. Even for fairly modest programs, the number of combinations can easily exceed the ability of the programmer or reader to keep them in mind. The checks within a callback to deal with the need for different actions for different cases can also become quite complex.

3 Ada-Gui philosophy

The main idea behind Ada GUI is to use the features of Ada to preserve the intuitive way of writing and understanding programs. Ada is a concurrent language, so the GUI can have its own task to respond to events, eliminating the need to register callbacks and give up the thread of control. The GUI can put events on a protected queue; the program takes events from the queue when appropriate for the program's logic. Significantly different cases can be handled by separate parts of the program, eliminating the combinatorial explosion and complex checks of handling them all from the same set of callbacks. There is no need to worry about when or how frequently events occur.

Widgets are created by functions that return the ID of the new widget. Operations take the ID of the widget to act on, with heavy use of preconditions to make sure that operations are not applied to inappropriate widgets. Events from the event queue contain the ID of the widget that generated the event.

4 Example

A simple example of the use of Ada GUI is provided by a Luhn-checksum generator. Luhn checksums are used for the last digit of credit and debit card numbers and the like.

```
-- A program for generating Luhn checksums
-- with an Ada_GUI UI
-- An Ada_GUI demo prograM
-- Copyright (C) 2022 by PragmAda
-- Software Engineering
-- Released under the terms of the BSD
-- 3-Clause license; see
-- https://opensource.org/licenses
--
with Ada.Exceptions;
with Ada.Text_IO;

with Ada_GUI;

procedure Luhn_Gen is
  Input   : Ada_GUI.Widget_ID;
  Err     : Ada_GUI.Widget_ID;
  Checksum : Ada_GUI.Widget_ID;
  Gen     : Ada_GUI.Widget_ID;
  Quit    : Ada_GUI.Widget_ID;

  procedure Generate;
  -- Obtain input from the GUI and
  -- calculate the checksum

  Err_Msg : constant String := "Enter some digits";

  procedure Generate is
    subtype Digit is Character range '0' .. '9';

    function Reversed (Value : String) return String;
    -- Reverses Value.

    function Squeezed (Value : String) return String;
    -- Keeps the Digits of Value and discards any other
    -- characters.

    function D2N (D : Digit) return Natural is
      (Character'Pos (D) - Character'Pos ('0') );

    function Reversed (Value : String) return String is
      Result : String (Value'Range);
      Last   : Natural := Value'Last;
    begin -- Reversed
      if Value = "" then
        return "";
      end if;

       Swap : for First in Value'First .. Value'First +
                (Value'Length - 1) / 2
      loop
        Result (First) := Value (Last);
        Result (Last)  := Value (First);
        Last := Last - 1;
      end loop Swap;

      return Result;
    end Reversed;
    function Squeezed (Value : String) return String is
      Result : String (1 .. Value'Length);
```

```
      Last   : Natural := 0;
    begin -- Squeezed
      All_Chars : for I in Value'Range loop
        if Value (I) in Digit then
          Last := Last + 1;
          Result (Last) := Value (I);
        end if;
      end loop All_Chars;
      return Result (1 .. Last);
    end Squeezed;

    Forward : constant String := Input.Text;
    Value   : constant String := Squeezed
              (Reversed (Forward) );
    Sum : Natural := 0;
    D   : Natural;
  begin -- Generate
    Err.Set_Text (Text => "");
    Err.Set_Visibility (Visible => False);
    Checksum.Set_Text (Text => "");

    if Value'Length = 0 then
      Err.Set_Visibility (Visible => True);
      Err.Set_Text (Text => Err_Msg);
      return;
    end if;

    All_Digits : for I in Value'Range loop
      D := D2N (Value (I) );
      if I rem 2 = 1 then
        D := 2 * D;
        if D > 9 then
          D := D - 9;
        end if;
      end if;

      Sum := Sum + D;
    end loop All_Digits;

    Checksum.Set_Text (Text => Integer'Image (
                              (9 * Sum) rem 10) );
  exception -- Generate
  when E : others =>
    Ada.Text_IO.Put_Line (Item => "Generate: " &
         Ada.Exceptions.Exception_Information (E) );
  end Generate;

  Event : Ada_GUI.Next_Result_Info;

  use type Ada_GUI.Event_Kind_ID;
  use type Ada_GUI.Widget_ID;
begin -- Luhn_Gen
  Ada_GUI.Set_Up (Title =>
                    "Luhn Checksum Generator");

  Input := Ada_GUI.New_Text_Box (Label => "Input :",
           Placeholder => Err_Msg);
  Err := Ada_GUI.New_Background_Text
         (Break_Before => True);
  Err.Set_Foreground_Color (Color =>
         Ada_GUI.To_Color (Ada_GUI.Red) );
  Err.Set_Visibility (Visible => False);
  Checksum := Ada_GUI.New_Text_Box (Label =>
        "Checksum :", Break_Before => True);
  Gen := Ada_GUI.New_Button (Text => "Generate",
        Break_Before => True);
  Quit := Ada_GUI.New_Button (Text => "Quit",
        Break_Before => True);

  All_Events : loop
    Event := Ada_GUI.Next_Event;

    if not Event.Timed_Out then
      exit All_Events when Event.Event.Kind =
           Ada_GUI.Window_Closed;

      if Event.Event.Kind = Ada_GUI.Left_Click then
        if Event.Event.ID = Gen then
          Generate;
        end if;

        exit All_Events when Event.Event.ID = Quit;
      end if;
    end if;
  end loop All_Events;

  Ada_GUI.End_GUI;
exception -- Luhn_Gen
when E : others =>
  Ada.Text_IO.Put_Line (Item =>
Ada.Exceptions.Exception_Information (E) );
end Luhn_Gen;
```

The reader will note that the intuitive way of reading a program is preserved. First the program sets up the GUI, then obtains GUI events and responds to them.

For a more complex example, the reader may want to see the Ada-GUI version of MP, a music player [3]. It demonstrates using separate parts of the program to deal with different operating modes. There is also an Ada-GUI version of Mine Detector [2] which does not have to worry about event arrival timing.

5 Implementation

Ada GUI comes with an example implementation derived from Gnoga, with many simplifications and modifications. It uses a browser as its platform, making programs that use it very portable. The implementation is also all Ada, making it easier for an Ada software engineer to understand. Other implementations are possible.

6 Summary

The traditional GUI framework results in an unintuitive approach to program implementation that can be difficult to understand. The main reason for Ada GUI is to provide a GUI that preserves the intuitive way of writing and understanding programs. Ada GUI is conceptually quite simple and straightforward, and the example implementation

is all Ada. Ada-GUI programs should be very portable, even to versions with alternative implementations.

References

[1] J. R. Carter, Ada-GUI repository, https://github.com/jrcarter/Ada_GUI.

[2] J. R. Carter, Mine-Detector respository, https://github.com/jrcarter/Mine_Detector.

[3] J. R. Carter, MP respository, https://github.com/jrcarter/MP

The Outsider's Guide to Ada Lessons from Learning Ada in 2021

Paul Jarrett

Abstract

Ada often gets written off as an old and obscure language, though few have first-hand experience with it. In this presentation, an Ada newcomer provides an objective overview of the language's major elements and features. A C++ software engineer with experience in over a dozen programming languages, Paul introduces Ada to fellow programmers using the C family vernacular.

Get introduced to the modern techniques and tools used for open source development with Ada. See how the Alire tool allows easy cross-platform development, as used by AdaCore's Ada Crate of the Year 2021 winner.

Learn how the language breaks into four syntax categories, which form a core and three opt-in sublanguages. Discover how structural elements come in four flavors, and how types and subprograms drive design. Also included is an illustration of fine tuning and low level control with aspects and attributes. The talk concludes by showing tasking in action from an open source project, and the various ways the language focuses on describing programmer intent.

This talk focuses on concepts, making it a great complement to Jean-Pierre Rosen's more detailed talk, "Introduction to Ada for Beginning and Experienced Programmers."

Watch the video [1] to see an outsider's perspective of Ada, or to introduce programmers to the language.

References

[1] https://fosdem.org/2022/schedule/event/ada_outsiders_guide/

Proving the Correctness of GNAT Light Runtime Library

Yannick Moy and Claire Dross
AdaCore, 46 rue d'Amsterdam, 75009 Paris; email: moy@adacore.com, dross@adacore.com

Abstract

The GNAT light runtime library is a version of the runtime library targeted at embedded platforms and certification, which has been certified for use at the highest levels of criticality in several industrial domains. It contains around 180 units focused mostly on I/O, numerics, text manipulation, memory operations. We have used SPARK to prove the correctness of 40 of them: that the code is free of runtime errors, and that it satisfies its functional specifications.

Keywords: SPARK, runtime, proof.

1 Introduction

As a programming language, Ada offers a number of features that require runtime support, e.g. exception propagation or concurrency (tasks, protected objects). The GNAT compiler implements this support in its runtime library, which comes in a number of different flavors, with more or less capability. The GNAT light runtime library is a version of the runtime library targeted at embedded platforms and certification, with an Operating System or without it (baremetal). It contains around 180 units focused mostly on I/O, numerics, text manipulation, memory operations.

Variants of the GNAT light runtime library have been certified for use at the highest levels of criticality in several industrial domains: avionics (DO-178), space (ECSS-E-ST40C), railway (EN 50128), automotive (ISO-26262). Details vary across certification regimes, but the common approach to certification used today is based on written requirements traced to corresponding tests, supported by test coverage analysis. Despite this strict certification process, some bugs were found in the past in the code. An ongoing project at AdaCore is applying formal proof with SPARK to the light runtime units, in order to prove their correctness: that the code is free of runtime errors, and that it satisfies its functional specifications. So far, 40 units (out of 180) have been proved, and a few bugs fixed along the way (including a buffer overflow).

2 Not all Bugs are Shallow

But first, let's consider a motivating example of why one may need formal proof to get confidence in the correctness of runtime units. Back in 2012, the late great programmer (and co-founder of AdaCore) Robert Dewar implemented runtime support for big integers in the GNAT compiler, in order to allow intermediate arithmetic computations without overflows (say, if you compute (A * B) / C but (A * B) might overflow, this allows you to tell the compiler to compute (A * B) / C with big integers, so that only the final result has to fit in a machine integer). The most complex function was the division between big integers, for which he implemented algorithm D by Donald Knuth from The Art of Computer Programming Vol 2, 2nd Edition - 1981, section 4.3.1. One of the code reviewers reported a possible integer overflow in a test, when computing the quantity ((u (j) & u (j + 1)) - DD (qhat) * DD (v1)) * b. Robert was initially not worried, given that this closely followed Knuth's published algorithm, but got concerned when it was shown that the overflow could be exercised! So that the computation of (A * B) / C with A = 18446744069414584318, B = 4294967296 and C = 18446744069414584319 was giving the result 2147483648 instead of the correct 4294967295.

Thankfully, we were not the first to spot the bug, which had already been corrected in 1995. Here is the relevant section of errata of TAOCP Vol 2, 2nd Edition, replacing the buggy test with new code (to the right of the strange arrow):

Page 258 first three lines of step D3 — 28 Sep 1995
If $u_j = v_1$, ... latter test determines $\leadsto$ Set $\hat{q} \leftarrow \lfloor (u_j b + u_{j+1})/v_1 \rfloor$ and $\hat{r} \leftarrow (u_j b + u_{j+1}) \bmod v_1$. Now test if $\hat{q} = b$ or $v_2\hat{q} > b\hat{r} + u_{j+2}$; if so, decrease $\hat{q}$ by 1, increase $\hat{r}$ by v_1, and repeat this test if $\hat{r} < b$. [The test on v_2 determines

In fact, with this patch, the rewritten test might still lead to an overflow! This was detected a decade later, in 2005. Here is the relevant section of errata of TAOCP Vol 2, 3rd Edition, changing the comparison operation:

▸**Page 272** line 2 of step D3 — 03 Feb 2005
test if $\hat{q} = b \leadsto$ test if $\hat{q} \geq b$

After careful code reviews, we convinced ourselves that the new version was correct, but, already at the time, we wondered whether this could be proved using SPARK tools (after all, the GNAT compiler is written itself in Ada, so we could hope to prove part of it). That was not possible at the time, but we kept it as a future challenge.

Of course, the same algorithm may get implemented numerous times in a given application, and GNAT was no exception. There were two other implementations of algorithm D in GNAT, one in uintp.adb for arbitrary-precision computation at compile time, and one in s-arit64.adb for runtime support of fixed-point arithmetic. In the specific context of these two other implementations, we found no clear bug: the fixes were propagated to uintp.adb which was using a similar test, while s-arit64.adb used a different comparison which could not overflow. But given

that the 1st Edition of Vol 2 was published in 1969, there must be hundreds of implementations of this algorithm out there that did not apply later fixes and are still incorrect!

Five years later, in 2019, our interest in the implementation of algorithm D in s-arit64.adb was raised by a remark of an external auditor, as part of the certification of this runtime unit for use in space. The auditor noted the high complexity of this function and asked for the addition of more comments in the code to be able to assess its correctness. Prompted by this request, we reviewed again this implementation and discovered that the code failed to raise an exception in a case where it should have done so (because the result of the division was too large), and that the code of another function in that unit contained two possible integer overflows when converting between signed and unsigned values. Thankfully, none was critical, because the former concerned a case of incorrect inputs, and because the overflows in the latter were silent in the runtime at that time (the runtime was compiled without runtime checking). Still, that was a close-enough call for us to wish that we could increase our confidence in the correctness of this code through proof.

3 Going Beyond Eyeballs (i.e. Reviews)

And this is what we did in the summer of 2021! Our intern Pierre-Alexandre Bazin used SPARK to prove that s-arit64.adb was correctly implementing all its functions: there were no possible runtime errors in the code, and all the functions implemented their specification faithfully. This required expressing the specification as contracts in SPARK, that is, preconditions and postconditions, like here for the function Scaled_Divide implementing algorithm D:

```
procedure Scaled_Divide
  (X, Y, Z : Double_Int;
   Q, R    : out Double_Int;
   Round   : Boolean)
with
  Pre  => Z /= 0
    and then In_Double_Int_Range
      (if Round then Round_Quotient (Big (X) * Big (Y), Big (Z),
                                     Big (X) * Big (Y) / Big (Z),
                                     Big (X) * Big (Y) rem Big (Z))
       else Big (X) * Big (Y) / Big (Z)),
  Post => Big (R) = Big (X) * Big (Y) rem Big (Z)
    and then
      (if Round then
         Big (Q) = Round_Quotient (Big (X) * Big (Y), Big (Z),
                                   Big (X) * Big (Y) / Big (Z), Big (R))
       else
         Big (Q) = Big (X) * Big (Y) / Big (Z));
```

The postcondition uses big numbers to express that the resulting quotient Q is the mathematical operation (X * Y / Z) and the resulting remainder R is the rounded value of the mathematical remainder. The precondition states that these values for Q and R should fit in the machine integer type Double_Int. See the code for the definition of the ghost functions Round_Quotient and Same_Sign which are used to define this contract.

The implementation of Scaled_Divide was slightly modified to make it provable, but more critically, Pierre-Alexandre had to use quite a lot of ghost code to guide automatic provers, including basic arithmetic lemmas to enunciate and prove mathematical properties, as well as a number of more complex lemmas to isolate parts of the proof, and a few intermediate assertions to simplify and share the proofs between provers.

Encouraged by this initial success, we have added contracts expressing the full functional specification of many other units in the GNAT light runtime, and proved with SPARK that the code correctly implemented these contracts. This includes units for character and string handling (like a-strsup.ads/a-strsup.adb), units for support of language attributes ‘Width, ‘Value and ‘Image (like s-widthu.ads/s-widthu.adb, s-valueu.ads/s-valueu.adb and s-imageu.ads/s-imageu.adb), support for exponentiation (like s-exponn.ads/s-exponn.adb). We have so far proven 40 such units, and, along the way, we have discovered and fixed a few cases of overflow check and range check failures, one of which could lead to a buffer overflow on a runtime built without runtime checks. As you can see from the source files, that required adding many specifications (around 400 preconditions and 500 postconditions) and ghost code (around 150 loop invariants, 400 assertions, 300 ghost entities), and the daily proof takes 1.5 hours on a Linux server with 36 cores.

Most remaining units remain out of reach for SPARK today, either because they rely on an untyped memory model (converting between raw Address values and typed pointers) or because they require precise reasoning on bitwise floating-point representation. Most units that use Address-to-pointer conversions use very simple algorithms, and those that manipulate floating-point values are direct translations in Ada of either reference C implementations or textbook algorithms, which increases confidence in their correctness. Our vision for the future is to both maintain the automatic proof of the 40 units proved so far as the analysis tool and provers get updated, so that we can benefit from the associated assurance in certification, and to grow the set of proved units as SPARK language allows more constructs and tooling improves.

The fact that this effort has not led to the discovery of serious bugs is a testament to the quality of the GNAT light runtime code, which has been submitted to a very high level of scrutiny in the past 20 years as it has been certified to the highest levels of multiple certification standards for avionics, railway, space, etc. Proof with SPARK is a new way to achieve this high level of assurance, with stronger guarantees about the absence of whole classes of errors, and about the faithfulness of all code paths to the specification.

4 Conclusion

Linus’s law states that “given enough eyeballs, all bugs are shallow”. Our past experience with subtle bugs remaining present in code after decades of detailed inspection by experts, including in the context of certification, tells us that this is not true of all bugs. Formal proof may provide a cost-effective way to gain the assurance that indeed subtle bugs escaping reviews and testing are not present in the software.

This work was presented in the Ada devroom at FOSDEM 2022. [1]

References

[1] https://fosdem.org/2022/schedule/event/ada_proving_gnat_light_runtime/

Implementing a Build Manager in Ada

Stéphane Carrez
Issy Les Moulineaux, France; email: Stephane.Carrez@protonmail.com

Abstract

A build manager is a tool used to automate the building, testing and delivery of software projets. Porion is a new build manager that was first presented at the FOSDEM 2022 event in the "Continuous Integration and Continuous Deployment Developer Room" [1] and in the "Ada Developer Room" [2]. This paper is a summary of these two presentations and it focuses on the complexity of designing a build manager. It highlights some security issues that apply to a build manager and its implementation. It explains the overall architecture that was chosen and the reason of the choice. Finally it presents the Ada generation tools that have been used in this project.

Keywords: build manager, continuous integration, continuous delivery, code generation.

1 Introduction

After having used Jenkins [3] with more than 30 projects during 8 years, it was time to switch to another build manager. Jenkins is a great build manager but it is written in Java and suffers from several performance and security issues. With my Jenkins configuration, the server is using more than 1.3Gb of resident memory. Jenkins also has regular security vulnerabilities due to the numerous plugins it is composed of and to the use of Java. The Jenkins community is however very reactive and these vulnerabilities are fixed quickly.

It was time to get rid of Java and put in place a better and safer design in Ada. Security is an important aspect of the project and it must be taken seriously from the beginning. As far as functionalities are concerned, it is not necessary to have all the features provided by Jenkins but having a command line interface and a web interface were two important aspects. Last is the performance of the final build manager which must deliver a fast and responsive web interface.

This article is structured as follow. Section 2 presents the project to highlight some of its concepts and features. The project architecture is described in section 3. Section 4 presents the UML and code generation tool which are used. Section 5 describes how some resources are embedded in the Ada program with the help of another code generation tool, before concluding in section 6.

2 Overview

Porion [4] is a continuous integration server and agent that helps automate the building, testing and deployment of software development projects written in any language.

The name of the project comes from the north of France and was used in coil mines to designate the master miner. The word means "leak" in English or "poireau" in French. Indeed the "porion" sits down and waits for others to do the job. This is what we expect for a build manager: wait, build and verify that the work is done correctly.

2.1 Porion concepts

Before looking at the project features and architecture, it is necessary to look at the different concepts used by a build manager. These concepts are common to most build managers, they are illustrated by figure 1.

First the build manager defines a notion of project that must be monitored and built. From that project the build manager must have access to the sources to check whether they are modified and start new builds. In most cases, sources are stored using a source control system such as Git, Subversion, Mercurial, Bazaar, CVS or others. But it is also possible to use a tar or a zip file to get access to the sources.

Figure 1: Porion concepts

To build a project we are going to use a build recipe that describes all the steps to build the project. A same project can be built by using different recipes and therefore different build steps. A build recipe can be setup to configure the build with code coverage analysis, another one to build with optimized options for a production release and a third one could enable or disable some specific project feature.

Having a build recipe, the build manager must decide to execute the associated build steps on a build node. The build node can be the machine where the build manager is running but it can also be a remote server dedicated to the build. The build node has specific properties such as the operating system or CPU architecture. For some projects it is useful to ask the build manager to build the project on different build nodes, each having different properties.

When executing the build recipes on a build node we obtain some build results. The build results are composed of the production of the build, test execution results, code coverage analysis and all the logs associated with execution of the build recipes.

2.2 What Porion must do?

A build manager has a lot of work to fulfill its job. Assuming the build manager has all the project configuration, it must first probe the sources to detect changes in the project. Such mechanism is specific to each source control system. Having detected changes, it must schedule the builds according to the projects and the recipes and for that is uses a build queue.

The build manager must launch builds either locally or remotely on a build node. For this it will look at the build recipe and execute every step one by one. Such execution is synchronous meaning that the build manager has to wait until the recipe step has completed. During that execution, the build manager must also track the execution and record build logs.

After the build recipes are executed, the build manager must extract some build results either from the build logs or from some files produced by running the build steps. The build manager can extract unit test results, code coverage results.

Once the build is finished, the build manager must publish some build summary and provide various reports.

2.3 What Porion must protect?

A build manager has access to sensitive data and we must not neglect the security aspects of such tool. The first set of data to protect is the source files and how they are fetched from the source control system.

Another set of sensitive information is the credentials used by the build manager. Some credentials are used to access and retrieve the source code of the projects. Some other credentials are used to connect and execute recipes on build nodes. The API secret keys which are used to interact with external tools is another form of credentials that must be protected.

A last class of secret information concerns the passwords used by certificates when a build must be signed. For example, an Android application must be signed with a certificate before being uploaded to the Google Play Console. Such certificate must be protected by a password and we don't want to expose that password to malicious users.

The build manager provides a web access in order to display build results. Web application security is often underestimated. This is an area where security is an important concern and using a secure framework can help reduce application vulnerabilities.

2.4 Project numbers

The project was started in May 2021 and took around 260 hours so far. It contains 32000 lines of Ada 2012 but half of them is in fact generated by some code generator. The program has only 43 Ada packages completed by 30 Ada private packages making it rather small and reasonably complex project.

3 Porion architecture

The Porion build manager is composed of a command line agent and a web server. The command line agent provides the basic commands to manage projects, recipes, build nodes, executing build recipes and collecting build results. On its side, the web server provides a simple dashboard that shows the projects and their build results. Both of them are written in Ada 2012 and they share almost the same Ada components and libraries.

3.1 Porion library

The heart of the build manager is composed of a library that is shared by the command line agent and the web server. This library is composed of 39 packages that organize the different operations of the build manager. The Ada packages are organized around the different functionalities that the build manager must perform. The use of Ada private child package is used whenever possible to restrict and limit the visibility of the package. Doing this has been very helpful for the refactoring because we know beforehand whether the impact is limited or larger. All the packages are contained in the `Porion` root package. Figure 2 tries to give a graphical overview of some packages and their organization. A greyed box package cannot be used outside of its enclosing box because it is a private Ada package. Packages marked with a star are generated by a code generator described later.

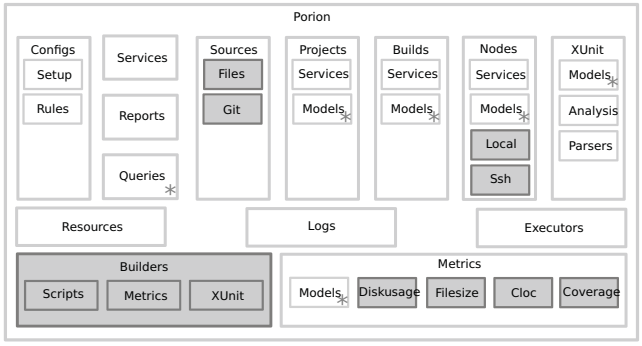


Figure 2: Porion library Ada packages

The sources of a project are controlled by the `Sources` child package. It contains two private child packages which are dedicated to the management of Git and tar or zip files. It is possible to add new source control systems by adding new private child package.

The projects and builds are managed by the `Projects` and `Builds` packages. The build nodes are represented by the `Nodes` package which contains specific private child packages dedicated to the management of these build nodes. The `Local` child package is dedicated to launching commands locally and the `Ssh` child package targets a build node with some ssh access.

The `Metrics` child package is dedicated at collecting various build metrics. Specific build metrics are implemented by a private child package. This allows to hide the implementation details of these build metrics and make sure that they are exposed using a common mechanism.

3.2 Porion agent

The Porion agent provides a set of commands that allow to control the projects, their configuration, define their recipes and execute build recipes. Each command has a specific name and dedicated options. The list of command example below is used to setup the workspace directory for hold the database and build area. The `add` command registers the `aflex` Git project, the `set` command configure some project variable.

```
porion init /build/porion
porion add https :// github.com/Ada-France/aflex.git
porion set aflex~main filter_rules git,make,gprbuild,cloc
```

The `build` command checks for changes on the project and build it. Last, the `list` and `info` commands report some information on the build or on a specific project.

```
porion build aflex
porion list -b
porion info -a aflex
```

Each command is implemented within its own Ada private child package as illustrated by figure 3. They are implemented by using the generic command framework provided by the Ada Utility Library [5]. That command framework handles the identification of the command to execute, the parsing of command arguments and the help support to give information about a command.

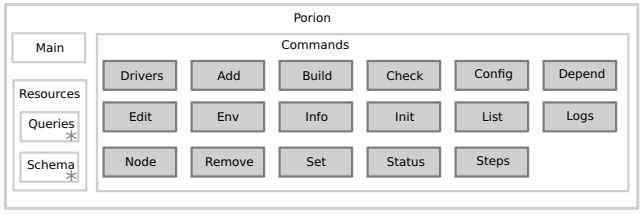


Figure 3: Porion agent Ada packages

To use command framework, the first step is to instantiate the `Util.Commands.Drivers` package with a context type and a parser package. The context type allows to define and give an application specific context when executing a command. The parser package defines the operations and types required to parse the command line. The `GNAT_Parser` handles the command line parsing by using the `GNAT.Command_Line` package which brings the support for numerous short and long option parser.

```
with Util.Commands.Drivers;
with Util.Commands.Parsers.GNAT_Parser;
private package Porion.Commands.Drivers is
  use Util.Commands;
  package Main_Driver is
   new Util.Commands.Drivers
    (Context_Type => Context_Type,
     Config_Parser => Parsers.GNAT_Parser.Config_Parser,
     Driver_Name => "porion");

  type Command_Type is abstract
    new Main_Driver.Command_Type with null record;
end Porion.Commands.Drivers;
```

Each command is implemented by a specific child package of the `Porion.Commands` package and it is defined by a specific Ada tagged type that must override an `Execute` procedure. The command framework will execute that procedure when the command name is given as program argument.

```
private package Porion.Commands.Info is

 type Command_Type is
   new Porion.Commands.Drivers.Command_Type with private;

 overriding
 procedure Execute
   (Command : in out Command_Type;
    Name    : in String;
    Args    : in Argument_List'Class;
    Context : in out Context_Type);
end Porion.Commands.Info;
```

An instance of the concrete `Command_Type` type is then declared in the `Porion.Commands` package body and registered to the driver with a name and a short description string. The driver uses the name to identify the command and the description for the help command.

```
package body Porion.Commands is
  Info_Command : aliased Porion.Commands.Info.Command_Type;
...
  Driver.Add_Command
    ("info",
     "report some information about a project or a build"),
     Info_Command'Access);
...
end Porion.Commands;
```

3.3 Porion server

The Porion web server is built on top of the Ada Web Application (AWA) [6] framework which was presented at the FOSDEM 2019 event [7]. Also based on the Ada Web Server [8], it can run on various servers including GNU/Linux, NetBSD and FreeBSD. AWA leverages Ada's safety features to provide a secure environment on top of which safe applications are built. AWA is based on several Java-like technologies such as Java Beans, Java Servlet, Java Server Faces and other standards such as OAuth2, REST and OpenAPI, all implemented in Ada.

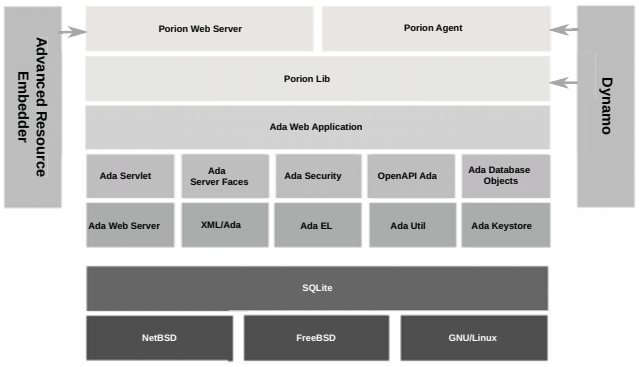


Figure 4: Porion architecture

Figure 4 shows the different Ada libraries that are provided or used by the AWA framework.

4 UML and code generation

To store and keep track of the different data, the Porion build manager uses a database. This gives enough flexibility and

scalability for the project management, tracking of project changes and representing the build results. The database has a specific schema for the representation of the data that must be stored. On the other hand, we need to access and store the data from an Ada 2012 program.

4.1 Database modeling

To easily describe the information that is stored in the database and accessed from Ada, a database model and a code generator is used. The database model is made by using UML [9] and the choice of UML was motivated by the fact that a UML class diagram is simple and flexible enough to represent a database table and its relations.

Figure 5 shows the development process. The first step is to design the database model either by using a UML tool such as ArgoUML [10] or by writing an XML or YAML description file. The second step is to generate the Ada mapping files by using the Dynamo [11] code generator. At the same time, the code generator can produce the SQL database schema. The last step is to write the application on top of the generated code that gives direct and simplified access to the database tables.

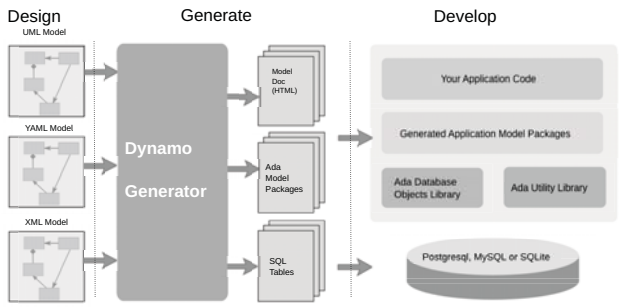


Figure 5: Dynamo code generator

The framework used by Porion to access the database is composed of the Ada Database Objects [12] library (ADO) and the Dynamo code generator. Most of the concepts developed for ADO come from the Java Hibernate [13] framework. The Ada Database Objects provides an "Object Relational Mapping" (ORM) for the Ada programming language. It allows to map database objects into Ada records and access database content easily. The library supports PostgreSQL, MySQL, SQLite as databases. On its side, Dynamo reads a model description and generates both the SQL schema and the Ada packages giving access to the database tables through the ADO framework.

4.2 Porion UML model

The Porion database model being described in UML with ArgoUML, it is composed of 19 tables organized in 5 packages. Furthermore the Dynamo code generator allows to map SQL queries in Ada vectors and these queries are organized in a separate package. At the end, the 6 generated Ada packages contain 14000 lines of Ada code.

UML mostly uses graphical notations to express the design of software components. The UML class diagram is one of such graphical notation. It describes classes by specifying a name for the class and defining a set of attributes that belong the class. Relations between different classes are expressed by drawing specific links between different classes. The UML class diagram can also describe operations provided by the class.

To describe some specific behavior, we are using UML stereotypes and apply them either on the UML class, UML attributes or UML relations. These stereotypes are recognized by the code generator and they control how the generated code or database SQL mapping is produced.

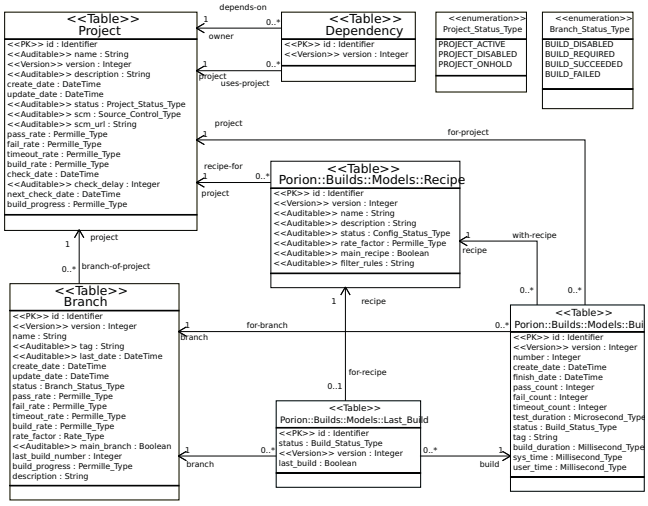


Figure 6: Projects, branches, dependencies, recipes and builds relations

Figure 6 shows several database tables with their relations. The `Project` class is marked with the `Table` stereotype to tell the code generator this is a database table and it must generate code for it. That table has a first `id` attribute whose type is `Identifier` and the `PK` stereotype indicates that this is a primary key. Next is the `name` attribute which is a `String` and has the `Auditable` stereotype. That stereotype indicates to the code generator to emit code to track changes of the attribute in a specific audit table.

A project can have several branches and this is tracked by the `Branch` table. A branch can belong to exactly one project and a project can have zero or several branches. The `Recipe` table describes the recipes to build the project and a project can have several build recipes.

The UML model and code generator allow the user to specify their own types. To track execution times and store them in a reasonable format, two types have been introduced: `Millisecond_Type` and `Microsecond_Type`. Both of them are represented as integer in the database but they are declared in the project as new natural types so that they cannot be mixed together. These two types are defined in a parent Ada package in a non generated Ada package. On its side the UML model contains some import definition that make these two types visible from the model.

4.3 Ada code generation with Dynamo

For each UML package, Dynamo generates an Ada package specification and a body with the definition of types and operations to represent the tables that are modelized in

UML. For the `Project` table, the code generator creates the `Project_Ref` tagged record and a `Project_Impl` private type. The first type is the type intended to be used by applications and it provides procedures and functions to operate on the project instance. Each project instance represents a single database row. Internally it uses a reference counter to the `Project_Impl` instance which holds all the record attributes.

```
with Ada.Containers.Vectors;
with ADO.Objects;
with ADO.Sessions;
with ADO.SQL;
package Porion.Projects.Models is

   type Project_Ref is new ADO.Objects.Object_Ref
      with null record;
   procedure Set_Name
    (Object : in out Project_Ref; Value : in String);
   function Get_Name
      (Object : in Project_Ref) return String;
   overriding
   procedure Save
      (Object  : in out Project_Ref;
       Session : in out ADO.Sessions.Master_Session'Class);
   package Project_Vectors
      is new Ada.Containers.Vectors
        (Index_Type   => Positive,
         Element_Type => Project_Ref,
         "="          => "=");
   procedure List
      (Object  : in out Project_Vector;
       Session : in out ADO.Sessions.Session'Class;
       Query   : in ADO.SQL.Query'Class);
    ...
private
    type Project_Impl is  ...
end Porion.Projects.Models;
```

The code generator generates an instantiation of the `Ada.Containers.Vectors` package and a `List` procedure that allows to retrieve a list of projects from the database. To select which projects are returned a query object can be configured to operate on the SQL `WHERE` clause that is generated to retrieve the rows from the database.

4.4 Using the generated Ada model

Using the generated Ada code to access the database is quite easy. The `Project` table is represented by the `Project_Ref` Ada tagged record and for each UML class attribute, the code generator provides a getter and a setter operation. The UML attributes are only accessed through function and procedures because the database framework must know when a value is modified in order to generate the correct SQL `INSERT` or `UPDATE` statement.

Populating a project information is easily done by calling the generated `Set_` procedures and then calling the `Save` procedure. Depending on whether the project existed or not in the database, the ADO runtime will either generate an SQL `INSERT` or `UPDATE` statement.

```
DB      : ADO.Sessions.Master_Session;
Project : Porion.Projects.Models.Project_Ref;
...
  Project.Set_Name (Name);
  Project.Set_Create_Date (Ada.Calendar.Clock);
  Project.Set_Status (Projects.Models.PROJECT_ACTIVE);
  Project.Set_Scm_Url ("https://gitlab.com/stcarrez/porion.git");
  Project.Set_Scm (SRC_GIT);
  Project.Save (DB);
```

4.5 Focus on the build queue scheduler

The build manager has a build queue that describes the project recipes that must be built. When registering a project, it is possible to define the dependencies between projects (this is represented by the `Dependency` class shown in figure 6). Such dependencies allow to trigger a build on a project when one of its dependent project was successfully built. The challenge is to reduce the number of builds and avoid building a project several times.

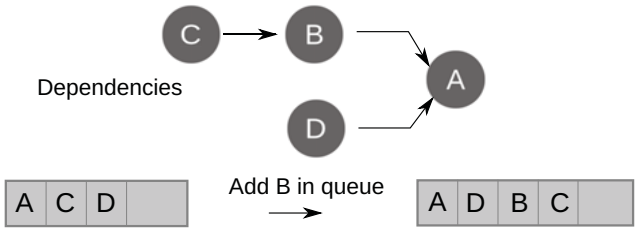


Figure 7: Project dependencies and build queue

Let's consider a set of 4 projects with a dependency tree of figure 7 such that project C depends on B which depends on project A and project D depends on project A. Let's assume that the build queue already contains project A, C and D, if the project B is modified, we have to add it in the build queue. If we add the project B at the end of the queue, the dependency tree will schedule again the build on project C. The job of the build scheduler is to avoid that and make sure that we insert the new project in the build queue before project C as shown in figure 7. However, at the same time we don't want to defer the execution of project D which does not depend on the new project.

To solve this problem, Porion has a build queue scheduler which looks at the build queue and the project dependencies when a new recipe is added in the build queue. The first step is to load the build queue from the database. The UML to Ada code generator has generated a `Build_Queue_Vector` type by instantiating the `Ada.Containers.Vectors` package. This allows us to call a `List` procedure that will execute an SQL `SELECT` statement and populate the Ada vector with each database row returned by the SQL statement.

```
with Porion.Builds.Models;
with ADO.Queries;
use Porion.Builds.Models;

Queues : Build_Queue_Vector;
Query  : ADO.Queries.Context;
   ...
  Porion.Builds.Models.List (Queues, DB, Query);
```

Having our build queue, the first step is to know whether the build queue already contains the build recipe we want to add. For this, we are going to write a simple `Contains` function that looks whether the recipe identifier is already contained in the list. The implementation is straightforward by the use of Ada 2012 quantified expression:

```
function Contains (List   : in Build_Queue_Vector;
                   Recipe : in Recipe_Ref) return Boolean
 is (for some Queue of List
       => Queue.Get_Recipe.Get_Id = Recipe.Get_Id);
```

To add the new recipe in the build queue, we only have to check whether it is already part of the queue in which case we have nothing to do. If it is not part of the queue, we only have to declare a `Build_Queue_Ref` object, setup the recipe, build node, creation date and append it to our list.

```
if not Contains (Queues, Recipe) then
  declare
    Queue : Porion.Builds.Models.Build_Queue_Ref;
  begin
    Queue.Set_Recipe (Recipe);
    Queue.Set_Node (Build_Node);
    Queue.Set_Create_Date (Ada.Calendar.Clock);
    Queues.Append (Queue);
  end;
end if;
```

The magic of the porion build scheduler is now fully implemented by using the Ada containers `Generic_Sorting` package. Because our `Build_Queue_Vector` is implemented on top of the Ada containers we only need to write a comparison function whose job is to decide whether a recipe in the queue must be built before another one. The complexity of the build scheduler is now moved to this comparison operation. Having it, we can instantiate the `Generic_Sorting` package to get our `Sort_Queue` package.

```
function "<" (Left, Right : in Build_Queue_Ref) return Boolean;

package Sort_Queue is
  new Build_Queue_Vectors.Generic_Sorting ("<" => "<");
```

Reordering the build queue according to project dependencies becomes as simple as calling the `Sort` procedure on our build queue vector.

```
Sort_Queue.Sort (Queues);
```

The last step is to save the result in the database. For this, we only have to update the build queue order for each element. This is done by calling the `Set_Order` procedure and then saving it by calling the `Save` procedure. If the build queue order was not changed the operation will do nothing but if it was changed, an SQL `UDPATE` statement is executed to update the order column in the database.

```
declare
  Order : Natural := 0;
begin
  for Queue of Queues loop
    Queue.Set_Order (Order);
    Queue.Save (DB);
    Order := Order + 1;
  end loop;
end;
```

4.6 Benefit of UML and Ada

When designing and implementing Porion, the UML database model was not correct at the first time. In the relatively short lifetime of this project, new requirements and ideas have been introduced and it forced changes in the database model. Several iterations were made on the model, sometimes to add new attributes, sometimes to introduce new classes and new relations. All these changes are easily done from the ArgoUML graphical tool. Very often it is useful to sketch a new set of classes and check if the new relations better fit the requirements and objectives. Such design trial phase can be made quickly by using UML.

From the UML database model, the Dynamo code generator generates quickly the Ada packages with the data types and operations to access the database. By using such code generator, changes to the UML model are easily propagated to the Ada source code. Because Ada is using strong typing, it happens that some change in the UML model breaks the compilation of the project. This occurs if a class is renamed, a class attribute is removed or has is type modified. Such compilation break occurred several time during the design phase of the project but each time it was easily fixed.

Refactoring the UML database model occurred several times. Sometimes to reorganize the classes in different packages. Sometimes to introduce new tables and relations. Thanks to Ada strong typing and the UML to Ada code generator, each time a refactoring was made and the compilation of the project was fixed, there was very little issues that remained.

The Ada representation of database tables and the use of standard Ada container packages to retrieve a subset of the database tables simplifies the implementation. Very often it is not necessary to deal with SQL statements to query, insert or update the database. It becomes possible to implement complex algorithm but very easily in Ada. The build queue scheduler is an example of such complex algorithm but its implementation has been made quite simple with the use of the Ada containers `Generic_Sorting` generic package.

5 Embedding resources in Ada program

An interesting issue came while setting up this project: how to install and configure the database on a fresh installation. Providing SQL schema files separate from the binary is of course a solution but it brings complexity and issues to package the solution. Instead, it would be nice that such SQL schema be part of the final binary program.

5.1 Advanced Resource Embedder

Incorporating files in a binary program can sometimes be a challenge. The “Advanced Resource Embedder” [14] is a flexible tool that collects files such as documentation, images, scripts, configuration files and generates a source code that contains these files. It is able to apply some transformations on the collected files:

- it can run a Javascript minifier such as `closure`,
- it can compress CSS files by running `yui-compressor`,
- it can compress files by running `gzip` or another tool.

Once these transformations are executed, it invokes a target generator to produce a source file either in C, Ada or Go language. The generated source file can then be used in the final program and taken into account during the compilation process of that program. At the end, the binary will contain the embedded files with their optional transformations.

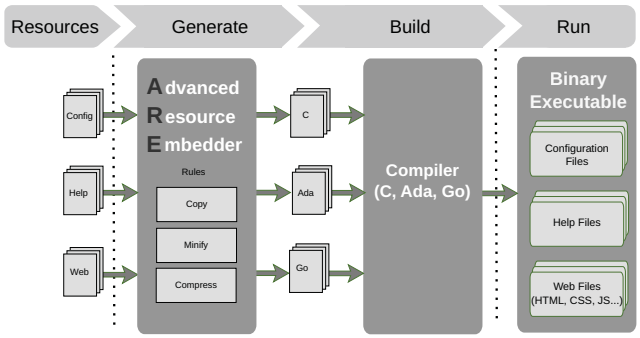


Figure 8: Advanced Resource Embedder

Figure 8 illustrates the process to use the tool. The first step is to describe the resources that must be embedded. The description is either made on command line arguments or by writing an XML file. The XML description gives more flexibility as it allows to define a transformation rule that must be executed on the original file before being embedded. This allows to minify a Javascript or CSS file, compress some files and even encrypt a file before its integration.

The second step is to run the command with the target language and rule description and give the tool a list of directories that must be scanned to identify the files that must be collected. The tool scans the directories according to the patterns that are given either on the command line or in the XML rule description. After identifying the files, the tool applies the rules and executes the transformations. The tool then invokes the target language generator that writes one or several files depending on the list of resources.

Once the files are generated, the generated code must be integrated in the build process as they are now part of the program sources. After building the program, it now embeds the resource files that were collected and optionally transformed.

5.2 Database schema integration in Porion

From our UML model, the Dynamo code generator has generated an SQL schema file. That schema file contains several SQL statements most of them are statements to create a database table but some of them also insert some data after a table is created. The resource embedder can embed the file as is and make it available either in the form of an Ada `String` type or a more low level data type such as the `Storage_Element`. For our use case, it is better to have one Ada `String` for each SQL statement. For Porion, the data storage representation is defined in a non generated Ada package. The generated Ada package will be a child of that package to make these types directly visible.

```
package Porion.Resources is
 type Content_Array is
   array (Natural range <>) of access constant String;
 type Content_Access is
   access constant Content_Array;
end Porion.Resources;
```

To control this code generation, an XML file is used to describe how the SQL schema file is integrated. The transformation that we are going to apply is to split the source file in separate lines on the "`;`" separator used by SQL and then apply several regular expression filters to remove SQL comments, remove duplicate spaces and drop line separators.

```
<package>
  <resource name='Porion.Resources.Schema'
            format='lines'>
    <line-separator>;</line-separator>
    <line-filter>[\r\n]</line-filter>
    <line-filter> /\*[^/]*\*/ </line-filter>
    <line-filter replace='␣'>[ \t][ \t]+</line-filter>
    <install mode='copy' strip-extension='yes'>
      <fileset dir="db/sqlite">
        <include name="create-porion_agent-sqlite.sql"/>
      </fileset>
    </install>
  </resource>
</package>
```

With this configuration, the tool will generate the `Porion.Resources.Schema` Ada package. The specification contains the declaration of the `Get_Content` function that can be used to retrieve the content of the embedded file. It returns an access type to the constant array of strings with each string being an access type to a single SQL statement that must be executed.

```
package Porion.Resources.Schema is
   Names : constant Name_Array;

   function Get_Content (Name : String) return
      Content_Access;
private
      ...
end Porion.Resources.Schema;
```

For the curious, the generated package body contains the content of files that are embedded basically as constant strings. When the resource is made available as a `String`, the content is stored by using an `aliased constant String`. Below is an extract of that body package:

```
package body Porion.Resources.Schema is
 L_1   : aliased constant String := "pragma␣synchronous=OFF";
 L_2   : aliased constant String := "CREATE␣TABLE␣IF␣NOT␣"
  & "EXISTS␣awa_audit_field␣(␣`id`␣INTEGER␣NOT␣NULL␣"
  & "PRIMARY␣KEY␣AUTOINCREMENT,␣`name`␣VARCHAR(255)"
  & "␣NOT␣NULL,␣`entity_type`␣INTEGER␣NOT␣NULL)";
  ...
 C_0 : aliased constant Content_Array :=
  (L_1'Access,
    ...
  );
 type Content_List_Array is array (Natural range <>)
   of Content_Access;
 Contents : constant Content_List_Array := (
   0 =>  C_0'Access);

 function Get_Content (Name : String) return Content_Access is
 begin
   return (if  Names (0).all = Name then Contents (0) else null);
 end Get_Content;

end Porion.Resources.Schema;
```

6 Conclusions and future work

Writing a build manager is a complex project and if we compare with a mature build manager such as Jenkins there is still a lot of work to do. The core of Jenkins contains more than 1000 Java classes and it has a complex architecture with more than 1800 plugins. Porion does not have this level of

complexity but by using Ada 2012 and some code generators it was possible to setup a functional build manager in a short development time frame. The Porion web server is now using only 50Mb of memory compared to the 1.3Gb used by Jenkins this is a reasonable reduction. The web server is also serving requests in less than 120ms where Jenkins required 1.5s on the same hardware.

Code generation can speed up development and the UML to Ada generation was key to easily access and operate on the SQL database. The "Advanced Resource Embedder" tool was created for this build manager when the needs came to embed the SQL schema in the binary. Writing dedicated tools for code generation is also very interesting to automate some tasks and reduce costs at the end.

High level database access in Ada was key for the implementation of this build manager. It provides secure and safe access to the database and brings strong typing to database management. By using such representation in Ada, it becomes easier to implement complex algorithm easily on top of an SQL database.

The Porion build manager will evolve to introduce the notification of build results in order to inform users when a build failed. It will be improved in the area of build nodes to be able to start and stop virtual machines on demand. Last, the web server user interface must be completed to provide a full editable configuration of projects from a web browser while keeping the application safe.

References

[1] S. Carrez, "Porion a new build manager," FOSDEM CiCd devroom https://fosdem.org/2022/schedule/event/porion_a_new_build_manager/, 2022.

[2] S. Carrez, "Implementing a build manager in ada," FOSDEM Ada devroom https://fosdem.org/2022/schedule/event/ada_build_manager/, 2022.

[3] K. Kawaguchi, "Jenkins." https://github.com/jenkinsci/jenkins.

[4] S. Carrez, "Porion build manager." https://gitlab.com/stcarrez/porion.

[5] S. Carrez, "Ada utility library." https://github.com/stcarrez/ada-util.

[6] S. Carrez, "Ada web application." https://github.com/stcarrez/ada-awa.

[7] S. Carrez, "Secure web applications with awa," FOSDEM Ada devroom https://archive.fosdem.org/2019/schedule/event/ada_secureweb/, 2019.

[8] D. Anisimkov and P. Obry, "Ada web server." https://github.com/AdaCore/aws.

[9] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*. Upper Saddle River, NJ: Addison-Wesley, 2005.

[10] L. Tolke, "Argouml." https://github.com/argouml-tigris-org/argouml.

[11] S. Carrez, "Dynamo." https://github.com/stcarrez/dynamo.

[12] S. Carrez, "Ada database objects library." https://github.com/stcarrez/ada-ado.

[13] "Hibernate orm." https://hibernate.org/orm/.

[14] S. Carrez, "Advanced resource embedder." https://gitlab.com/stcarrez/resource-embedder.

Exporting Ada Software to Python and Julia

Jan Verschelde
University of Illinois at Chicago, Department of Mathematics, Statistics, and Computer Science, 851 S. Morgan St. (m/c 249), Chicago, IL 60607-7045; email: janv@uic.edu, http://www.math.uic.edu/∼jan

Abstract

The objective is to demonstrate the making of Ada software available to Python and Julia programmers using GPRbuild. GPRbuild is the project manager of the GNAT toolchain. With GPRbuild the making of shared object files is fully automated and the software can be readily used in Python and Julia. The application is the build process of PHCpack, a free and open source software package to solve polynomial systems by homotopy continuation methods, written mainly in Ada, with components in C++, available at github at `https://github.com/janverschelde/PHCpack`.

1 Language Agnostic Computing

This paper describes interface development from the perspective of an Ada programmer, aimed to export the functionality of a software package to Python [1] and Julia [2] computational environments, available through Jupyter notebooks [3]. The Jupyter notebook is the interface to SageMath [4], a free open source system for mathematical computing.

In order to export all functionality the interface passes through C, which may be regarded as a least common multiple of programming languages, as Ada, Python, and Julia share enough common ground to enable language agnostic computing, as Jupyter stands for Julia, Python, R, and many others.

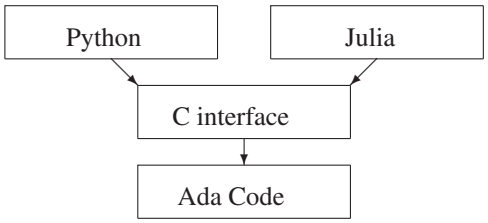


Figure 1: C as the least common multiple language.

The main point is to automate building with GPRbuild.

2 GPRbuild and Interface Development

The mixed language development is supported by GPRbuild, the project manager of the gnu-ada compiler GNAT. The build process, defined via library projects, results in shared object files (with the extension `.so` on Linux, `.dll` on Windows, and `.dylib` on Mac OS X). These shared object files can be called directly from a Python script or a Julia program.

In the C interface layer, the control is passed to a C program. The C program passes input data to some Ada procedure, calls an exported procedure, and extracts the output data via another call to an Ada procedure. The most basic and versatile manner to pass data is via a plain sequence of characters of 32-bit integers. As the *hello world* for this interface, consider the swapping of characters in a string.

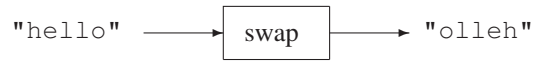


Figure 2: Swapping characters via an interface package.

The interface package as shown in Figure 3 exports a procedure to pass the input data, the `DoIt` procedure to compute the output data, and then a third function to return the output.

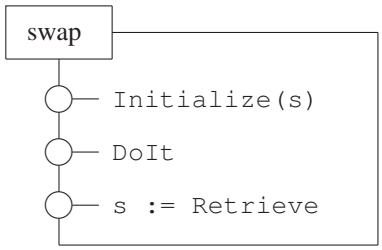


Figure 3: An interface package to swap characters in a string.

Then the C program calls the function `call_swap`, declared in Ada as below.

```
with C_Integer_Arrays;
 use C_Integer_Arrays;

function call_swap
  ( jobnbr : integer;
    sizedata : integer;
    swapdata : C_intarrs.Pointer;
    verbose : integer ) return integer;
```

where the package `C_Integer_Arrays` defines `C_Integer_Array` as an array of C integers, of type `Interfaces.C.int`. The package contains

```
  package C_intarrs is
    new Interfaces.C.Pointers
      (Interfaces.C.size_T,
       Interfaces.C.int,
       C_Integer_Array,0);
```

Observe that the `void` idiom of C is avoided. The details of this introductory project are posted at `github.com/janverschelde/ExportAdaGPRbuild`.

The C code to test takes a string `word`, converts the string into an array of 32-bit integers, and then calls the Ada code:

```
sizeword = strlen(word);

for(int idx = 0; idx < sizeword; idx++)
   dataword[idx] = (int) word[idx];

adainit();
fail = _ada_call_swap(0,sizeword,dataword,1);
fail = _ada_call_swap(1,sizeword,dataword,1);
fail = _ada_call_swap(2,sizeword,dataword,1);
adafinal();

for(int idx = 0; idx < sizeword; idx++)
   word[idx] = (char) dataword[idx];
```

The contents of the file `demo.gpr` defines the build of the C test program.

```
project Demo is

    for Languages use ("Ada", "C");

    for Source_Dirs use ("src");

    for Main use
    (
        "hello_world.adb",
        "main.adb",
        "test_call_swap.c"
    );

    for Object_Dir use "obj";

    for Exec_Dir use "bin";

end Demo;
```

To make a shared object file, a `library project` is defined. Below are the essentials of the instructions to make the `libdemo` as a shared object.

```
for Library_Dir use "lib";
for Library_Name use "demo";
for Library_Kind use "dynamic";
for Library_Auto_Init use "true";
for Library_Interface use
(
   "hello_world", "main", "swap",
   "call_swap", "c_integer_arrays"
);
for Library_Standalone use "encapsulated";

package Compiler is

  for Switches ("call_swap.adb") use ("-c");

end Compiler;

package Binder is

  -- use "-Lada" for adainit and adafinal
  for Default_Switches ("Ada")
     use ("-n", "-Lada");

end Binder;
```

Julia has the function `ccall()` to execute compiled C code. The Julia code below calls the `call_swap` procedure.

```
LIBRARY = "../Ada/lib/libdemo"

word = [Cint('h'), Cint('e'), Cint('l'),
        Cint('l'), Cint('o')]
println(word)
ptr2word = pointer(word, 1)
p = ccall((:_ada_call_swap, LIBRARY), Cint,
           (Cint, Cint, Ref{Cint}, Cint),
            0, 5, ptr2word, 1)
p = ccall((:_ada_call_swap, LIBRARY), Cint,
           (Cint, Cint, Ref{Cint}, Cint),
            1, 5, ptr2word, 1)
p = ccall((:_ada_call_swap, LIBRARY), Cint,
           (Cint, Cint, Ref{Cint}, Cint),
            2, 5, ptr2word, 1)
println(word)
```

The string `"hello"` is represented by `Int32[104, 101, 108, 108, 111]`. The last `println(word)` shows `Int32[111, 108, 108, 101, 104]`.

To extend Python code, an extension module must be defined in C or C++. The `setup.py` script has the list `extra_objects` to define the location of the compiled Ada code and the location of the Ada runtime libraries. The shared object made running `python setup.py build_ext` can then be directly imported in a Python session. The making of this extension can be done without makefiles.

3 Building PHCpack

As a demonstration to a large scale project, GPRbuild is applied to make share objects for PHCpack, a free and open source software package to solve polynomial systems with homotopy continuation. The python interface to PHCpack is phcpy [5]. Written mainly in Ada, PHCpack contains MixedVol [6] and DEMiCs [7] to count bounds on the number of isolated solutions fast. For MixedVol, a translation into Ada was made. The package DEMiCs is written in C++ and incorporated into PHCpack as such. As described in [8], the code for multiple double precision is provided by QDlib [9] and CAMPARY [10].

A Julia interface is under development. From the `Julia` folder of the PHCpack source distribution, running the Julia program `version.jl` at the command prompt:

```
$ julia version.jl
-> in use_c2phc4c.Handle_Jobs ...
PHCv2.4.85 released 2021-06-30
$
```

The `ccall()` uses the `libPHCpack` shared object, made with GPRbuild.

Acknowledgements

Supported by the National Science Foundation under grant DMS 1854513.

The author thanks Dirk Craeynest and Fernando Oleo Blanco for the organization of the Ada Devroom at FOSDEM 2022.

References

[1] F. Pérez, B. Granger, and J. Hunter, "Python: An ecosystem for scientific computing," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 12–21, 2011.

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[3] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. D. Team, "Jupyter Notebooks—a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents, and Agendas* (F. Loizides and B. Schmidt, eds.), pp. 87–90, IOS Press, 2016.

[4] W. Stein, "Sage: Creating a viable free open source alternative to Magma, Maple, Mathematica, and MATLAB," in *Foundations of Computational Mathematics, Budapest 2011* (F. Cucker, T. Krick, A. Pinkus, and A. Szanto, eds.), vol. 403 of *London Mathematical Society Lecture Note Series*, pp. 230–238, Cambridge University Press, 2012.

[5] J. Otto, A. Forbes, and J. Verschelde, "Solving polynomial systems with phcpy," in *Proceedings of the 18th Python in Science Conference*, pp. 563–582, 2019.

[6] T. Gao, T. Y. Li, and M. Wu, "Algorithm 846: MixedVol: a software package for mixed-volume computation," *ACM Trans. Math. Softw.*, vol. 31, no. 4, pp. 555–560, 2005.

[7] T. Mizutani and A. Takeda, "DEMiCs: A software package for computing the mixed volume via dynamic enumeration of all mixed cells," in *Software for Algebraic Geometry* (M. Stillman, N. Takayama, and J. Verschelde, eds.), vol. 148 of *The IMA Volumes in Mathematics and its Applications*, pp. 59–79, Springer-Verlag, 2008.

[8] J. Verschelde, "Parallel software to offset the cost of higher precision," *ACM SIGAda Ada Letters*, vol. 40, no. 2, pp. 59–64, 2020.

[9] Y. Hida, X. S. Li, and D. H. Bailey, "Algorithms for quad-double precision floating point arithmetic," in *15th IEEE Symposium on Computer Arithmetic (Arith-15 2001)*, pp. 155–162, IEEE Computer Society, 2001.

[10] M. Joldes, J.-M. Muller, V. Popescu, and T. W., "CAMPARY: Cuda Multiple precision arithmetic library and applications," in *Mathematical Software – ICMS 2016, the 5th International Conference on Mathematical Software*, pp. 232–240, Springer-Verlag, 2016.

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

Ada-Deutschland

Dr. Hubert B. Keller CEO
ci-tec GmbH
Beuthener Str. 16
76139 Karlsruhe
Germany
+491712075269
Email: h.keller@ci-tec.de
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

Ada-Europe Sponsors

27 Rue Rasson
B-1030 Brussels
Belgium
Contact:Ludovic Brenta
ludovic@ludovic-brenta.org

46 Rue d'Amsterdam
F-75009 Paris
France
Contact: Jamie Ayre
sales@adacore.com
www.adacore.com

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

22 St. Lawrence Street
Southgate, Bath BA1 1AN
United Kingdom
www.capgemini.com
.
.

Jacob Bontiusplaats 9
1018 LL Amsterdam
The Netherlands
Contact: Wido te Brake
wido.tebrake@deepbluecap.com
www.deepbluecap.com

24 Quai de la Douane
29200 Brest, Brittany
France
Contact: Pierre Dissaux
pierre.dissaux@ellidiss.com
www.ellidiss.com

In der Reiss 5
D-79232 March-Buchheim
Germany
Contact: Frank Piron
info@konad.de
www.konad.de

3271 Valley Centre Drive,Suite 300
San Diego, CA 92069
USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tools

Signal Business Centre
2 Innotec Drive, Bangor
North Down BT19 7PD
Northern Ireland, UK
enquiries@sysada.co.uk
www.sysada.co.uk

1090 Rue René Descartes
13100 Aix en Provence
France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/

Tiirasaarentie 32
FI 00200 Helsinki
Finland
Contact: Niklas Holsti
niklas.holsti@tidorum.fi
www.tidorum.fi

Corso Sempione 68
20154 Milano
Italy
Contact: Massimo Bombino
massimo.bombino@vector.com
www.vector.com

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch


Ada
europe