The journal for the international Ada community

# Ada User Journal

Volume 44
Number 2
June 2023

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 44
Number 2
June 2023

## Contents

# Editorial Policy for Ada User Journal

**Publication**

*Ada User Journal* — The Journal for the international Ada Community — is published by Ada-Europe. It appears four times a year, on the last days of March, June, September and December. Copy date is the last day of the month of publication.

**Aims**

*Ada User Journal* aims to inform readers of developments in the Ada programming language and its use, general Ada-related software engineering issues and Ada-related activities. The language of the journal is English.

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

The Journal publishes the following types of material:

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

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

**Original Papers**

Manuscripts should be submitted in accordance with the submission guidelines (below).

All original technical contributions are submitted to refereeing by at least two people. Names of referees will be kept confidential, but their comments will be relayed to the authors at the discretion of the Editor.

The first named author will receive a complimentary copy of the issue of the Journal in which their paper appears.

By submitting a manuscript, authors grant Ada-Europe an unlimited license to publish (and, if appropriate, republish) it, if and when the article is accepted for publication. We do not require that authors assign copyright to the Journal.

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

**Proceedings and Special Issues**

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

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

**News and Product Announcements**

Ada User Journal is one of the ways in which people find out what is going on in the Ada community. Our readers need not surf the web or news groups to find out what is going on in the Ada world and in the neighbouring and/or competing communities. We will reprint or report on items that may be of interest to them.

**Reprinted Articles**

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.

We have a reciprocal approach in granting permission for other publications to reprint papers originally published in *Ada User Journal.*

**Commentaries**

We publish commentaries on Ada and software engineering topics. These may represent the views either of individuals or of organisations. Such articles can be of any length – inclusion is at the discretion of the Editor.

Opinions expressed within the *Ada User Journal* do not necessarily represent the views of the Editor, Ada-Europe or its directors.

**Announcements and Reports**

We are happy to publicise and report on events that may be of interest to our readers.

**Reviews**

Inclusion of any review in the Journal is at the discretion of the Editor. A reviewer will be selected by the Editor to review any book or other publication sent to us. We are also prepared to print reviews submitted from elsewhere at the discretion of the Editor.

**Submission Guidelines**

All material for publication should be sent electronically. Authors are invited to contact the Editor-in-Chief by electronic mail to determine the best format for submission. The language of the journal is English.

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

# Editorial

I would like to start this editorial by highlighting the new AUJ distribution procedure. The AUJ is released in both printed and digital formats, and now all Ada-Europe members and AUJ subscribers can opt to receive only the digital version, instead of both. With many members expressing interest in receiving only the digital version, the number of printed and mailed copies of the journal was already reduced with the previous March 2023 issue. This represents an important cost saving for Ada-Europe, hence a step forward towards the sustainability of the publication (whatever happens in 2024 concerning the planned merge of the AUJ and the Ada Letters). Members wishing to receive only the digital version can let us know anytime!

Concerning the technical contents of this issue, we start with a special contribution by Frank Singhoff from the University of Brest, providing a summary of the 2022 ADEPT Workshop (whose proceedings were published in the previous issue) and analysing the strengths and weaknesses of the AADL ecosystem.

The rest of the issue is dedicated to publishing the first set of articles from the AEiC 2023 Work-in-Progress session. The session included fifteen articles, eight of which are herein included. The first two articles are authored by researchers from the Faculty of Sciences of the University of Lisbon and describe work on embedded systems security and on a simulation-based platform to study protocols for cooperative autonomous driving systems. The third article, authored by F. Lucchetti and M. Voelp from the University of Luxembourg, is also related to autonomous vehicles, in this case proposing a solution that exploits the rejuvenation of some components to improve fault tolerance. Then, the reader will find two papers on the application of formal methods for the early verification of cyber-physical systems (CPS) designs. The first, written by D. de Niz and L. Wrage, from Carnegie Mellon University in the USA, describes an approach named Symbolic Assurance Refinement for the verification of architectural models. The second, by F. Malaquias, G. Giantamidis, S. Basagiannis, S. F. Rollini and I. Amundson, all from Collins Aerospace, proposes a model-based engineering approach for the design of CPS and addresses the verification of security requirements. Two papers related to the Ada language follow, one by L. Creuse, M. Eyraud and V. Garèse from AdaCore, on tools specifically suited for testing Ada programs, and another by J. Hughes, from Carnegie Mellon University, reporting on an ongoing effort to mechanize the Ada Ravenscar profile using the Coq theorem prover. The last article included in this issue is authored by researchers from the Barcelona Supercomputing Center and the Polytechnic University of Catalonia, which addresses the problem of WCET estimation in multicore and GPU platforms using a measurement-based probabilistic timing analysis method.

As usual, the issue includes the News Digest section prepared by its editor Alejandro R. Mosteo, and the Calendar and Events section, prepared by Dirk Craeynest. Last but not least, we would like to call the attention of the reader to the 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024), which next year will take place in Barcelona, Spain, and whose preliminary announcement can be found on page 123.

*Antonio Casimiro*
*Lisboa*
*June 2023*
*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,

I place the single focus of this issue on the momentous event all of us Ada enthusiasts have long been anticipating: Ada 2022 is officially here [1][2].

Sincerely,

Alejandro R. Mosteo.

[1] "Ada 2022 LRM by Springer", in References to Publications.

[2] "Ada 2022 at Last!", in Ada Practice.

## Ada-related Events

### Ada-Belgium Spring 2023 Event

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Subject: Ada-Belgium Spring 2023 Event, Sun 28 May 2023*
*Date: Thu, 11 May 2023 14:31:54 -0000*
*Newsgroups: comp.lang.ada, fr.comp.lang.ada,be.comp.programming*

--------------------------------------------------------

Ada-Belgium Spring 2023 Event

Sunday, May 28, 2023, 12:00-19:00

Leuven, Belgium

including at 15:00

2023 Ada-Belgium General Assembly

and at 16:00

Ada Round-Table Discussion

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/23/230528-abga.html

--------------------------------------------------------

*** Announcement

The next Ada-Belgium event will take place on Sunday, May 28, 2023 in Leuven.

After an interruption of 3 years due to the COVID-19 pandemic, and for the 13th time, Ada-Belgium organizes their "Spring Event", which starts at noon, runs until 7pm, and includes an informal lunch, the 30th General Assembly of the organization, and a round-table discussion on Ada-related topics the participants would like to bring up.

*** Schedule

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

*** Participation

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

For practical reasons registration is required. If you would like to attend, please send an email before Thursday, May 25, 18:00, to Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> with the subject "Ada-Belgium Spring 2023 Event", so you can get precise directions to the place of the meeting. Even if you already responded to the preliminary announcement, please reconfirm your participation ASAP.

If you are interested to join Ada-Belgium, please register by filling out the 2023 membership application form[1] and by paying the appropriate fee before the General Assembly. After payment you will receive a receipt from our treasurer and you are considered a member of the organization for the year 2023 with all member benefits[2]. Early enrollment ensures you receive the full Ada-Belgium membership benefits (including the Ada-Europe indirect membership benefits package).

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

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium/forms/member-form23.html

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

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

*** Informal lunch

The organization will provide food and beverage to all Ada-Belgium members. Non-members who want to participate at the lunch are also welcome: they can choose to join the organization or pay the sum of 20 Euros per person to the Treasurer of the organization.

*** General Assembly

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

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

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/23/230528-abga-conv.html

*** Ada Round-Table Discussion

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

*** Directions

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

Looking forward to meet many of you!

Dirk Craeynest

President Ada-Belgium

Dirk.Craeynest@cs.kuleuven.be

--------------------------------------------------

Acknowledgements

We would like to thank our sponsors for their continued support of our activities: AdaCore, and KU Leuven (University of Leuven).

If you would also like to support Ada-Belgium, find out about the extra Ada-Belgium sponsorship benefits:

http://www.cs.kuleuven.be/~dirk/ada-belgium/member-benefit.html#sponsor

--------------------------------------------------

(V20230511.1)

## AEiC 2023 Final Call

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Subject: Press Release - AEiC 2023, Ada-Europe Reliable Softw. Technol.*
*Date: Fri, 9 Jun 2023 10:56:57 -0000*
*Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc*

--------------------------------------------------

FINAL Call for Participation

*** UPDATED Program Summary ***

27th Ada-Europe International Conference on

Reliable Software Technologies (AEiC 2023)

13-16 June 2023, Lisbon, Portugal

www.ada-europe.org/conference2023

Organized by Ada-Europe

in cooperation with ACM SIGAda, SIGBED, SIGPLAN,

the Ada Resource Association (ARA), and the University of Lisbon

#AEiC2023 #AdaEurope #AdaProgramming

*** Final Program available on the conference web site ***

*** Add tutorials and/or a workshop to your conference registration ***

www.ada-europe.org/conference2023/tutorials.html

*** Welcome Event on Tuesday evening ***

--------------------------------------------------

Press release:

27th Ada-Europe Int'l Conference on Reliable Software Technologies

International experts meet in Lisbon

Lisbon, Portugal (9 June 2023) - Ada-Europe together with the University of Lisbon organizes from 13 to 16 June 2023 the 27th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2023), in cooperation with the Ada Resource Association (ARA), and with ACM's Special Interest Groups on Ada (SIGAda), on Embedded Systems (SIGBED) and on Programming Languages (SIGPLAN).

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

This year's conference offers 4 tutorials, a keynote and a panel discussion, a technical program of 6 sessions with peer-reviewed papers, industrial and work-in-progress presentations, posters, social events, and 2 workshops. Presentations are given by authors from 15 countries.

Six tutorials are scheduled on Tuesday, targeting different audiences:

- "The HAC Ada Compiler",
- "Controlling I/O Devices with Ada and the Linux Simple I/O Library",
- "Everything you Always Wanted to Know about Characters and Strings",
- "Introduction to the Development of Safety Critical Software",
- "Rust Fundamentals",
- "Concurrency and Parallelism in Rust".

On Wednesday and Thursday, the networking area features WiP posters, as well as an Ada-Europe booth.

Eminent speakers have been invited on each of the core conference days:

- on Wednesday, a keynote talk by Alcides Fonseca, from LASIGE, University of Lisbon Faculty of Sciences, who will talk about "Applications of liquid types for more reliable software";
- on Thursday, a panel on "Promises and Challenges of AI-enabled Software Development Tools for Safety-Critical Applications" with Douglas Schmidt (Vanderbilt University, USA), Jochen Quante (Robert Bosch GmbH, Germany), and Jon Pérez Cerrolaza (IKERLAN, Spain).

The technical program on Wednesday and Thursday includes 6 journal-track refereed technical papers, 7 industrial, and 15 work-in-progress presentations, in sessions on: Verification and Validation 1, Advanced Systems, Reliability and Performance, Verification and Validation 2, Reliable Programming, Real-Time Systems.

On Friday the conference hosts for the 8th year the workshop on "Challenges and New Approaches for Dependable and Cyber-Physical Systems Engineering" (DeCPS 2023), as well as the workshop "AADL by its Practitioners (ADEPT)".

Peer-reviewed papers have been submitted to a special issue of the Journal of Systems Architecture and are heading towards final acceptance as open-access publications. Industrial and work-in-progress presentations, together with tutorial abstracts, and workshop papers, will appear in issues of the Ada User Journal, the quarterly magazine of Ada-Europe.

The social program includes on Tuesday evening a Welcome Reception in the gardens of the National Museum of Science & Natural History, and on Wednesday evening the Conference Banquet in the "Casa do Alentejo" restaurant, an old palace in downtown Lisbon with several exquisite rooms, that served as a casino in the 20th century.

The Best Presentation Award will be offered during the Closing session.

The full program is available on the conference web site.

Online registration is still possible.

-------

Latest updates:

The 16-page "Final Program" is available at www.ada-europe.org/conference2023/media/AEiC_2023_Final_Program.pdf

Check out the tutorials in the PDF program, or in the schedule at

www.ada-europe.org/conference2023/tutorials.html.

Registration is done on-line. For all details, go to

www.ada-europe.org/conference2023/registration.html.

A printed Conference Booklet with abstracts of all technical papers and industrial presentations will be included in every conference handout, and will be available on the conference web site.

AEiC 2023 is sponsored by Ada-Europe (www.ada-europe.org), AdaCore (www.adacore.com), and GMV (www.gmv.com).

Help promote the conference by advertising it.

Recommended Twitter hashtags: #AEiC2023 #AdaEurope #AdaProgramming.

--------------------------------------------------

Our apologies if you receive multiple copies of this announcement.

Please circulate widely.

Dirk Craeynest, AEiC 2023 Publicity Chair

Dirk.Craeynest@cs.kuleuven.be

* 27th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2023)

* June 13-16, 2023, Lisbon, Portugal, www.ada-europe.org/conference2023

(V7.1)

## Ada Monthly Meetup 2023

[see also “Ada Monthly Meeting Proposal” in this AUJ issue, pp.114-115 —arm]

*From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>*
*Subject: Ada Monthly Meetup 2023*
*Date: Wed, 31 May 2023 14:27:07 +0200*
*Newsgroups: comp.lang.ada*

Hi all,

This message contains the final time of the meeting, connection details and other info.

The (first!) Ada Monthly Meetup will take place this Saturday 3rd of June at 13:00 UTC Time. That corresponds to 15:00 CET (Central European Time: Madrid, Paris, Berlin, Rome...).

The meetup will take place over at Jitsi, a conferencing software that runs on any modern browser. The link is https://meet.jit.si/2023AdaMonthlyMeetupJune The room name is "2023AdaMonthlyMeetupJune" and in case it asks for a password, it will be set to "first". I do not want to set up a password, but in case it is needed, it will be the one above without the quotes. The room name is generally not needed as the link should take you directly there, but I want to write it down just in case someone needs it.

Talks:

No one proposed any topics, but that is fine as **this first meeting will not be recorded.** I will record it for internal testing and to see how it works, but it will not be published.

Having no talks will allow us, the community, to discuss any technical issues and comments that may help improve the experience of the monthly meetup. However, I will give a short introduction and share my ideas at the beginning :) Someone could also propose a topic for the next meetup too.

If I forgot something, please, point it out so that any issues can get patched out.

Best regards,

Fer

P.S: I, Fer, will post this over at the C.L.A and Ada-Lang.io. Feel free to repost this to Reddit, Gitter/Matrix, Telegram or any other channels! The more people know about this, the better (I hope).

P.P.S: this if for C.L.A only. The main thread was named "Ada Monthly Meetup Proposal". However, as this is no longer a proposal, but the actual thing, I am creating a new thread. For more information, please, refer to the aforementioned thread!

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Date: Thu, 1 Jun 2023 09:58:34 -0000*

Fernando Oleo Blanco <irvise_ml@irvise.xyz> wrote:

>The more people know about this, the better (I hope).

Reposted to all Ada-Belgium members. HTH

# Ada and Education

## New Project: Alice

*From: frances...@gmail.com <francesc.rocher@gmail.com>*
*Subject: New project: Alice*
*Date: Tue, 16 May 2023 11:22:12 -0700*
*Newsgroups: comp.lang.ada*

After months of dedicated work, I'm thrilled to introduce my project: Alice!

Alice, "Adventures for Learning and Inspiring Coding Excellence", is a collaborative Ada framework that allows programmers to enhance and share their solutions to various problem sources (e.g. Project Euler, CodinGame and Advent of Code), fostering collaboration, learning and creativity.

While it's currently in the proof of concept stage, and only Project Euler is supported, I believe it holds immense potential.

The wiki pages, https://github.com/alice-adventures/Alice/wiki, offer a glimpse into Alice's concept, participation opportunities, and development ideas.

I warmly invite all members of the Ada community, as well as beginners and students exploring Ada, to read across the wiki pages and share your valuable feedback. Your insights and input will be instrumental in shaping Alice's future. Together, let's unlock the possibilities and make a significant impact.

Stay tuned for the upcoming public release, as we embark on this exciting journey together!

## "Ada Computer Science" at Raspberrypi.org

*From: Ingo M. <it.marks.info@gmail.com>*
*Subject: "Ada Computer Science" at raspberrypi.org*
*Date: Sun, 28 May 2023 07:06:51 -0700*
*Newsgroups: comp.lang.ada*

The Raspberry Pi Foundation announces an "Ada Computer Science" project which has nothing to do with the Ada programming language.

https://www.raspberrypi.org/blog/ada-computer-science/

"We are excited to launch Ada Computer Science, the new online learning platform for teachers, students, and anyone interested in learning about computer science."

So far the focus is set on the current ChatGPT hype, and code examples in Python, Java, VB, and C#. It could be a good opportunity to promote the Ada language by providing similar courses. Otherwise there could be a risk that newcomers associate Ada with this project rather than the language.

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Date: Mon, 29 May 2023 07:36:04 -0000*

FWIW, I just posted the following comment on that page:

"Will you also be using the Ada programming language, a modern language with a long track record of successful projects and ideally suited to develop reliable and trustworthy software?"

It is currently marked as "This comment is awaiting moderation."

# Ada-related Resources

[Delta counts are from February 12th to July 28th. —arm]

## Ada on Social Media

*From: Alejandro R. Mosteo <amosteo@unizar.es>*
*Subject: Ada on Social Media*
*Date: 28 Jul 2023 14:35 CET*
*To: Ada User Journal readership*

Ada groups on various social media:

- Reddit: 8_371 (+22) members [1]

- LinkedIn: 3_448 (+12) members [2]
- Stack Overflow: 2_345 (+22) questions [3]
- Gitter: 230 (+11) people [4]
- Telegram: 159 (-1) users [5]
- Ada-lang.io: 133 (+26) users [6]
- Libera.Chat: 73 (-1) concurrent users [7]
- Twitter: Discontinued due to free API being removed.

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

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

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

[4] https://app.gitter.im/#/room/#ada-lang_Lobby:gitter.im

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

[6] https://forum.ada-lang.io/u

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

## Repositories of Open Source Software

*From: Alejandro R. Mosteo <amosteo@unizar.es>*
*Subject: Repositories of Open Source software*
*Date: 28 Jul 2023 14:41 CET*
*To: Ada User Journal readership*

| | |
|---|---|
| Rosetta Code: | 941 (+15) examples [1] |
| | 41 (+1) developers [2] |
| GitHub: | 987* (+224) developers [3] |
| Alire: | 363 (+26) crates [4] |
| Sourceforge: | 243 (+3) projects [5] |
| Open Hub: | 214 (=) projects [6] |
| Codelabs: | 57 (+3) repositories [7] |
| Bitbucket: | 31 (=) repositories [8] |

*This number is an unreliable lower bound 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://alire.ada.dev/crates.html

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

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

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

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

## Language Popularity Rankings

*From: Alejandro R. Mosteo <amosteo@unizar.es>*
*Subject: Ada in language popularity rankings*
*Date: 28 Jul 2023 14:53 CET*
*To: Ada User Journal readership*

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

- TIOBE Index: 23 (+5) 0.77% (+0.35%) [1]

- PYPL Index: 16 (+3) 1.06% (+0.23%) [2]

- Stack Overflow Survey 42 (new) 0.77% (new) [3]

- IEEE Spectrum (general): 35 (=) Score: 1.16 [4]

- IEEE Spectrum (jobs): 33 (=) Score: 0.79 [4]

- IEEE Spectrum (trending): 32 (=) Score: 3.95 [4]

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

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

[3] https://survey.stackoverflow.co/2023/

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

## GnatStudio Cookbook

*From: Rod Kay <rodakay5@gmail.com>*
*Subject: [Ann] GnatStudio Cookbook*
*Date: Wed, 28 Jun 2023 06:53:49 +1000*
*Newsgroups: comp.lang.ada*

Hello again all,

In the hope it might help other people building or OS packaging GnatStudio, I've prepared a 'cookbook' of sorts.

It provides build instructions for the entire GnatStudio project stack, beginning with gprbuild-bootstrap and culminating in the build of gnatstudio. The individual 'recipes' take the form of pacman PKGBUILDs with tarballs and patches.

Here is the link ...

https://github.com/charlie5/archlinux-gnatstudio-support/tree/main/gnatstudio-cookbook

# Ada-related Tools

## AdaStudio 2023 Release 03/04/2023 Free Edition

*From: Leonid Dulman <leonid.dulman@gmail.com>*
*Subject: Announce: AdaStudio-2023 release 03/04/2023 free edition*
*Date: Sun, 2 Apr 2023 21:12:06 -0700*
*Newsgroups: comp.lang.ada*

It based on Qt-6.5.0-everywhere opensource (expanded with modules from Qt-5.15: qtgamepad, qtx11extras,qtwinextras),VTK-9.2.0,FFMPEG-5.2.1,OpenCV-4.7.0,SDL2-2.24.0,QtAV-1.13 MDK-SDK(wang-bin)

Qt6ada version 6.5.0 open source and qt6base.dll ,qt6ext.dll (win64),libqt6base.so,libqt6txt.so(x86-64) built with Microsoft Visual Studio 2023 x64 Windows, GCC amd64 in Linux.

Package tested with GNAT gpl 2020 Ada compiler in Windows 64bit , Linux amd64 Debian 11.2

AdaStudio-2023 includes next modules : qt6ada,vtkada,qt6mdkada,qt6cvada(face recognition, face detection,face identification,objects detectection,QRcode detector,BARcode detection and others ) and voice recognizer.

Qt6Ada is built under GNU LGPLv3 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/drive/folders/0B2QuZLoe-yiPbmNQRl83M1dTRVE?resourcekey=0-b-M35gZhynB6-LOQww33Tg&usp=sharing

WebPage is https://r3fowwcolhrzycn2yzlzzw-on.drv.tw/AdaStudio/index.html

Directories tree is

[…]

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

The simple manual how to build Qt6Ada application can be read in "How to use Qt6ada.pdf"

If you have any problems or questions, tell me know.

Leonid(leonid.dulman@gmail.com)

## VisualAda 1.0.0.12

*From: Alex Gamper <alby.gamper@gmail.com>*
*Subject: ANN: VisualAda (Ada Integration for Visual Studio 2022) release 1.0.0.12*
*Date: Mon, 10 Apr 2023 18:43:56 -0700*
*Newsgroups: comp.lang.ada*

Dear Ada Community

VisualAda version 1.0.0.12 for Visual Studio 2022 has been released

Enhancements include the following

- Bug fixes in Intellisense (Statement completion)

Please feel free to download the free plugin from the following URL

https://marketplace.visualstudio.com/items?itemName=AlexGamper.VisualAda-2022

## Currency Library for Ada?

*From: A.J. <ianozia@gmail.com>*
*Subject: Currency Library for Ada?*
*Date: Thu, 13 Apr 2023 07:17:27 -0700*
*Newsgroups: comp.lang.ada*

Does anyone know if Ada has a currency library? Ideally one that includes the ISO 4217 currency standard?

I've seen currency referenced as examples in the Style Guide[1] and other documentation[2] but I'm having trouble searching for anything concrete (and googling "ada" and "currency" has not helped due to some unfortunately named crypto stuff).

I'm also interested in if there's any Ada libraries for iso 3166 (country codes).

If none of this exists, that's fine, it just gives me a reason to build it out myself, but I don't want duplication of effort :)

[1] https://ada-lang.io/docs/style-guide/Reusability/#guideline-16

[2] https://docs.adacore.com/live/wave/aunit/html/aunit_cb/aunit_cb/fixture.html

*From: J-P. Rosen <rosen@adalog.fr>*
*Date: Thu, 13 Apr 2023 19:37:19 +0200*

Le 13/04/2023 à 16:17, A.J. a écrit :

> I'm also interested in if there's any Ada libraries for iso 3166 (country codes).

It's a standard package, Ada.Locales

*From: A.J. <ianozia@gmail.com>*
*Date: Thu, 13 Apr 2023 11:12:19 -0700*

On Thursday, April 13, 2023 at 1:37:22 PM UTC-4, J-P. Rosen wrote:

> It's a standard package, Ada.Locales

Thanks for finding that! This looks like a good foundation for validating county codes, though it doesn't appear to contain an index of them, or expand into the 3-letter codes (e.g. USA vs US). I was looking into the implementation, and the GNAT[1] runtime seems to be true to spec, while the Drake runtime[2] looks like it's expanding into closer to what I'm looking for with its iso639 tables [3]. I may be able to build on this set, though and use the existing structures.

[1] https://github.com/gcc-mirror/gcc/blob/master/gcc/ada/libgnat/a-locale.ads & https://github.com/gcc-mirror/gcc/blob/master/gcc/ada/libgnat/a-locale.adb

[2] https://github.com/ytomino/drake/blob/ master/source/environment/a-locale.ads

[3] https://github.com/ytomino/drake/blob/master/source/environment/a-locale.adb#L60

*From: Devin Rozsas <devinrozsas@gmail.com>*
*Date: Thu, 4 May 2023 10:59:11 -0700*

Em quinta-feira, 13 de abril de 2023 às 11:17:28 UTC-3, A.J. escreveu:

> Does anyone know if Ada has a currency library? Ideally one that includes the ISO 4217 currency standard?

I'm actually making something like this, but it isn't complete, and uses Lua scripts to handle different currencies (and formatting). It also has location support (country, state, city) and language support, including message translation (so the program can output stuff in the user's language). For this, TOML is used.

It uses Glottolog codes to identify languages, and FIFA codes for the countries.

I paused the development because I'm focusing on another project that has been causing me some headaches lately. It's broken and cannot deliver what you want - as of now.

*From: Devin Rozsas <devinrozsas@gmail.com>*
*Date: Thu, 4 May 2023 11:27:47 -0700*

By the way, the code is here: https://sr.ht/~devin/Azurite-Ada/

Again, it's incomplete, and probably isn't exactly what you're looking for.

## Ada Interface to Excel File

*From: Adamagica <christ-usch.grein@t-online.de>*
*Subject: Ada interface to Excel file*
*Date: Wed, 19 Apr 2023 10:36:10 -0700*
*Newsgroups: comp.lang.ada*

I create Ada code from an Excel file. For this, I first manually export the file to csv format. The code generator works on the csv file. I'd like to automate this first step by including the export into the code generator.

I guess there is a C interface for Excel. I only just need the export functionality, not a full interface. However, being illiterate in C, I'd further welcome help on the way to define an Ada interface to this C code.

Can anyone help, please? Thanx a lot.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Wed, 19 Apr 2023 20:22:34 +0200*

G. de Montmollin has an Ada Excel writer, an Ada pkg for writing Excel files (https://sourceforge.net/projects/excel-writer/). Presumably it could be modified to read them.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Thu, 20 Apr 2023 11:18:53 +0200*

AFAIK, Excel has an ODBC driver. So you can simply read/write an Excel table directly from Ada using ODBC SQL statements.

## Units of Measurement for Ada 3.13

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Subject: ANN: Units of measurement for Ada v3.13*
*Date: Sun, 23 Apr 2023 09:20:44 +0200*
*Newsgroups: comp.lang.ada*

The library provides means for handling measurement units in Ada.

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

Changes to the previous version:

- The package Generic_Complex_Measures was added to provide dimensioned complex values;
- The package Complex_Measures added as an instance of Generic_Complex_Measures with the type Float.

*From: Simon Wright <simon@pushface.org>*
*Date: Sun, 23 Apr 2023 11:14:35 +0100*

Thanks for this.

The link in "You also may wish to visit this site devoted to the problem of dimensioned values in Ada."

(http://www.christ-usch-grein.homepage.t-online.de/Ada/Dimension/SI.html)

results in "Host not found".

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Sun, 23 Apr 2023 03:48:50 -0700*

> The link [...] results in "Host not found".

This can be found there:

https://www.adaic.org/ada-resources/tools-libraries/ see "Christoph Grein's Essentials"

or more directly:

http://archive.adaic.com/tools/CKWG/Dimension/Dimension.html

## Simple Components 4.66

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Subject: ANN: Simple components for Ada v 4.66*
*Date: Sun, 23 Apr 2023 09:25:10 +0200*
*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, persistent storage, multiple connections server/client designing tools and protocols implementations.

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

Changes to the previous version:

- The ambiguities in the ODBC.API package implementation are fixed.

## GCC 13.1.0 for MacOS Monterey++

*From: Simon Wright <simon@pushface.org>*
*Subject: ANN: GCC 13.1.0 for macOS Monterey++*
*Date: Thu, 27 Apr 2023 16:22:37 +0100*
*Newsgroups: comp.lang.ada*

Find this release, built on Intel but runs on Apple silicon under Rosetta, at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-13.1.0-x86_64

NB, previous builds were for macOS El Capitan or later, but that machine was elderly.

## SparForte 2.6

*From: Ken Burtch <koburtch@gmail.com>*
*Subject: ANN: SparForte 2.6*
*Date: Tue, 2 May 2023 05:07:35 -0700*
*Newsgroups: comp.lang.ada*

SparForte is a scripting language, template language and shell based on Ada and Bourne shell. It has been in development for 22 years and has about 129,000 lines of code.

This release includes

New features: 11

Changes: 7

Fixes: 12

New features include case procedures, named shell sessions, and Alire support (experimental).

The details of the release are at

https://sparforte.com/news/2023/news_may_2023.html

A summary of the new features is at

https://www.pegasoft.ca/coder/coder_march_2023.html

SparForte can be downloaded from the home page at

https://sparforte.com

*From: 196...@googlemail.com <1963bib@googlemail.com>*
*Date: Sat, 6 May 2023 06:08:45 -0700*

Can't be built without sound...

```
gcc -c -O1 -march=athlon64 -gnat12 -gnatfaoN -gnatVaep -gnateEeEeF -fstack-protector -I./adacgi-1.6/ -I./apq-2.1/ -I./pegasock/ -I./areadline/ -lpcre parser_sound.adb
```

```
parser_sound.adb:93:06: statement expected
```

```
gnatmake: "parser_sound.adb" compilation error
```

If you try and build it without readline...

```
gcc -c -O1 -march=athlon64 -gnat12 -gnatfaoN -gnatVaep -gnateEeEeF -fstack-protector -I./adacgi-1.6/ -I./pegasock/ pegasoft-user_io-getline.adb
```

```
pegasoft-user_io-getline.adb:286:26: "optional_bold" is undefined
```

```
gnatmake: "pegasoft-user_io-getline.adb" compilation error
```

*From: Ken Burtch <koburtch@gmail.com>*
*Date: Sat, 6 May 2023 06:19:09 -0700*

Thank you for sharing this issue.

My automated release tests build SparForte without sound and without readline each night and they have been successful.

I will attempt to diagnose what the problem is. Do you have any further information about your build environment that might be related to this issue?

*From: Ken Burtch <koburtch@gmail.com>*
*Date: Sat, 6 May 2023 06:34:50 -0700*

I have pushed fixes for these two issues to the GitHub master branch.

I will do a full set of build tests manually to ensure nothing else is missing.

I will also investigate what the errors were not caught by the automated tests.

Thank you for reporting this issue.

## UXStrings 0.5.0

*From: Blady <p.p11@orange.fr>*
*Subject: [ANN] Release of UXStrings 0.5.0*
*Date: Fri, 5 May 2023 05:36:42 +0200*
*Newsgroups: comp.lang.ada*

This Ada library, providing Unicode character strings of dynamic length, is enriched by a third implementation: UXStrings3 [1] also available on Alire [2]. With this latter implementation, the characters are stored in Unicode form and the management of dynamic size uses the standard Wide_Wide_Unbounded strings library.

Performance with Gnoga [3] is better. UXStrings2 already brought better performance in the case of strings only made up of ASCII characters (improvement by a factor 2 to 3 compared to UXStrings1). With UXStrings3 performance in the latter case is still improved (factor 6 to 7 compared to UXStrings1) moreover in the case of strings accentuated in French and strings containing emojis the process times are also improved (factor 7 to 8 by compared to UXStrings1 or even more in the case of emojis).

For all cases, the global memory occupation of the Gnoga application is generally similar (9 to 10 Mb). The memory occupation due to UXStrings3 is negligible compared to the memory occupation of the server engine implemented in Gnoga.

Study case: AdaEdit application using the Gnoga graphics library with

UTF-8 files:

English 315 kb

French: 447 kb

Emojis: 439 kb

Process: read all lines of the given file and display the full text

Regardless of the implementation chosen, the appealing of a library is mainly based on the capabilities it offers (API). So far in UXStrings, these are similar to those of the strings Ada standard libraries. If you find some missing, make your proposals on Github [4].

Pascal.

[1] https://github.com/Blady-Com/UXStrings/blob/master/src/uxstrings3.ads

[2] https://alire.ada.dev/crates/uxstrings.html

[3] https://sourceforge.net/projects/gnoga

[4] https://github.com/Blady-Com/UXStrings/issues

*From: Vincent D. <vincent.diemunsch@gmail.com>*
*Date: Thu, 29 Jun 2023 01:49:12 -0700*

Hello Pascal,

Thank you for this contribution. Here are some comments:

- since UTFString is a class ("a tagged record type"), why don't you create an abstract root "UXString" and then derive specialized object types ? Like UTF_8_XString, UTF_16_XString, ASCII_XString, Win_1252_XString, Latin_XString, etc.

- The default format to convert between different encodings should be UTF-8 as it is now ubiquitous.

> [...] moreover in the case of strings accentuated in French and strings containing emojis the process times are also improved (factor 7 to 8 by compared to UXStrings1

- I find quite astonishing to have a factor 8 compared to UTF-8 encoding. Do you have an explanation ? This looks like a poor implementation because UTF-8 encoding is fast and allows direct manipulation in most cases. Maybe because random access is treated as sequential access for UTF-8 encoded strings but this again is poor implementation.

## GnatStudio 20230501

*From: 196...@googlemail.com <1963bib@googlemail.com>*
*Subject: GnatStudio 20230501 released*
*Date: Sun, 14 May 2023 14:31:05 -0700*
*Newsgroups: comp.lang.ada*

And for Linux it's an appimage. Why? I mean? Its...?

I just wish they could get it into shape where the build was doable without so much hassle - I've never been able to manage it.

*From: Rod Kay <rodakay5@gmail.com>*
*Date: Mon, 15 May 2023 20:41:14 +1000*

On 15/5/23 07:31, 196...@googlemail.com wrote:

> I just wish [...] the build was doable without so much hassle

The build *has* been getting easier. I maintain the Archlinux gnatStudio package and have nearly got it to build. Currently, I'm waiting on a new/matching release of the AdaCore spawn project. I could, I suppose, use the latest commit version but would prefer to use a formal release.

Also, in the new binary, 'Find all references' appears to be broken (it finds no references). I guess, the same would apply for the refactoring tool. I've been advised to report the issue and will do so tomorrow. It might help to know if other people also experience the same problem(s), before reporting ?

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Mon, 15 May 2023 15:58:43 +0200*

On 2023-05-15 12:41, Rod Kay wrote:

> It might help to know if other people also experience the same problem(s)

After typing "with Ada.Strings." I received a use clause as a suggested completion. After completing the subprogram name in a subprogram call, I was shown something other than the subprogram specification. After typing the '(' for the parameter list, I was shown something other than the formal parameters.

*From: Maxim Reznik <reznikmm@gmail.com>*
*Date: Thu, 1 Jun 2023 02:21:55 -0700*

понедельник, 15 мая 2023 г. в 13:43:31 UTC+3, Rod Kay:

> On 15/5/23 07:31, 196...@ wrote:

> > And for Linux it's an appimage. Why?

Why not? It's compact. It doesn't require any installation, so it's handy. You can extract content with --appimage-extract and install GS with ./squashfs-root/usr/doinstall as before.

> Currently, I'm waiting on a new/matching release of the AdaCore spawn project.

All sources are in release assets, like gnatstudio-sources-x86_64-linux.tar.gz. It has spawn-24.0w-20230428-162D4-src.tar.gz for example.

> Also, in the new binary, 'Find all references' appears to be broken

It looks like your ada_language_server doesn't work. Take a look in GS log files (in ~/.gnatstudio/ folder).

*From: Rod Kay <rodakay5@gmail.com>*
*Date: Fri, 2 Jun 2023 04:27:02 +1000*

On 1/6/23 19:21, Maxim Reznik wrote:

> All sources are in release assets

Ah, great. I will try to rebuild with this.

> It looks like your ada_language_server doesn't work.

I've just re-tested and 'Find all references' works perfectly. How embarrassing!

All i can think of is that I may have had an old gnatstudio version running when I did the GS update and so was still using the old version when I initially tested.

Thanks very much Reznik, very helpful.

*From: Maxim Reznik <reznikmm@gmail.com>*
*Date: Mon, 5 Jun 2023 03:15:13 -0700*

Great! Waiting for GNAT Studio in Arch Linux :)

Speaking about AppImage. If you want installed version of GNAT Studio (for instance to have an access to the gnatdoc/gnatdoc4), then you can extract AppImage as an old .tag.gz archive and run doinstall:

```
chmod +x ./GNAT_Studio-x86_64.AppImage

./GNAT_Studio-x86_64.AppImage --appimage-extract

./squashfs-root/usr/doinstall
```

*From: Rod Kay <rodakay5@gmail.com>*
*Date: Mon, 5 Jun 2023 22:55:51 +1000*

On 5/6/23 20:15, Maxim Reznik wrote:

> Great! Waiting for GNAT Studio in Arch Linux :)

Heh, I've just this minute finished the build/install of GNAT Studio for Arch Linux. The build of GS (and all of it's dependencies) went very well, largely due to using all of the sources provided in the recent GS sources tarball release. So thank you again for suggesting that.

I still have one problem to solve. When I run GS, i get the

following Python error ...

```
Fatal Python error: init_fs_encoding: failed to get the Python codec of the filesystem encoding

Python runtime state: core initialized

ModuleNotFoundError: No module named 'encodings'
```

A quick google did not yield any promising solutions but I will look again tomorrow. If anyone can suggest possible reasons/solutions I'd be very grateful. I know little about that pesky snake and less about how to treat one constricted by the beast :).

*From: Maxim Reznik <reznikmm@gmail.com>*
*Date: Sat, 10 Jun 2023 03:25:24 -0700*

Probably something wrong with your Python installation. I've tried in GNAT Studio console:

```
>>> import encodings

>>> print(encodings.__file__)

/tmp/gs/share/gnatstudio/python/lib/python3.9/encodings/__init__.py
```

While if I run system packaged Python in my Ubuntu:

```
$ python3

>>> import encodings

>>> print(encodings.__file__)

/usr/lib/python3.10/encodings/__init__.py

$ dpkg-query -S /usr/lib/python3.10/encodings/__init__.py

libpython3.10-minimal:amd64: /usr/lib/python3.10/encodings/__init__.py
```

So, it's part of libpython3.10-minimal

*From: Rod Kay <rodakay5@gmail.com>*
*Date: Wed, 28 Jun 2023 06:47:39 +1000*

It turns out that gnatstudio expects '/usr/share/gnatstudio/python' to contain or point to the root of an OS's Python installation. So a simple soft link to '/usr' fixed this problem.

The only other problem was a deprecated Python module, which was very easy to patch/fix.

So now gnatstudio builds/runs on Archlinux with all the bells/whistles.

A final thanks, Maxim, for your help.

## GCC 13.1.0 for Apple Silicon

*From: Simon Wright <simon@pushface.org>*
*Subject: [ANN] GCC 13.1.0 for Apple silicon*
*Date: Wed, 17 May 2023 20:23:04 +0100*
*Newsgroups: comp.lang.ada*

See new GCC 13.1.0 releases for aarch64-apple-darwin (i.e. Apple silicon), both native and cross compilation to arm-eabi, at https://github.com/simonjwright/distributing-gcc/releases

## GWindows 29-May-2023

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Subject: Ann: GWindows release, 29-May-2023*
*Date: Mon, 29 May 2023 09:19:11 -0700*
*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):

GWindows release, 29-May-2023 [revision 480]

* Fixes: color picker dialog, mouse wheel methods

478: Contribution: added package GWindows.Pipes

477: Contribution: added package GWindows.Timers

476: Contribution: added package GWindows.Persistence_IO

466: Contribution: initial release of package Office_Applications for helping creating office-like applications.

GWindows release, 13-Nov-2022 [revision 459]

458: GWindows.Common_Controls. Ex_List_View: added `Using_Payloads` to the enumerated type `Comparison_Technique_Type`.

With this choice, sorting runs 100x faster.

451: GWindows.Common_Controls. Ex_List_View: added `As_Strings_Default` to the enumerated type `Comparison_Technique_Type` (sorting runs faster if default alphabetical sorting is desired).

449: GWindows.Application: added procedure `Add_To_Recent_Documents`.

Windows Explorer & Desktop puts the name on top of various "recent documents" lists, for instance in the task bar.

447: GWindows.Common_Controls. Ex_List_View: massive speedup on sorting of large lists (e.g. 6x faster for 20,000 items).

GWindows release, 18-Jun-2022 [revision 440]

* Installer: ResEdit.xml configuration file for the ResEdit

Resource Editor is automatically created and set up for current GNAT installation(s), GWindows and GWenerator.

* Fixed a few 32/64 bit incompatibilities in GWindows.Windows and GWindows.Common_Controls.Ex_List_View.

* Fixed various GNATCOM issues.

GWindows Project site:

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

GWindows GitHub clone:

https://github.com/zertovitch/gwindows

*From: Drpi <314@drpi.fr>*
*Date: Mon, 29 May 2023 21:55:27 +0200*

What do you mean by "pure Ada" ?

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Mon, 29 May 2023 16:59:39 -0700*

IIRC, there are a few GNAT-only attributes, like Unrestricted_Access, used. No big deal.

But good point, I could check "purity" with the ObjectAda compiler.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Tue, 30 May 2023 09:56:13 +0200*

I took a quick look. Wouldn't all of Gnatcom need to be replaced?

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Sat, 3 Jun 2023 23:09:18 -0700*

Good question.

When I compile a project using GWindows, GNAT uses 10 of the 53 GNATCOM packages.

One GNATism is 4x 'Unrestricted_Access in GNATCOM.Types, for accesses such as:

```
VARIANT_MISSING : aliased constant
   VARIANT :=  (
    VT_ERROR, 0, 0, 0, u => (Which => 8,
    scode => DISP_E_PARAMNOTFOUND));
PVARIANT_MISSING :
   Pointer_To_VARIANT :=
    VARIANT_MISSING'Unrestricted_Access;
```

that could be either resolved into a standard Ada form or exiled into another package (GWindows doesn't need them).

Something tougher is a couple of intrinsic imports (sync_add_and_fetch, sync_sub_and_fetch):

```
  function sync_add_and_fetch
      (Ref : access Interfaces.Unsigned_32;
       Add : Interfaces.Unsigned_32)
    return Interfaces.Unsigned_32
   with Import,
         Convention => Intrinsic,
         External_Name =>
              "__sync_add_and_fetch_4";
```

which seems to be specific to GCC (and actually, not even all versions of GCC...)

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Sat, 17 Jun 2023 02:18:05 -0500*

This looks like an atomic operation. A portable Ada definition of such operations is found in C.6.1-C.6.4 of Ada 2022. Probably those could be used to replace the operation (of course, that would limit one to compilers supporting that part of Ada 2022; dunno if anyone is doing that yet).

## LEA 0.87

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Subject: Ann: LEA v.0.87*
*Date: Mon, 29 May 2023 09:29:20 -0700*
*Newsgroups: comp.lang.ada*

LEA is a Lightweight Editor for Ada

Web site: http://l-e-a.sf.net/

Source repository #1: https://sf.net/p/l-e-a/code/HEAD/tree/

Source repository #2: https://github.com/zertovitch/lea

Changes since last announcement here:

- Added auto insert feature: e.g. typing `(` inserts `)`.
- Added color theme Solarized Light.
- Added a "stealth mode" in which LEA doesn't leave traces in the registry.
- Editor adds `--  ` if the cursor is within a comment when the Return key is pressed (consequence: a comment is split into two comments).
- If the cursor is within a string literal when the Return key is pressed, the string literal is split into two valid string literals with a `&` between them.

- Added unhandled exception information to message list
- Tabs with the various file names
- LEA doesn't write scilexer.dll as a file; thus, it runs as a portable application (in the sense: you can run it from a read-only drive directly, without installation)
- Added a Build & Run button (for the HAC compiler).

Features:

- multi-document
- multiple undo's & redo's
- multi-line & multi-point edit, rectangular selections
- color themes, easy to switch
- duplication of lines and selections
- syntax highlighting
- parenthesis matching
- bookmarks

Currently available on Windows.

Gtk or other implementations are possible: the LEA_Common[.*] packages

are pure Ada, as well as HAC.

*From: Drpi <314@drpi.fr>*
*Date: Mon, 29 May 2023 21:56:35 +0200*

Just missing Alire and ALS compatibility ;)

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Mon, 29 May 2023 22:27:14 -0700*

Alire: are you missing a LEA crate?

ALS: = Ada language server?

*From: Drpi <314@drpi.fr>*
*Date: Tue, 30 May 2023 08:03:19 +0200*

> Alire: are you missing a LEA crate?

Why not but I was thinking about compiling/running Alire projects from LEA.

> ALS: = Ada language server?

That's it. Auto-completion and mouse-over documentation in LEA.

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Wed, 31 May 2023 20:18:57 -0700*

> Why not but I was thinking compiling/running Alire projects from LEA.

Good idea! For instance the "Build & Run" command (the green button) would launch "alr run" in that context.

Added to the to-do list.

Side note: a cool project would be a graphical tool, "Alire Explorer" (good name to be found) with buttons for the key Alire commands, a box displaying the contents of "alr show", ...

Perhaps something to be made with GNOGA.

>> ALS: = Ada language server?

> That's it. Auto-completion and mouse-over documentation in LEA.

Also added to the to-do list.

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Date: Fri, 9 Jun 2023 14:41:27 -0700*

LEA is now available on Alire (https://alire.ada.dev/) !

alr index --update-all

alr get lea

cd lea <-- here you press the Tab key to complete

alr build

lea

## Ayacc and Aflex 2023

*From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>*
*Subject: Re: Status of ayacc and aflex?*
*Date: Wed, 31 May 2023 13:42:18 -0700*
*Newsgroups: comp.lang.ada*

Old thread, but since some search engines point to here as top hit when searching for "ayacc" and "aflex", it is worth mentioning that the new developments (as of mid 2023) of ayacc and and aflex are located here:

https://github.com/Ada-France/ayacc

https://github.com/Ada-France/aflex

## PragmAda Reusable Components

*From: Pragmada Software Engineering <pragmada@pragmada.x10hosting.com>*
*Subject: [Reminder] The PragmAda Reusable Components*
*Date: Thu, 1 Jun 2023 10:38:17 +0200*
*Newsgroups: comp.lang.ada*

The PragmARCs are a library of (mostly) useful Ada reusable components provided as source code under the GMGPL or BSD 3-Clause license at https://github.com/jrcarter/PragmARC.

This reminder will be posted about every six months so that newcomers become aware of the PragmARCs. I presume that those who want notification when the PragmARCs are updated have used Github's notification mechanism to receive them, so I no longer post update announcements. Anyone who wants to receive notifications without using Github's mechanism should contact me directly.

## Qplt

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Subject: Ann: Qplt*
*Date: Fri, 2 Jun 2023 17:49:20 +0200*
*Newsgroups: comp.lang.ada*

I have created Qplt (Quick Plot), and Ada-GUI program to quickly produce a plot of a data set, and make it publicly available in hopes that it might prove useful. The program automatically selects axis ranges and tick intervals. The user may select whether points, lines, or both are plotted, and supply a title and axis labels.

Qplt is available at

https://github.com/jrcarter/Qplt

# Ada and Operating Systems

## GCC 13.1.0 (x86_64) on Ventura 13.3.1

*From: Bill Findlay <findlaybill@blueyonder.co.uk>*
*Subject: Trying GCC 13.1.0 (x86_64) on Ventura 13.3.1*
*Date: Sat, 29 Apr 2023 00:55:53 +0100*
*Newsgroups: comp.lang.ada*

Hi Simon,

Many thanks for the x86 macOS build of GNAT. Does it incorporate front-end updates since the Sep 30 build of gnat-12.2.0-1?

> which gnat

> /opt/gcc-13.1.0/bin/gnat

Using the command:

> GCC -c -I./ -I../Source -funwind-tables -gnatl12j96 -gnatw.e -gnatwD -gnatwH -gnatwP -gnatwT -gnatw.W -gnatw.B -gnatwC -gnatw.u -gnatyO -gnatw.Y -gnatw.N -fdata-sections -ffunction-sections -gnatfn -mtune=native -Ofast -fno-stack-check -fomit-frame-pointer -flto -I /Users/wf/KDF9/emulation/Source/ee9.adb

I got:

> clang (LLVM option parsing): Unknown command line argument '-x86-pad-for-align=false'. Try: 'clang (LLVM option parsing) --help'

> clang (LLVM option parsing): Did you mean '--x86-slh-loads=false'?

> gnatmake: "/Users/wf/KDF9/emulation/Source/ee9.adb" compilation error

*From: Simon Wright <simon@pushface.org>*
*Date: Sat, 29 Apr 2023 16:08:04 +0100*

Bill Findlay <findlaybill@blueyonder.co.uk> writes:

> Does it incorporate front-end updates since the Sep 30 build of gnat-12.2.0-1?

It includes whatever changes AdaCore & fellow maintainers have made! From here <https://gcc.gnu.org/gcc-13/changes.html>,

Ada

Traceback support added in RTEMS for the PPC ELF and ARM architectures.

Support for versions older than VxWorks 7 has been removed.

General improvements to the contracts in the standard libraries.

Addition of GNAT.Binary_Search.

Further additions and fixes for the Ada 2022 specification.

The Pragma SPARK_Mode=>Auto is now accepted. Contract analysis has been further improved.

Documentation improvements.

## Ada FreeDos/DOS Experiences

*From: Hou Van Boere <houvanboere@gmail.com>*
*Subject: Please Share Ada -Freedos - Dos experiences*
*Date: Sat, 27 May 2023 08:44:55 -0700*
*Newsgroups: comp.lang.ada*

Hi Everyone

I am thinking about using FreeDos as a kind of RTOS. The application is to control scientific instruments so portability is a non-issue.

Can you please share bits and pieces about running Ada on FreeDos (or MS DOS)

*From: Joakim Strandberg <joakimds@kth.se>*
*Date: Sat, 27 May 2023 12:49:36 -0700*

I wrote about how to get DJGPP compiler on DOS: https://www.reddit.com/r/ada/comments/vrhsv5/how_to_install_gnat_314b_on_freedos_13/

I recommend installing a recent version of DJGPP, you will be able to use a lot of the Ada language except for tasking which DJGPP does not support on FreeDos.

I have been looking for an Ada83 or Ada95 compiler for DOS which compiles real-mode executables but the ones I found are still proprietary and can be bought. It indicates there are still old systems on old hardware still in use.

I haven't built something on DOS, just toying with the idea. I've successfully been able to execute my Advent of code solutions for 2022 on FreeDOS. I've also successfully switched to VGA mode and putting pixels on the screen and switching back to text mode from an Ada application. I did it by interfacing with C code that had assembler embedded, if I remember correctly.

*From: Joakim Strandberg <joakimds@kth.se>*
*Date: Sat, 27 May 2023 13:02:58 -0700*

Another idea is to use the ObjectAda 7.0 compiler (free version) from 1996 that can be downloaded here:

https://archive.org/details/ObjectAdaSE7

It runs on Windows 95/98 but looking at the documentation for the ObjectAda compiler it says it is possible to use the compiler to create executables for DOS by using a DOS Extender. I haven't tried it but it should be possible to get working. Unfortunately there are limitations with the free version. One good thing is that it is possible to use tasks freely for creating a FreeDOS application but one must restrict one-self to Ada95 since the compiler is from 1996.

There is a professional version of ObjectAda from 2002 that can be downloaded here: https://vetusware.com/download/ObjectAda%207.2.2%20Enterprise%207.2.2/?id=17315

I've tested it and it works but the documentation no longer talks about being able to create executables for FreeDOS. Maybe it can still be used to make executables for FreeDOS?

*From: Joakim Strandberg <joakimds@kth.se>*
*Date: Sat, 27 May 2023 13:07:30 -0700*

However, the biggest obstacle for using FreeDOS is hardware support. FreeDOS depends upon BIOS and all motherboards since 2020 no longer support BIOS. Does anybody know of any hardware produced today that supports FreeDOS?

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 27 May 2023 14:54:28 -0700*

Thanks Joakim! this is super helpful.

I downloaded the compiler cd.

I have tried this:

https://github.com/andrewwutw/build-djgpp

It looks helpful to build dlgpp but it does not work well enough on Trisquel Linux. I find that building GCC on current or old Slackware versions seems to work well and I am going to re-try this project. I know I will have to re-run with --enable-languages=c,Ada later but at least it should set up most of what is needed.

Your Freedos environment tips will help a lot.

I just bought my son a new computer and I am kind of depressed after. The store was huge but completely geared towards gaming. It seems like today's computers are not well suited for hardware interfacing and hacking with electronics. There was way more expansion in the past and I hate having to configure for legacy bios. I think this will be dropped soon too and then we will be stuck

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 27 May 2023 14:55:28 -0700*

P.S I use less than half of Ada 95 so this compiler could help a lot.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Sun, 28 May 2023 01:00:21 +0200*

On 2023-05-27 17:44, Hou Van Boere wrote:

> Can you please share bits and pieces about running Ada on FreeDos(or MS DOS)

I used Ada (83) (Janus/Ada and Meridian Ada) on DOS PCs in the 80s and 90s. It was much like writing command-line applications for Linux or Windows today. I also did some low-level stuff, trapping key strokes and doing graphics. But if you're thinking of using DOS as an RTOS then that's probably not very helpful for you.

RR Software (rrsoftware.com) continues to sell an Ada-83 DOS compiler, and may be able to provide an Ada-95 DOS compiler on request. Their prices are reasonable: the personal edition of their Ada-95 Windows compiler is $195.

Or you could look at the MaRTE OS RTOS (https://marte.unican.es/) which is written mostly in Ada and supports GNAT compilers. I don't know how that would compare in terms of ease of getting things set up or developing S/W for it.

*From: Keith Thompson <keith.s.thompson+u@gmail.com>*
*Date: Sat, 27 May 2023 16:31:39 -0700*

Joakim Strandberg <joakimds@kth.se> writes:

> There is a professional version of ObjectAda from 2002 that can be downloaded

I wonder if those are authorized copies. I suspect they aren't.

Aonix no longer exists, but apparently its assets are now owned by PTC, which still sells (a much newer version of) ObjectAda. https://www.ptc.com/en/products/developer-tools/objectada

The copy on archive.org is of a CD whose label says "This edition of ObjectAda is not licensed for development of commercial software. This CD may not be re-sold. It does have an "All rights reserved" copyright message.

(I worked for Aonix many years ago, but I have no current connection with them or their successors.)

*From: Drpi <314@drpi.fr>*
*Date: Sun, 28 May 2023 13:01:17 +0200*

> However, the biggest obstacle for using FreeDOS is hardware support.

Do you really need to use old PC hardware ?

On a PC (and ARM), you can also run QNX which is a real-time micro-kernel OS. It is a commercial product but is free for education and research.

On PCs it is currently easy to use PCIe extension boards. Like FPGA boards.

Also, there are very powerful non x86 (mostly ARM) hardware today. Most of these boards have PCIe ports to easily add extension boards.

*From: Drpi <314@drpi.fr>*
*Date: Sun, 28 May 2023 19:42:08 +0200*

I forgot to say that AdaCore sells a Ada compiler for some QNX versions (7.x +) but I don't know if there is a free version for education/research.

### ARM 64-bit Binary Support

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Subject: ARM 64-bit binary support*
*Date: Sat, 24 Jun 2023 13:22:09 +0200*
*Newsgroups: comp.lang.ada*

Recently Linux Fedora and Ubuntu distributions stopped ARMv7 support (32-bit).

I added 64-bit architecture to the repositories of the following libraries for Debian, Fedora and Ubuntu:

- Ada industrial control widget library

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

- Fuzzy machine learning framework

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

- Fuzzy sets, logic, numbers

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

- GtkAda (pre-built)

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

- GtkAda contributions

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

- MAX! cube GUI for management of indoor radiator thermostats

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

- Interval arithmetic

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

- Measurement units

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

- Simple component

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

- String editing, UTF-8 issues

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

- Table management

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

ARMv7 builds are continued for the last official releases of the corresponding OSes.

## References to Publications

### Ada 2022 LRM by Springer

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Subject: Ada 2022 Language Reference Manual to be Published by Springer*
*Date: Wed, 14 Jun 2023 06:49:45 -0000*
*Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc*

-------------------------------------------------

FOR IMMEDIATE RELEASE

Ada 2022 Language Reference Manual to be Published by Springer

-------------------------------------------------

Lisbon, Portugal, June 14, 2023 - Ada-Europe today announced, at its 27th International Conference on Reliable Software Technologies (AEiC 2023), that the Ada 2022 Language Reference Manual (LRM) will be published by Springer in its LNCS series later this year.

Ada 2022 is the latest edition of the Ada programming language standard, technically denominated ISO/IEC 8652:2023, which was formally approved and officially published by ISO, the Geneva-based International Organization for Standardization, on May 2, 2023.

The Ada 2022 LRM is available online:

www.ada-auth.org/standards/ada22.html.

An overview of Ada 2022 is at:

www.ada-auth.org/standards/overview22.html.

To mark this official milestone, and in continuation of its established practice, Ada-Europe undertook to support the production of the new LRM as a dedicated issue of the Springer-published LNCS series.

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of the Ada programming language in academia, research and industry. Its flagship event is the annual International Conference on Reliable Software Technologies, a high-quality technical and scientific event that has been successfully running in the current format for the last 27 years. Ada-Europe has member organizations in Belgium, Denmark, France, Germany, Spain, and Switzerland, as well as individual members in many other countries. For information about Ada-Europe, its charter, activities and sponsors, please visit: www.ada-europe.org. Ada-Europe is headquartered in Brussels, Belgium.

A PDF version of this press release is available at www.ada-europe.org.

Organization Contacts

Ada-Europe

Tullio Vardanega, Ada-Europe President

president@ada-europe.org

Press Contacts

Ada-Europe

Dirk Craeynest, Ada-Europe Vice-president

c/o KU Leuven, Department of Computer Science

dirk.craeynest@cs.kuleuven.be

-------------------------------------------------

(VAda2022.1)

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Wed, 14 Jun 2023 01:20:19 -0700*

> The Ada 2022 LRM is available online:

> www.ada-auth.org/standards/ada22.html.

This is still Draft 35. The final version is not yet available. See also https://groups.google.com/g/comp.lang.ada/c/P26SS3L7kA0 - Ada 23 at Last!

*From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>*
*Date: Wed, 14 Jun 2023 14:13:33 -0000*

AdaMagica <christ-usch.grein@t-online.de> wrote:

>This ist still Draft 35. The final version is not yet available.

Note that the page at the above URL mentions:

"This is draft 35. This draft contains all ARG-approved AI12s. This is the draft that has been submitted to complete the standardization process."

So draft 35 *is* what was submitted to ISO.

Randy, the RM editor, is aware that this and a few other web pages still have to be updated now ISO published the new RM, and he assured me after the WG9 meeting yesterday that this is on his "to do list".

>See also https://groups.google.com/g/comp.lang.ada/c/P26SS3L7kA0 - Ada 23 at Last!

That message claimed about the ISO document: "The ToC is very different from Draft 35."

While draft 35 is what was submitted to ISO, the documents indeed are not identical. Though I would not say the ToC's are "very different".

Yes, the introductory chapters in the ISO document are slightly different from those in the RM on ada-auth.org, and there's no Annex on "Obsolescent Features" nor a "Glossary" (that was removed in draft 35 anyway). All this is due to specific requirements that ISO has for its standards. There are more differences, such as the ISO document not having any paragraph numbers as those are not allowed in ISO standards.

But the bulk of the ToC is identical, apart from those differences required by ISO. Most importantly: the described language in both documents is identical.

*From: Egil H H <ehh.public@gmail.com>*
*Date: Wed, 14 Jun 2023 09:11:05 -0700*

On Wednesday, June 14, 2023 at 3:13:36 PM UTC+1, Dirk Craeynest wrote:

> But the bulk of the ToC is identical, apart from those differences required by ISO. Most importantly: the described language in both documents is identical.

The clause numbering is not the same, as clause 1 has been split into 4 clauses in the ISO version, so clause `2 Lexical Elements` in the Draft corresponds to `5 Lexical Elements` in the ISO version.

And (at least) one bug in the ISO ToC, `7.3.4 Delta Aggregates` and `7.3.5 Container Aggregates` are collapsed beneath `7.3.3. Array Aggregates`, even though the subclause level is the same.

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Wed, 14 Jun 2023 09:15:37 -0700*

Dirk Craeynest schrieb am Mittwoch, 14. Juni 2023 um 16:13:36 UTC+2:

> So draft 35 *is* what was submitted to ISO.

Yes; I know...

> That message claimed about the ISO document: "The ToC is very different from Draft 35."

Funny, when I first opened the preview, the complete table of contents with page numbers could be read. The ISO document had far fewer pages than Draft 35 (951 pages). I wondered how this could be...

Now the ToC is without page numbers, so I cannot compare.

If you compare the ISO ToC and the Draft 35 one, you'll see that clause and subclause numbers differ. So old references like RM 3.5 will lead astray.

--- ISO locuta, causa finita. ---

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Wed, 14 Jun 2023 10:03:26 -0700*

Egil H H schrieb am Mittwoch, 14. Juni 2023 um 18:11:07 UTC+2:

> And (at least) one bug in the ISO ToC,

Not only this. The whole of 7.4 to 7.10 is collapsed under 7.3.3.

*From: Keith Thompson <keith.s.thompson+u@gmail.com>*
*Date: Wed, 14 Jun 2023 12:47:29 -0700*

dirk@orka.cs.kuleuven.be. (Dirk Craeynest) writes:

> the ISO document not having any paragraph numbers as those are not allowed in ISO standards.

Is disallowing paragraph numbers a recent change I have a copy of the 2011 ISO C standard, ISO/IEC 9899:2011 (E), and it definitely has paragraph numbers. (Which are extremely useful, BTW; it seems silly for ISO to disallow them.)

*From: Simon Wright <simon@pushface.org>*
*Date: Thu, 15 Jun 2023 20:36:35 +0100*

Egil H H <ehh.public@gmail.com> writes:

> And (at least) one bug in the ISO ToC,

From my point of view, never mind the bug, this makes the ISO document a white elephant.

The stability of the clause numbering, and the hyperlinking, make the RM the valuable document that it is.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Sat, 17 Jun 2023 02:49:12 -0500*

Actually, paragraph numbers weren't allowed back in the Ada 83/Ada 95 days. So the original ISO versions didn't have them. You can use them in ISO documents now (I don't know when this changed), but you have to get a special waiver to do so - for *every* individual standard that you want to have them (that's a recent change, for the worse). And if we added them to the ISO version (after getting the appropriate waiver -- which I didn't know about for this last round of standardization), they'd be different than the ones in the RM (because they wouldn't allow versioning or inserted numbers). That doesn't seem helpful to me, YMMV.

ISO no longer lets us be compatible with the clause numbering of previous versions -- ALL standards have to follow their numbering for initial stuff. They've also changed from requiring not using Annexes I and O (since they're easily confused with chapters (nope, now sections (nope, now clauses)) -- to requiring having Annexes I and O.

Bob Duff explained it best: The people maintaining the "standards for standards" have made no attempt to keep upward compatibility in their work (unlike us Ada people). Every standard in existence has to be changed substantially with each new edition in order to meet the ever-changing requirements. It's hard to believe that these people don't understand (or don't care) that these standards are used for a very long time.

Randy Brukardt, Project Editor, ISO/IEC 8652

## Ada and Other Languages

### Java and Python get "record" Type after 40 Years.

*From: Nasser M. Abbasi <nma@12000.org>*
*Subject: Java and Python have just discovered "record" type finally after 40 years.*
*Date: Fri, 12 May 2023 12:50:14 -0500*
*Newsgroups: comp.lang.ada*

Java 14 now has "Record" !

"records are meant to be data carriers"

https://www.digitalocean.com/community/tutorials/java-records-class

And Python 3.7 now has records, they call it "data class"

https://realpython.com/python-data-classes/

"One new and exciting feature coming in Python 3.7 is the data class. A data class is a class typically containing mainly data"

What took them so long? Pascal and Ada had records from day one, only 40 years ago or so.

*From: Richardthiebaud <thiebauddick2@aol.com>*
*Date: Fri, 12 May 2023 14:58:52 -0400*

And Cobol had them 63 years ago.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Fri, 12 May 2023 23:33:54 +0200*

Pascal had them in 1970. Algol, I think, had them in 1960.

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Sat, 13 May 2023 10:13:22 +0300*

On 2023-05-12 19:50, Nasser M. Abbasi wrote:

> What took them so long?

Java and Python have classes, which have records as a special case, if the term "record" is understood as in most other languages, including Ada.

But it seems that the Java 14 "record" is not quite the same as an Ada record, because Java 14 records are meant to be immutable data carriers, not mutable data structures. Still, Java 14 records are described as a (very) special case of classes.

> Pascal had them in 1970. Algol, I think, had them in 1960.

Algol 60 did not have records, only arrays.

Algol W, a precursor to Pascal, had them in 1966.

Simula had them in 1967. (Wikipedia says "In 1966 C. A. R. Hoare introduced the concept of record class construct".)

Algol 68 had them in 1968.

Pascal had them in 1970, as you say.

*From: Luke A. Guest <laguest@archeia.com>*
*Date: Sat, 13 May 2023 12:18:04 +0100*

What about COBOL and LISP?

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Sat, 13 May 2023 19:53:45 +0300*

On 2023-05-13 14:18, Luke A. Guest wrote:

> What about COBOL and LISP?

As I understand it (but I don't claim to be expert), the early COBOL languages could describe the structure of file records, and of working-storage objects, as nested sequences of components and sub-records, but each such description defined a _single_ "record" object, not a "record" data-type that could have many instances. So if you wanted to have two record objects with the same structure, you had to duplicate the whole record description.

However, Wikipedia says that the COBOL record structure inspired records for Pascal.

Early LISP languages did not have record types, AFAIK. But you could of course use lists to program record-like data structures.

*From: J-P. Rosen <rosen@adalog.fr>*
*Date: Sun, 14 May 2023 08:46:15 +0200*

Le 13/05/2023 à 18:53, Niklas Holsti a écrit :

> So if you wanted to have two record objects with the same structure, you had to duplicate the whole record description.

AFAIR, COBOL didn't have types, but you could define a variable LIKE another one.

> Early LISP languages did not have record types, AFAIK. But you could of course use lists to program record-like data structures.

Of course, in LISP there is only one structure, for data and programs alike: the list!

*From: Nasser M. Abbasi <nma@12000.org>*
*Date: Sun, 14 May 2023 02:20:42 -0500*

On 5/14/2023 1:46 AM, J-P. Rosen wrote:

> Of course, in LISP there is only one structure, for data and programs alike: the list!

This is similar to Mathematica. I programmed a little in Lisp, and it was kinda fun.

In Mathematica, its main data struct is also the list and list of lists and list of list of lists and so on.

a={1,2,3};

a={{1,2,3},{4,5,6}};

Everything in Mathematica is pretty much build using lists.

Few years ago, Wolfram introduced Association, which acts like a RECORD. It is really like a dictionary. It has key->value pairs so one can do:

myData = <| "name"->"me","age"->99 |>

To read the value of a field one uses myData["name"] or myData["age"].

It is amazing how people can program so much code using only just a list as the main basic data structure and be able to get away with it :)

I think RECORD is the most important data structure myself.

Without a RECORD (called struct in C), programming is much harder. This is what Java and Python have discovered just now. I guess the language designers of these languages never bothered to look at Pascal or Ada before.

But better late than never I guess.

*From: Luke A. Guest <laguest@archeia.com>*
*Date: Sun, 14 May 2023 10:45:36 +0100*

On 14/05/2023 07:46, J-P. Rosen wrote:

> Of course, in LISP there is only one structure, for data and programs alike: the list!

Well, that's not true anymore, especially not in common lisp which has a variety of data structures including records, I was quite surprised to see that when I was looking at it last year.

*From: Luke A. Guest <laguest@archeia.com>*
*Date: Sun, 14 May 2023 10:49:17 +0100*

On 14/05/2023 08:20, Nasser M. Abbasi wrote:

> This is what Java and Python have discovered just now.

I think people might finally be realising that you can't do everything with only one programming paradigm.

*From: Ben Bacarisse <ben.usenet@bsb.me.uk>*
*Date: Sun, 14 May 2023 11:37:21 +0100*

"J-P. Rosen" <rosen@adalog.fr> writes:

> Of course, in LISP there is only one structure, for data and programs alike: the list!

LISP had S-expressions -- pairs of atoms or other S-expressions. A list was just a special case. Many other structures could be built using S-expressions. An important one was that association list -- a list of (key . value) pairs that was often used very much like a record type (though it's quite a different beast).

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Sun, 14 May 2023 12:39:02 +0200*

On 2023-05-14 08:46, J-P. Rosen wrote:

> Of course, in LISP there is only one structure, for data and programs alike: the list!

In the LISP I learned, there were only S-expressions (SEXes). A SEX is either an atom or a list of SEXes. Another way of putting it was there were atoms and lists of atoms or lists. Either way, there were also atoms.

*From: J-P. Rosen <rosen@adalog.fr>*
*Date: Sun, 14 May 2023 17:10:23 +0200*

Le 14/05/2023 à 12:39, Jeffrey R.Carter a écrit :

> A SEX is either an atom or a list of SEXes.

Right, but I would define atoms as the basic data, not a data /structure/. Oh well, just a matter of definition...

*From: Ben Bacarisse <ben.usenet@bsb.me.uk>*
*Date: Sun, 14 May 2023 16:14:33 +0100*

"Jeffrey R.Carter" <spam.jrcarter.not@spam.acm.org.not> writes:

> A SEX is either an atom or a list of SEXes.

I never saw a LISP S-expressions defined that way. Did this list really not have a "dotted pair" as the basic structure with lists being simply a special case?

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Sun, 14 May 2023 18:56:27 +0200*

On 2023-05-14 17:14, Ben Bacarisse wrote:

> Did this list really not have a "dotted pair" as the basic structure with lists being simply a special case?

The book I learned from (/Let's Talk LISP/ by Laurent Siklóssy, 1976) introduces dotted pairs in chapter 10.7.1 (out of 12 chapters) on page 145 (out of 213, excluding appendices and index). Chapter 10 deals with the internal representation of data in LISP. The implication is that they were not considered part of the normal use of the language.

S-expressions, on the other hand, are introduced in chapter 1.1 on page 2. The book also presents the grammar

S-expression ::= atom | list

list ::= '(' inside ')'

inside ::= empty | S-expression | S-expression inside

empty ::=

*From: Ben Bacarisse <ben.usenet@bsb.me.uk>*
*Date: Mon, 15 May 2023 02:11:55 +0100*

"Jeffrey R.Carter" <spam.jrcarter.not@spam.acm.org.not> writes:

> The book I learned from (/Let's Talk LISP/ by Laurent Siklóssy, 1976)

Do you still have it? Does it discuss association lists? I'd call them a normal part of LISP and it would be odd to force the associations to be lists rather than pairs. Does Siklóssy imply that an ASSOC list is a list of lists of length 2, or does he not discuss them until the very end?

> The book also presents the grammar

That's an interesting way to simplify things for the leaner though I would not have chosen to use a term that already had another meaning by 1976. The author could have used something like L-expression and avoided any future confusion.

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Mon, 15 May 2023 12:44:13 +0200*

On 2023-05-15 03:11, Ben Bacarisse wrote:

> Do you still have it? Does it discuss association lists?

The Function ASSOC is discussed in Chapter 9 as an auxiliary function used by EVAL (Chapter 9 discusses the working of EVAL). It says

"ASSOC finds the value of a variable in the ALIST. The ALIST is a list of sublists of two SEXes each of the form (variable value-of-the-variable)."

In a footnote he notes that the ALIST could be a list of dotted pairs, which are defined in the next chapter.

I never did much with LISP after learning it, and never looked at any other textbooks, so he might have an idiosyncratic approach. Seems rather OT for c.l.a.

*From: Ben Bacarisse <ben.usenet@bsb.me.uk>*
*Date: Wed, 17 May 2023 01:24:32 +0100*

"Jeffrey R.Carter" <spam.jrcarter.not@spam.acm.org.not> writes:

> ASSOC finds the value of a variable in the ALIST. The ALIST is a list of sublists of two SEXes each of the form (variable value-of-the-variable).

Thanks. Is this a dialect made up for pedagogic purposes? I don't know of any practical LISP that went down this route.

> Seems rather OT for c.l.a.

Yes, it is. Happy to stop. I was just curious about where your use of terms originated and that now explained.

## Ada Scales Down!

*From: Hou Van Boere <houvanboere@gmail.com>*
*Subject: Ada Scales Down!*
*Date: Sat, 13 May 2023 17:17:50 -0700*
*Newsgroups: comp.lang.ada*

Hi Everyone

Just a little cross post:

https://sourceforge.net/p/gnucobol/discussion/cobol/thread/5f771109ad/

I am having so much fun with Ada again. I think the foreign binding examples on the net are horrible. With little wimpy inline packages, you can bring foreign code in easily.

Everyone complains about C but a teenager can fool around with it on a weekend and end up a C programmer a few years later. Ada does not present this way but in fact it is easy to write little wimpy programs in it just for fun and even wimpy programs will often need non-Ada libraries.

*From: Luke A. Guest <laguest@archeia.com>*
*Date: Sun, 14 May 2023 10:53:29 +0100*

On 14/05/2023 01:17, Hou Van Boere wrote:

> Hi Everyone

>

> Just a little cross post:

> https://sourceforge.net/p/gnucobol/discussion/cobol/thread/5f771109ad/

It's not 1979 anymore, you can use unicode in Ada and even lowercase letters. This is not Oberon where the language is stuck in the 70's where there was a limited character set available on keyboards. I think even COBOL can now accept lowercase keywords now, but I'm not sure about this.

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sun, 14 May 2023 06:59:25 -0700*

Hi Luke

I knew someone would mention this :) Most people program in lowercase with COBOL now. It is a personal preference. I use a smaller font and have more code on the screen with uppercase and I am just kinda retro about a lot of things. Think Amish using a computer :)

## Does Safer Mean Slower?

*From: Nasser M. Abbasi <nma@12000.org>*
*Subject: does a safer language mean it is slower to run?*
*Date: Wed, 7 Jun 2023 22:55:51 -0500*
*Newsgroups: comp.lang.ada*

Some folks in this thread

https://discourse.julialang.org/t/comparison-of-rust-to-julia-for-scientific-computing/78508

"I'm not an expert, but my feeling is that Rust is a "safer" language, which to me means it must be slower."

etc..

Some in that thread seem to argue that a safer language will/could be slower than otherwise.

Since Ada is known to be one of the safest languages, do others here feel there is any truth to this?

I thought that by having more type information in the language, the compiler will be able to make more optimizations (because it knows more), and hence the generated code should actually be faster, not slower with a language that is less safe?

I am not a compiler expert but what do others here think?

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Thu, 8 Jun 2023 09:57:14 +0300*

If a language needs run-time checks to ensure safety, those checks usually take some time, making for slower execution.

If a language has a type system and compilation-time (legality) rules such that the compiler can prove that some run-time checks are not needed, that reduces or eliminates the slow-down. This is the case for Ada.

The effect of type information on optimization is harder (at least for me) to understand. If the type information lets the compiler assume that some objects are not aliased, that can help optimization because more computation can be done in registers alone, without using main memory. This applies to Ada, but also applies to standard C, for example, although some people use non-standard C features (compiler options) to negate this.

However, when comparing the "speed" of two languages and their two implementations I think that the implementations usually matter more than the languages.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Thu, 8 Jun 2023 10:00:52 +0200*

On 2023-06-08 05:55, Nasser M. Abbasi wrote:

> "I'm not an expert, but my feeling is that Rust is a "safer" language, which to me means it must be slower."

I think the comparison is misplaced. Julia is an interpreted language, very slow, on par with Python. It has memory mapped arrays like Ada does, but lacks Python's precompiled modules. The syntax is wonderfully arbitrary and unpredictable...

If safety is prevention of logical errors (bugs) you and your team and people deploying the software could make, then techniques and processes determine the outcome. The language can only support certain techniques. Of these techniques and processes some may require run-time overhead. When people compare languages, they frequently do programming techniques instead. As it was observed many decades ago:

"Besides, the determined Real Programmer can write Fortran programs in any language."

And finally, if you are determined to use some technique, then lack of language support makes the language less safe. E.g. if you are in some agile programming league then semantic constraints imposed by Ada would make things only worse.

Even Brainf*ck might be the safest language under circumstances... (:-))

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Thu, 8 Jun 2023 10:50:44 +0200*

On 2023-06-08 05:55, Nasser M. Abbasi wrote:

> Since Ada is known to be one of the safest languages, do others here feel there is any truth to this?

Equivalent programs in compiled, non-GC languages have equivalent execution times. Robert Dewar famously had a set of equivalent Ada and C programs that produced identical machine code when compiled with gcc. So this is false.

The problem is getting equivalent programs. If the safe language includes run-time checks, then equivalent checks must be manually added to the unsafe language. Ada.Text_IO is not equivalent to C's I/O facilities. And so on.

One consequence of this is that both programs will be equally correct. What is usually compared is a correct (run-time checks) program in the safe language to an incorrect (no run-time checks) program in the unsafe language.

About optimization, Tartan made its living selling highly optimizing C compilers for TI chips, which came with a free C compiler. They also made highly optimizing Ada compilers, which did a better job of optimization than their C compilers. This was documented in

C vs Ada: arguing performance religion

(https://dl.acm.org/doi/10.1145/216578.216583)

which discusses four advantages Ada (83) has over C for optimization.

See also

Ada Outperforms Assembly: A Case Study

(https://www2.seas.gwu.edu/~adagroup/sigada-website/lawlis.html)

TI bought Tartan and sold its Ada compilers to DDC-I.

## Ada Practice

### Working around -ffreestanding Limitations

*From: Hou Van Boere <houvanboere@gmail.com>*
*Subject: Working around -freestanding limitations?*
*Date: Sat, 1 Apr 2023 05:26:37 -0700*
*Newsgroups: comp.lang.ada*

Hi Everyone.

I know there are several floss RTOS options for us but I don't really need all of the support they offer and they just make things more complex.

Here are my goals:

1)I want to build my own circuit board with a microprocessor not microcontroller.

2)I want to run with gcc/gnatmake ... -freestanding

3)I only need the Ada 83 subset, which I guess is pretty close to Ravenscar

What options do I have? I like to keep things small and simple when possible.

Thanks for reading

*From: Simon Wright <simon@pushface.org>*
*Date: Sat, 01 Apr 2023 14:35:23 +0100*

Hou Van Boere <houvanboere@gmail.com> writes:

> 2)I want to run with gcc/gnatmake ... -freestanding

-freestanding isn't an option for gnatmake; where does it come from?

> 3)I only need the Ada 83 subset, which I guess is pretty close to Ravenscaler

The Ada 83 subset is going to be larger than Ravenscar.

If you don't want an RTOS you could use one of the light runtimes, e.g. light-cortex-m0.

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 1 Apr 2023 09:12:54 -0700*

Could you tell me where to find the light runtimes? I have only worked with Ada on full desktops. Does the FSF version ship with light runtimes?

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Sat, 1 Apr 2023 19:14:28 +0300*

Certainly using most RTOS from Ada is more complex than using an Ada RTS from Ada.

Do you want to use tasking at all? Or just a single thread?

> 1)I want to build my own circuit board with a microprocessor not microcontroller.

Can you explain why? Input/output is often more complex with a microprocessor (I assume you mean something that could run a PC or a tablet) than with a microcontroller. A microprocessor may need a lot of complex initialization and driver SW which you can get in some RTOS but not in an Ada RTS. And I believe that circuit-board design is more complex for microprocessors than for microcontrollers, however I have no experience with either case.

> 3)I only need the Ada 83 subset, which I guess is pretty close to Ravenscaler

I see the Ada 83 tasking features as almost orthogonal to Ravenscar. Ada 83 has no protected objects, and all inter-task communication must be done with rendez-vous using task entries. Ravenscar forbids task entries and rendez-vous and substitutes protected objects. Both work, but Ravenscar is perhaps more resistant to deadlock errors.

*From: Drpi <314@drpi.fr>*
*Date: Sat, 1 Apr 2023 18:29:45 +0200*

Le 01/04/2023 à 18:12, Hou Van Boere a écrit :

> " -freestanding isn't an option for gnatmake; where does it come from?"

I guess it is -ffreestanding :

https://stackoverflow.com/questions/17692428/what-is-ffreestanding-option-in-gcc

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 1 Apr 2023 09:42:26 -0700*

Thanks for your feedback Niklas. I am new to Ravenscar (just spelled it wrong today), this is very helpful.

It would be nice to have threads but I am not sure I actually need them.

I have serviced scientific instruments for 24 years now. I want to start fabricating them. I will have some bumps along the way with PCB design but I am confident that it will work out.

I have been playing around with Ada since 2012 but I still have lots to learn and I don't program in the day so it is not my strong suit. The hardware side should work out but I am worried about the software end of things. Ada is lovely but massive. There are so many features, so many libraries(some of which are abandoned). There are only so many hours in a day.

I have a subset of the language I like and if I can just control CPU address and data lines, I shouldn't need a RTOS. Trying several of them out could take a great deal of time.

I don't seem to have any extra runtimes with my install:

gnatls -v

GNATLS 11.2.0

Copyright (C) 1997-2021, Free Software Foundation, Inc.

Source Search Path:

<Current_Directory>

/usr/lib64/gcc/x86_64-slackware-linux/11.2.0/adainclude

Object Search Path:

<Current_Directory>

/usr/lib64/gcc/x86_64-slackware-linux/11.2.0/adalib

Project Search Path:

<Current_Directory>

/usr/x86_64-slackware-linux/lib/gnat

/usr/x86_64-slackware-linux/share/gpr

/usr/share/gpr

/usr/lib/gnat

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 1 Apr 2023 09:50:28 -0700*

"Could you tell me where to find the light runtimes?"

I am just answering my own question to avoid wasting people's time. I found this:

https://github.com/AdaCore/bb-runtimes

*From: Drpi <314@drpi.fr>*
*Date: Sat, 1 Apr 2023 18:54:20 +0200*

> 1) I want to build my own circuit board with a microprocessor not microcontroller.

Nowadays, microprocessors are rare. Even x86 microprocessors could be named microcontrollers since they integrate many (not all) peripherals.

High end microcontrollers are very complex to initialize. Especially since they integrate security functionalities (like secure boot), SDRAM controllers, PCIe controllers, Gigabit Ethernet controllers, 3D GPUs, video encoders/decoders, camera interface, LCD interface, HDMI interface...

Even middle range microcontrollers are (very) complex.

Manufacturers provide drivers source code (in C) for all peripherals. They also provide tools to graphically set chip configuration and output C code to help the programmer.

Complexity depends on the chip you choose.

*From: Drpi <314@drpi.fr>*
*Date: Sat, 1 Apr 2023 18:55:36 +0200*

Le 01/04/2023 à 18:50, Hou Van Boere a écrit :

> I am just answering my own question to avoid wasting people's time. I found this:

> https://github.com/AdaCore/bb-runtimes

The best way is to use Alire https://alire.ada.dev/

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 1 Apr 2023 09:58:49 -0700*

Thanks DrPi

I will probably stick with what I know. Most of the instruments I work on have Motorola chips and parallel buses. I don't think I will use SPi, IC2 or dozens of other protocols/features found in most modern circuit boards.

*From: Drpi <314@drpi.fr>*
*Date: Sat, 1 Apr 2023 18:59:42 +0200*

Le 01/04/2023 à 18:42, Hou Van Boere a écrit :

> I don't seem to have any extra runtimes with my install:

Today, the easiest route is to use ARM based chips as there are maintained runtimes for them (through Alire and bbruntimes).

*From: Drpi <314@drpi.fr>*
*Date: Sat, 1 Apr 2023 19:02:15 +0200*

Le 01/04/2023 à 18:58, Hou Van Boere a écrit :

> I will probably stick with what I know. Most of the instruments I work on have Motorola chips

Great chips at their time but I'm afraid you'll have hard time compiling a dedicated GNAT compiler.

*From: Hou Van Boere <houvanboere@gmail.com>*
*Date: Sat, 1 Apr 2023 10:24:28 -0700*

I am sure you are right but still, you get the general idea.

Thermo Electron has pretty much bought most of the industry out. I will copy and paste, mix and match old stuff to re-implement instruments they don't care about anymore. I don't need to make anything cutting edge. The old stuff was more than good enough

*From: Drpi <314@drpi.fr>*
*Date: Sat, 1 Apr 2023 20:33:36 +0200*

Le 01/04/2023 à 19:24, Hou Van Boere a écrit :

> The old stuff was more than good enough

Indeed, an interesting project.

You first need an Ada cross-compiler. Here is a link about this : https://wiki.osdev.org/GNAT_Cross-Compiler

You also need a runtime. This is up to you to code it. You can use bbruntimes as a template. This can request modifications on your hardware. For example, the runtime needs a timer to track time. If your microprocessor does not have an embedded timer, you'll have to add one on your board.

Other links of interest :

https://forum.ada-lang.io/

https://github.com/ohenley/awesome-ada

Matrix rooms (https://matrix.org/clients) :

Ada news : https://matrix.to/#/#ada-lang:matrix.org

Ada language : https://matrix.to/#/#ada-lang:matrix.org

Alire : https://matrix.to/#/#ada-lang_Alire:gitter.im

Many other resources exist.

*From: philip...@gmail.com*
*<philip.munts@gmail.com>*
*Date: Wed, 5 Apr 2023 09:21:44 -0700*

I would suggest you look at my Linux Simple I/O Library: https://github.com/pmunts/libsimpleio

The Ada binding makes it pretty easy to build test fixtures, control devices, etc. I even used an Ada program to replace a multizone sprinkler controller.

Next, take a look at MuntsOS Embedded Linux: https://github.com/pmunts/muntsos

Together, they make it possible to replace many microcontroller applications with a Raspberry Pi or a BeagleBone or a PocketBeagle. With the Raspberry Pi family, it is very easy to fabricate custom boards using a Raspberry Pi Zero, CM3, or CM4 (least to most complex) as a CPU module. If you run Raspberry Pi OS instead of MuntsOS, it is even self hosting.

Currently I don't have any support for IEEE-488, though I have a USB interface and an old CalComp plotter on the shelf I've been meaning to play around with.

I'll be teaching a workshop at AdaEurope 2023 in Lisbon in June showing how all this works.

## Constancy of X'Address

*From: Niklas Holsti*
*<niklas.holsti@tidorum.invalid>*
*Subject: Constancy of X'Address?*
*Date: Wed, 5 Apr 2023 12:24:39 +0300*
*Newsgroups: comp.lang.ada*

A discussion in comp.arch (on the new C23 standard for C) brought up these questions, which I could not answer with confidence:

- Is the address of an object constant in Ada? That is, if I have some object X in an Ada program, do repeated applications of X'Address always return the same value?
- Does the answer depend on how X is allocated (created): on the library level, on the stack, or in a pool ("new")?

The issue behind this question is whether an Ada program could use garbage collection that moves objects around, for example a compacting collector.

*From: Maxim Reznik*
*<reznikmm@gmail.com>*
*Date: Fri, 7 Apr 2023 10:04:24 -0700*

If the type of the object is limited, then the object address is a constant. For other objects there is no such guarantee I guess.

*From: Jeffrey R.Carter*
*<spam.jrcarter.not@spam.acm.org.not>*
*Date: Fri, 7 Apr 2023 19:19:14 +0200*

'Address is discussed in ARM 13.3 (http://www.ada-auth.org/standards/aarm12_w_tc1/html/AA-13-3.html). (12.c) says "The validity of a given address depends on the run-time model; thus, in order to use Address clauses correctly, one needs intimate knowledge of the run-time model."

Under Implementation Advice, (15-16) say "The recommended level of support for the Address attribute is:

"X'Address should produce a useful result if X is an object that is aliased or of a by-reference type, or is an entity whose Address has been specified."

There is nothing specific about whether the value can change.

*From: Randy Brukardt*
*<randy@rrsoftware.com>*
*Date: Sat, 8 Apr 2023 04:03:02 -0500*

I agree with the other answers (for the most part). Not much is guaranteed about 'Address. But by-reference and aliased objects have to *work* like they are not moved. If the runtime can deal with moving such things, Ada doesn't care.

As a practical matter, most Ada implementations assume objects don't move. Janus/Ada assigns everything at compile-time, so the only time anything moves is when it is created or destroyed.

The big problem with garbage collection in Ada is that early finalization is not allowed (other than a few tiny exceptions in failure cases and [in post-Ada 22] certain function results.) So any object that might have a controlled part can never be garbage collected, even if there is no other use or access to it.

Changing that is a very hard problem, as you cannot allow finalization to happen at any instant or by any arbitrary task (if you did, every finalization would be a race scenario, and every Finalize routine would need dedicated locking). I've suggested allowing it for "unreachable objects" (not a useful definition by itself, it would need to be defined) at places where masters are being exited anyway (so finalization should be expected at those locations). But it's unclear if you can build a useful garbage collector that way (and what the overhead would be).

## Contracts in Generic Formal Subprogram

*From: Mockturtle*
*<framefritti@gmail.com>*
*Subject: Contracts in generic formal subprogram*
*Date: Sat, 8 Apr 2023 00:00:38 -0700*
*Newsgroups: comp.lang.ada*

Dear.all,

this is something that looked like a natural and nice idea to me, but the compiler disagreed :-): specifying contracts in formal subprograms in generic declarations. Actually, RM 12.6 does not prohibit this on a syntactic level (an aspect_specification part is included), but the compiler complains.

To understand what I mean, please check the following real code toy-zed (can you hear the grammar screaming?)

```
-----------------------
generic
  type Ring is private;
  with function Divides (Num, Den : Ring)
    return Boolean;
  with function Is_Invertible (X : Ring)
    return Boolean;
  with function Inv (X : Ring) return Ring
    with  Pre => Is_Invertible (X);

  with function Gcd (X, Y : Ring)
    return Ring
    with  Post => Divides (X, Gcd'Result)
          and Divides (Y, Gcd'Result);
package Pippo is
  -- stuff
end Pippo;
----------------------------------
```

The meaning I have in mind is something like

* For "Pre" aspect: whoever writes the function Inv can assume that X is invertible since Inv will never be called (by the package code, at least) with X not invertible. Also Inv cannot have a stricter pre-condition. In a sense, the package expects Inv to work correctly if and only if the pre-condition is true.

* For "Post" aspect: I expect that the result of GCD satisfies the post condition. Post conditions for the actual subprogram can be stricter, as long as the post condition of the formal parameter is satisfied. For example, if Ring is Integer, GCD could always return a positive value that divides both X and Y. The fact that the result is positive does not hurt.

Should the actual subprogram specify the same contract? I am not sure (and I guess this could be a stumbling block for the adoption of this idea). One could say that the actual subprogram gets a contract that is the AND of the actual subprogram and the contract specified in the generic declaration, it is up to the programmer to check that they are compatible. I guess the compatibility could be verified by the compiler itself in simple cases, but I expect that this could not be feasible in some cases (maybe of academic interest?).

*From: Dmitry A. Kazakov*
*<mailbox@dmitry-kazakov.de>*
*Date: Sat, 8 Apr 2023 10:02:31 +0200*

On 2023-04-08 09:00, mockturtle wrote:

> Should the actual subprogram specify the same contract? I am not sure (and I guess this could be a stumbling block for the adoption of this idea).

The general principle of substitutability is that the preconditions can be weakened, the postoconditions can be strengthened.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Sat, 8 Apr 2023 04:09:38 -0500*

Ada 2022 allows such contracts; Ada 2012 did not. (See 6.1.1, and specifically 6.1.1(1/5)). Whether your compiler has caught up, who knows.

Logically the contracts should "match" (with the weakening/strengthing that Dmitry mentioned), but that was too hard for Ada, so they're just additive. (A proper matching mechanism is more the sort of thing that SPARK does, Ada only enforces these contracts at runtime) That is, when you call through a generic formal subprogram, you enforce the preconditions of both the formal and the actual subprogram, and similarly for the postconditions. If they mismatch, you might not be able to make a successful call. If it hurts, don't do that. ;-)

*From: Simon Wright <simon@pushface.org>*
*Date: Sat, 08 Apr 2023 17:48:11 +0100*

GCC 12.2.0 accepts this code with -gnat2022.

*From: Mockturtle <framefritti@gmail.com>*
*Date: Sat, 8 Apr 2023 10:27:16 -0700*

On Saturday, April 8, 2023 at 6:48:14 PM UTC+2, Simon Wright wrote:

> GCC 12.2.0 accepts this code with -gnat2022.

True!  Cool...  In my opinion, contracts are among the coolest (and maybe more exclusive) features of Ada

*From: G.B. <bauhaus@notmyhomepage.invalid>*
*Date: Tue, 11 Apr 2023 07:56:45 +0200*

On 08.04.23 10:02, Dmitry A. Kazakov wrote:

> The general principle of substitutability is that the preconditions can be weakened, the postconditions can be strengthened.

Side track: "weak" and "strong" alone sounding like a valuation to the uninitiated, but neither technical nor precise; and the "objects" of comparison of sets of conditions being implicit; and the ARM not defining a technical term for these adjectives unless weak ordering helps.

If these adjectives induce confusion, can they be avoided? E.g., by instead mentioning the sets of Pre- and Post-conditions of those actual/formal/overriding subprograms. I guess that super- and subset relations will permit helpfully defining an ordering to be understood (in general, if not in the ARM).

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Tue, 11 Apr 2023 14:03:27 +0200*

On 2023-04-11 07:56, G.B. wrote:

> Side track: "weak" and "strong" alone sounding like a valuation to the uninitiated [...]

The formal meaning of weaker/stronger relation on predicates P and Q:

weaker   P => Q

stronger Q => P

The formal rationale is that if you have a proof

  P1 => P2 => P3

Then weakening P1 to P1' => P1 and strengthening P3 => P3' keeps it:

  P1' => P2 => P3'

As for ARM.

Regarding dynamic checks all the above is irrelevant because dynamic checks are no contracts. Furthermore, since the proper contracts include Constraint_Error or Storage_Error raised, do you really care how are you going to bomb your program while keeping proper contracts? (:-)) Sincere advice: forget about this mess.

For static checks a proof of implication is rather straightforward since we assume that all static predicates must be decidable anyway.

Of course, with generics you might run into troubles as you would have both proper contracts, i.e. the instantiated ones, and the generic ones expressed in generic terms. Instantiated contracts are easy to check, but what one would actually wish is checking generic contracts, which might turn out to be impossible. The glimpse of the problem is what any Ada programmer knows: generic instantiations may fail to compile even if the actual parameters match...

*From: Spiros Bousbouras <spibou@gmail.com>*
*Date: Wed, 12 Apr 2023 02:18:45 -0000*

On Tue, 11 Apr 2023 14:03:27 +0200

"Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> wrote:

>    P1' => P2 => P3'

You have it backwards ; if P1 implies P then P1 is stronger than P1 .

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Wed, 12 Apr 2023 09:49:35 +0300*

Speaking of logic in general, rather than Ada contracts in particular, I would say that you got it right, and Dmitry did not.

Suppose we have a theorem about geometrical figures F, and at first we can prove the theorem only if we assume (precondition) that the figure F is a square. Later we manage to improve the proof so that it holds also for rectangles. I would say, and I think mathematicians would say, that we /weakened/ the assumptions from "F is a square" to "F is a rectangle", and indeed the former (stronger) implies the latter (weaker), which is not as Dmitry defined "stronger".

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Thu, 13 Apr 2023 08:27:30 +0200*

On 2023-04-12 04:18, Spiros Bousbouras wrote:

> On Tue, 11 Apr 2023 14:03:27 +0200
> "Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> wrote:
>>    P1' => P2 => P3'
>
> You have it backwards ; if  P1'  implies  P1  then  P1'  is stronger than  P1 .

Yes, you are right. Inclusion is an inverse of implication.

A weaker predicate is true on a set that contains the set where the stronger predicate is.

## Looking for Feedback: ISO 3166-1 Country Country Code Reference in Ada

*From: A.J. <ianozia@gmail.com>*
*Subject: Looking for feedback: ISO 3166-1 country Country Code Reference in Ada*
*Date: Sat, 15 Apr 2023 11:52:08 -0700*
*Newsgroups: comp.lang.ada*

I just created a library for accessing ISO 3166-1 records in Ada compatible with Ada.Locales. Before I try to publish it to Alire, I'm hoping to get some feedback if anyone has some. It's possible that feedback will result in the function calls, naming convention, or structure being set up differently, so please let me know what you think.

https://github.com/AJ-Ianozi/iso_3166

I also posted this on the subreddit, so apologies for any redundancy for those viewing both!

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Mon, 17 Apr 2023 11:36:53 +0200*

Some initial thoughts on what you have:

It seems likely that your clients will use the alpha codes for input and display. It will be more convenient for that if the alpha codes are subtypes of String rather than distinct types.

Since you have already enumerated all 250 possible alpha codes, your predicates could look like

```
subtype Alpha_Code_2 is String (1 .. 2)
  with Dynamic_Predicate => Alpha_Code_2
                          in "AF" | "AL" | ...;
```

and similar for the 3-letter codes.

Since you have already enumerated all 250 possible numeric codes, you could use a restricted range for your numeric (sub)type, with a predicate restricting it to valid values.

These use the language to do validity checking for you.

Regarding the design of such a pkg, my initial instinct was to use enumeration types for the alpha codes, but a little investigation shows that some of the codes are Ada reserved words, so that doesn't work. So I would stick with the String subtypes and provide functions such that, given one of the values, the client can obtain the others, as well as the name. Alternatively, one could have functions to return a record such as you provide. Which is preferable depends on how such a pkg is typically used.

There are various possible implementations, with different tradeoffs.

## 2023 Stack Overflow: Ada in the Programming Options for the First Time

*From: Fabien Chouteau <fabien.chouteau@gmail.com>*
*Subject: 2023 Stack Overflow: Ada in the programming options for the first time*
*Date: Tue, 9 May 2023 02:39:56 -0700*
*Newsgroups: comp.lang.ada*

The 2023 Stack Overflow survey is live: https://stackoverflow.blog/2023/05/08/the-2023-developer-survey-is-now-live/

And for the first time Ada is listed in the options for "programming, scripting, and markup languages"!

Don't hesitate to fill the survey and show that the Ada community is alive.

## Ada 2022 at Last!

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Subject: Ada 23 at Last!*
*Date: Wed, 10 May 2023 11:45:01 +0200*
*Newsgroups: comp.lang.ada*

https://www.iso.org/standard/83621.html

*From: Nasser M. Abbasi <nma@12000.org>*
*Date: Wed, 10 May 2023 09:27:18 -0500*

Is there a site that gives summary of new features/changes/improvements in Ada 2023?

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Wed, 10 May 2023 07:32:25 -0700*

Yes, Ada now has an ARM and a LEG (language enhancement guide), see:

http://www.ada-auth.org/standards/overview22.html

## Ada Monthly Meeting Proposal

[see also “Ada Monthly Meetup 2023” in this AUJ issue, p.98 —arm]

*From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>*
*Subject: Re: Ada Monthly Meeting proposal*
*Date: Wed, 10 May 2023 17:39:45 +0200*
*Newsgroups: comp.lang.ada*

* Reboot of the Ada Monthly Meeting

Dear all. Once again, after a long pause, I want to revive the idea of a monthly meeting to discuss the latest Ada events, projects, releases or just have a chat about a topic.

I will not repeat what I said in the original message as all points still stand.

I was happy with the reception that the proposal gathered, alas it did not take place. However, I was thinking about having one at the beginning of each month. There would be a pause during summer (August for most people and potentially September) and FOSDEM.

* When do "we" start?

As I would not like to postpone it much more, I would like to kickstart it this June. So the first one would be either Saturday 3 or Sunday 4 of July.

I know this sounds a bit rushed. However, if I do not set a date for me and other people, we will just keep pushing it further and further. This first meetup would just be to test the waters and receive feedback. There would be another one in July and then summer, after which I hope to get a serious and continuous stream of meetups.

I was thinking that we could have a meetup at around 12PM UTC time. It is early but not crazy early for those in the USA and late for those in far east Asia such as Australia. Here in Europe it falls close to the meal time, which is not ideal... If a lot of people do not like this time, it can be easily moved a bit earlier or later... I WOULD LIKE TO RECEIVE SOME FEEDBACK ON THIS.

* What to expect?

I would like to keep these meetups sweet and short. I was thinking maybe 45 minutes long, maybe an hour. That would allow for a quick round of news, topics and introductions (something like what Maxim Reznik does but a lot shorter). Then 2 to 4 topics (depending on the time needed by each one). The topics would be what other and I already proposed. This would give between 20 to 10 minutes for each topic.

Once again, this is the starting proposal. Adjustments will be needed.

* What do I need?

Feedback:

- What is your opinion?
- Do you have a topic/project that you would like to show to the community?
- Do you like the chosen time?
- Do you like the week of the month?
- Is Jitsi [1] a good enough platform to do the meetings?
- Do you think that 45 min / 1 h is a good enough duration?
- Would you like to participate on Saturday 3 or Sunday 4?

[1] https://meet.jit.si/

*From: Simon Wright <simon@pushface.org>*
*Date: Wed, 10 May 2023 20:26:36 +0100*

Fernando Oleo Blanco <irvise_ml@irvise.xyz> writes:

> - What is you opinion?

Good idea.

> - Do you have a topic/project that you would like to show to the community?

Will have to think about that! Mac issues? Alire vs Mac?

> - Do you like the chosen time?

Fine by me.

> - Do you like the week of the month?

No problem

> - Is Jitsi [1] a good enough platform to do the meetings?

Will have to see!

> - Do you think that 45 min / 1 h is a good enough duration?

Certainly OK for the first meeting

> - Would you like to participate on Saturday 3 or Sunday 4?

Either could be managed! Slight preference for Saturday

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Wed, 10 May 2023 21:41:33 +0200*

On 2023-05-10 17:39, Fernando Oleo Blanco wrote:

> - Do you like the chosen time?

Any time will be inconvenient for some, but one must be chosen. 12:00 UTC is fine with me, but during January, the time on the west coast of the US is UTC -08, and on the east coast of Australia, UTC +11. 12:00 UTC corresponds to 04:00 in California and 23:00 in Sydney. 04:00 is rather painful. It might be better to choose 13:00 UTC (05:00 and 00:00).

*From: Ben Bacarisse <ben.usenet@bsb.me.uk>*
*Date: Wed, 10 May 2023 20:47:02 +0100*

Fernando Oleo Blanco <irvise_ml@irvise.xyz> writes:

> * When do "we" start?

Presumably 3rd or 4th June.

*From: frances...@gmail.com <francesc.rocher@gmail.com>*
*Date: Wed, 10 May 2023 23:55:48 -0700*

El dia dimecres, 10 de maig de 2023 a les 17:39:50 UTC+2, Fernando Oleo Blanco va escriure:

> - What is you opinion?

Great initiative!

> - Do you have a topic/project that you would like to show to the community?

Not yet, but for sure I'd like to show a couple of projects I'm working on.

> - Do you like the chosen time?

No problem.

> - Do you like the week of the month?

Good enough, easy to remember.

> - Is Jitsi [1] a good enough platform to do the meetings?

Let's see how it works.

> - Do you think that 45 min / 1 h is a good enough duration?

It could be flexible as it depends on the schedule and Q&A, so let's see.

> - Would you like to participate on Saturday 3 or Sunday 4?

Both are ok, but preferably Saturday.

Thanks Fernando for leading this proposal,

*From: amo...@unizar.es <amosteo@unizar.es>*
*Date: Thu, 11 May 2023 04:05:44 -0700*

On Wed, May 10, 2023 at 5:39 PM 'Fernando Oleo Blanco' via comp.lang.ada <comp.lang.ada@googlegroups.com> wrote:

- What is you opinion?

Great initiative!

- Do you have a topic/project that you would like to show to the community?

Not right now, happy to just meet people.

- Do you like the chosen time?

Works for me.

- Do you like the week of the month?

No opinion.

- Is Jitsi [1] a good enough platform to do the meetings?

It's worked for me in the past.

- Do you think that 45 min / 1 h is a good enough duration?

Yes, no more than that.

- Would you like to participate on Saturday 3 or Sunday 4?

For this instance, I can only on the 3rd. It should be indifferent normally.

Thanks Fer for leading.

*From: A.J. <ianozia@gmail.com>*
*Date: Mon, 15 May 2023 18:19:07 -0700*
> Feedback:

> - What is you opinion?

I'm absolutely up for this.

> - Do you have a topic/project that you would like to show to the community?

I recently released an Ada ISO Library for country and currency codes[1], I could talk about that if anyone is interested. I also use Ada with Alire on a mac, so I'm interested in listening to that discussion.

> - Do you like the chosen time?

It looks like 12pm UTC is 8am EDT. I normally get up around 6AM, so I can make this work.

> - Do you like the week of the month?

That should be fine.

> - Is Jitsi [1] a good enough platform to do the meetings?

If it works in a browser, I have no issues with it.

> - Do you think that 45 min / 1 h is a good enough duration?

This is good for the first such meeting. We can see how it goes and adjust in later meetings.

> - Would you like to participate on Saturday 3 or Sunday 4?

I prefer Saturday over Sunday, but either one works for me.

[1] https://github.com/ada-iso/ada_iso/tree/v2.0.0

*From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>*
*Date: Thu, 25 May 2023 19:35:12 +0200*

Hi all and especially A.J.

I will try to make the meeting happen. I made the announcement here [1].

If you would like to participate, save the date! If you have any ideas or proposals, they are welcome (this goes specially to you A.J., I assume you would like to present :).

There is a bit more info in the link if anybody else is interested. I will select the exact time next week.

[1] https://forum.ada-lang.io/t/ada-monthly-meeting/384/2?u=irvise

*From: Keith Thompson <keith.s.thompson+u@gmail.com>*
*Date: Thu, 25 May 2023 13:22:56 -0700*

The proposed time is 12:00 or 13:00 UTC on Sat 2023-06-03.

That's 05:00 or 06:00 in the US Pacific time zone (California et al).

I understand that scheduling meetings for an international audience is hard. I might join if it's later in the day in my time zone -- but I wouldn't have much to contribute anyway, so please don't base your decision on that. But I suspect a lot of people in the US won't join if it's that early.

## Is a Boolean Type Inherently Atomic?

*From: Rod Kay <rodakay5@gmail.com>*
*Subject: Is a Boolean type inherently atomic?*
*Date: Fri, 12 May 2023 22:17:48 +1000*
*Newsgroups: comp.lang.ada*

Surely only the least significant bit of the least significant byte is relevant and so the value cannot be garbled by one task writing and another reading at the same time ?

*From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>*
*Date: Fri, 12 May 2023 14:53:24 +0200*

Boolean types with other representations using multiple bits are possible, so your assumption doesn't hold.

*From: J-P. Rosen <rosen@adalog.fr>*
*Date: Fri, 12 May 2023 18:56:54 +0200*

Le 12/05/2023 à 14:53, Jeffrey R.Carter a écrit :

> Boolean types with other representations using multiple bits are possible, so your assumption doesn't hold.

True, especially considering the special exception for boolean types in 13.4(8)

Anyway, if you intend to access a variable from multiple tasks, it doesn't cost much to add an Atomic aspect to the declaration, at least to inform the reader!

*From: Niklas Holsti <niklas.holsti@tidorum.invalid>*
*Date: Fri, 12 May 2023 20:38:29 +0300*

On 2023-05-12 15:17, Rod Kay wrote:

> Surely only the least significant bit of the least significant byte is relevant and so the value cannot be garbled by one

> task writing and another reading at the same time ?

That seems very likely indeed, unless (as others have commented) the representation has been specified to use more bits. However, the Ada RM states in C.6(8/3) that "every atomic type or object is also defined to be volatile", and of course Boolean variables are not considered volatile unless they are specified to be Atomic or Volatile. So a Boolean type is not inherently atomic in the Ada RM sense of "atomic".

And of course if you use a shared variable to communicate data between tasks, that variable should be marked as Volatile, and there should also be some Atomic accesses to ensure that actions are "sequential", so marking the variable as Atomic is best.

*From: Adamagica <christ-usch.grein@t-online.de>*
*Date: Fri, 12 May 2023 11:02:15 -0700*

AARM 3.5.3(1.a), 13.4(8.b, 10/5) has some information about boolean representations.

## Ada in Jest

### Doggerel

*From: Rod Kay <rodakay5@gmail.com>*
*Subject: Doggerel*
*Date: Sat, 3 Jun 2023 12:33:30 +1000*
*Newsgroups: comp.lang.ada*

I've been holding off posting this for fear of rotten tomatoes … but here goes ...

"Tis no uncertain adage,

That that balmy beggar Babbage,

Was to antsy Aunty Ada,

No uncertain ennui saviour!"

... just putting on my hazmat suit now, so fire away :).

# 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. In general the situation seems to improve further, and only a few events are still planned to be held "virtually" or in "hybrid" mode. 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).

## 2023

| | |
|---|---|
| July 02-06 | 23rd **International Conference on embedded computer Systems: Architectures, MOdeling and Simulation** (SAMOS'2023), Samos Island, Greece. Topics include: advances in systems efficiency in various domains; novel architectures and computing methodologies and solutions for accelerating applications in various embedded domains, such as next generation automotive and avionics, next generation (machine) learning systems for surveillance and recognition, ...; software tools, compilation techniques and optimizations, and code generation for reconfigurable architectures; embedded parallel systems and MultiProcessor Systems-on-Chip; application-level resource management of multi-core architectures; all design processes for embedded systems ranging from design languages, modeling and simulation, performance, reliability, ...; specification languages and models; system-level design, simulation, and verification; MP-SoC programming, compilers, simulation and mapping technologies; profiling, measurement and analysis techniques; (design for) system adaptivity; testing and debugging; etc. |
| July 11-14 | 35th **Euromicro Conference on Real-Time Systems** (ECRTS'2023), Vienna, Austria. |
| ☺ July 17-21 | 37th **European Conference on Object-Oriented Programming** (ECOOP'2023), Seattle, USA. 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. |
| July 18-21 | **Software Technologies: Applications and Foundations** (STAF'2023), Leicester, UK. Topics include: practical and foundational advances in software technology. |
| | July 18-19 — 17th **International Conference on Tests And Proofs** (TAP'2023). Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the connection between proofs (and other static techniques) and testing (and other dynamic techniques); verification and analysis techniques combining proofs and tests; program proving with the aid of testing techniques; formal techniques supporting the automated generation of test vectors and oracles, and supporting novel definitions of coverage criteria; specification inference by deductive and dynamic methods; testing and runtime analysis of formal specifications; verification of verification tools and environments; applications of test and proof techniques in new domains; combined approaches of test and proof in the context of formal certifications; case studies, tool and framework descriptions, and experience reports about combining tests and proofs; etc. |
| July 18-21 | 19th **European Conference on Modelling Foundations and Applications** (ECMFA'2023), Leicester, UK. Co-located with STAF'2023. Topics include: all aspects of model-based engineering (MBE); foundations of MBE, including model transformations, domain-specific languages, verification and validation approaches, ...; application of MBE methods, tools, and techniques to specific domains, e.g., automotive, aerospace, cyber-physical systems, robotics, Artificial Intelligence or IoT; educational aspects of MBE; tools and initiatives for the successful adoption of MBE in industry; etc. |

☺ Aug 28 – Sep 09 29th **International European Conference on Parallel and Distributed Computing** (Euro-Par'2023), Limassol, Cyprus. Topics include: all aspects of parallel and distributed processing, ranging from theory to practice, from small to the largest parallel and distributed systems and infrastructures, from fundamental computational problems to applications, from architecture, compiler, language and interface design and implementation, to tools, support infrastructures, and application performance aspects.

September 06-08 49th **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2023), Durres, Albania. Topics include: information technology for software-intensive systems; tracks on Cyber-Physical Systems (CPS), Emerging Computing Technologies (ECT), Model-Driven Engineering and Modeling Languages (MDEML), Software Engineering and Debt Metaphors (SEaDeM), Software Process and Product Improvement (SPPI), etc.

September 11-13 2nd **Summer School on Security Testing and Verification 2023**. Brussels, Belgium. Topics include: static and dynamic security testing; software verification; security by design; etc. Deadline for early registration: July 31, 2023.

September 11-13 16th **International Conference on the Quality of Information and Communications Technology** (QUATIC'2023), Aveiro, Portugal. Topics include: all quality aspects in ICT systems engineering and management.

September 11-15 38th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2023), Kirchberg, Luxembourg. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems. Deadline for submissions: July 14 - August 4, 2023 (workshop papers), Jul 15, 2023 (industry challenge competition). Deadline for early registration: August 17, 2023.

September 17-22 **Embedded Systems Week 2023** (ESWEEK'2023), Hamburg, Germany. Includes CASES'2023 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2023 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2023 (International Conference on Embedded Software). Deadline for submissions: July 10, 2023 (student travel grants, undergraduate scholar program), July 14, 2023 (PhD forum). Deadline for early registration: August 25, 2023.

☺ Sep 17-22 ACM SIGBED **International Conference on Embedded Software** (EMSOFT'2023). Topics include: the science, engineering, and technology of embedded software development; research in the design and analysis of software that interacts with physical processes; results on cyber-physical systems, which integrate computation, networking, and physical dynamics; embedded distributed, networked systems (time-critical embedded systems, scheduling, resource allocation, and execution time analysis; ...); embedded software design and analysis (safety/mixed-critical embedded software, software design for cyber-physical systems, ...); resilience (embedded software security, robust implementation of control systems); process, methods (formal modeling and verification; testing, validation, and certification; model- and component-based approaches); empirical studies and their reproduction; application areas including automotive, avionics, energy, health care, mobile devices, multimedia, machine learning, and autonomous systems; etc.

Sep 17-22 **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems** (CASES'2023). Topics include: latest advances in design, optimization, validation, and applications of embedded systems, Internet of Things (IoT), and the emergent trend of integrating Artificial Intelligence into IoT (AIoT); architecture, design, and compiler techniques for reliability, and aging; modeling, analysis, and optimization for timing and predictability; validation, verification, testing, and debugging of embedded software; etc.

Sep 17-22 **International Conference on Hardware/Software Codesign and System Synthesis** (CODES+ISSS'2023). Topics include: system-level design, hardware/software co-design, modeling, analysis, and implementation of modern Embedded Systems, Cyber-Physical Systems, and Internet-of-Things, from system-level specification and optimization to synthesis of system-on-chip hardware/software implementations.

September 18-23 34th **International Conference on Concurrency Theory** (CONCUR'2023), Antwerp, Belgium. Co-located with FORMATS, FMICS and QEST as part of CONFEST 2023. Topics include: semantics, logics, verification and analysis of concurrent systems; basic models of concurrency; verification and analysis techniques for concurrent systems such as abstract interpretation, model checking, race detection, run-time verification, static analysis, testing, theorem proving, type systems, security analysis; distributed algorithms and data structures; theoretical foundations of architectures, execution environments, and software development for concurrent systems such as multiprocessor and multi-core architectures, compilers and tools for concurrent programming, programming models such as component-based, object-oriented, ...; etc.

September 19-21 21st **International Conference on Formal Modeling and Analysis of Timed Systems** (FORMATS'2023), Antwerp, Belgium. Co-located with CONCUR, FMICS and QEST as part of CONFEST 2023. Topics include: fundamental and practical aspects of timed systems; modelling, design and analysis of timed computational systems; theoretical foundations of timed systems, languages and models; techniques, algorithms, data structures, and software tools for analyzing timed systems and resolving temporal constraints, such as scheduling, worst-case execution time analysis, optimization, model checking, testing, constraint solving; adaptation and specialization of timing technology in application domains in which timing plays an important role (real-time software, scheduling in manufacturing and telecommunication, robotics, ...); etc.

September 19-22 42nd **International Conference on Computer Safety, Reliability and Security** (SafeComp'2023), Toulouse, France. Topics include: development, assessment, operation and maintenance of safety-related and safety-critical computer systems; safety/security risk assessment; model-based analysis, design, and assessment; formal methods for verification, validation, and fault tolerance; validation and verification methodologies and tools; methods for qualification, assurance and certification; compositional verification and certification; cyber-physical threats and vulnerability analysis; safety guidelines, standards and certification; safety and security interactions and tradeoffs; etc. Domains of application include: railways, automotive, space, avionics, nuclear and process industries; autonomous systems, advanced robotics; telecommunication and networks; critical infrastructures; medical devices and healthcare; defense, emergency & rescue; logistics, industrial automation, off-shore technology; etc. Deadline for submissions: July 10, 2023 (position papers). Deadline for early registration: July 20, 2023.

☺ September 20-22 28th **International Conference on Formal Methods for Industrial Critical Systems** (FMICS'2023), Antwerp, Belgium. Co-located with CONCUR, FORMATS and QEST as part of CONFEST 2023. 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, tool qualification, and certification); impact of adoption of formal methods on development process and associated costs; application of formal methods in standardisation and industrial forums.

September 20-22 22nd **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SOMET'2023), Naples, Italy. Topics include: new directions in software development methodologies and related tools and techniques; software methodologies and tools for robust, reliable, non-fragile software design; software development techniques for legacy systems; software evolution techniques; agile software and lean methods; software optimization and formal methods for software design; software maintenance; software security tools and techniques; formal techniques for software representation, software testing and validation; object-oriented, aspect-oriented, component-based and generic programming, multi-agent technology; model driven development (DVD), code centric to model centric software engineering; etc.

October 02-04 25th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2023), Jersey City, New Jersey, USA. Topics include: design and development of distributed systems with a focus on systems that are able to provide guarantees on their structure, performance, and/or security in the face of an adverse operational environment; distributed and concurrent computing (foundations, fault-tolerance and scalability); distributed, concurrent, and fault-tolerant algorithms; synchronization protocols; formal methods, validation, verification, and synthesis; secure software and secure programming methodologies; formal methods, semantics and verification of secure systems; fault tolerance, reliability, availability of distributed secure systems; etc.

October 03-06 23rd **International Conference on Runtime Verification** (RV'2023), Thessaloniki, Greece. Topics include: monitoring and analysis of runtime behaviour of software and hardware systems; program instrumentation; logging, recording, and replay; combination of static and dynamic analysis; monitoring techniques for concurrent and distributed systems; fault localization, containment, resilience, recovery and repair; etc. Deadline for early registration: August 15, 2023.

☺ October 10-12 — 5th **International Conference on Reliability, Safety and Security of Railway Systems** (RSSRail'2023), Berlin, Germany. Topics include: safety in development processes and safety management; combined approaches to safety and security; system and software safety analysis; formal modelling and verification techniques; system reliability; validation according to the standards; tool and model integration, tool chain; domain-specific languages and modelling frameworks; model reuse for reliability, safety and security; etc. Deadline for submissions: July 14, 2023 (posters). Deadline for early registration: July 31, 2023.

☺ October 17 — **High Integrity Software Conference** (HISC'2023), Bristol, UK. Topics include: advanced software development for high-integrity and high-assurance systems, including programming languages, AI-assisted software development, verifiable code generation; verification of novel, high-integrity and high-assurance systems; assurance of high-integrity, high-assurance systems; infrastructure & ecosystem for high-integrity software.

October 18-20 (h) — 16th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS'2023), Marrakech, Morocco. Topics include: analysis of computer and communication systems, where functional and extra-functional properties are inter-related; cross-fertilization between various formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems.

October 19-20 (v) — 19th **International Conference on Formal Aspects of Component Software** (FACS'2023), Internet. Topics include: applications of formal methods in all aspects of software components and services; formal methods, models, and languages for software-intensive systems, components and services: formal aspects of concrete software-intensive systems, including real-time/safety-critical systems, hybrid and cyber physical systems, components that use artificial intelligence, ...; tools supporting formal methods for components and services; case studies and experience reports over the above topics; special track on formal methods at large; etc. Deadline for submissions: July 3, 2023 (abstracts), July 10, 2023 (papers).

☺ October 21-25 — 32nd **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2023), Vienna, Austria. Topics include: parallel architectures; compilers and tools for parallel computer systems; applications and experimental systems studies of parallel processing; computational models for concurrent execution; support for correctness in hardware and software; reconfigurable parallel computing; parallel programming languages, algorithms, and applications; middleware and run time system support for parallel computing; etc. Deadline for submissions: July 3, 2023 (workshops), August 4, 2023 (artifacts), August 14, 2023 (tutorials).

October 22-24 — 30th **Static Analysis Symposium** (SAS'2023), Cascais (Lisbon), Portugal. Co-located with SPLASH'2023. Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.

October 22-26 — 23nd IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2023), Chiang Mai, Thailand. Topics include: reliability, security, availability, and safety of software systems; software testing, verification, and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; operating system security and reliability; benchmark, tools, industrial applications, and empirical studies; etc. Deadline for submissions: July 15, 2023 (abstracts), July 22, 2023 (regular and short papers), August 15, 2023 (workshop papers, fast abstracts, industry track, posters).

☺ October 22-27 — ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2023), Lisbon, Portugal. Topics include: all aspects of software construction and delivery, at the intersection of programming languages and software engineering. Deadline for submissions: July 7, 2023 (GPCE), July 12, 2023 (workshop papers), July 21, 2023 (Student Research Competition), July 24, 2023 (PLMW), July 27, 2023 (SPLASH-E), August 15, 2023 (posters).

> October 22-27 — 16th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2023). Topics include: software language engineering rather than engineering a specific software language; software language design and implementation; software language validation (verification and formal methods for languages, testing techniques for languages, simulation techniques for languages); software language integration and

composition; software language maintenance (software language reuse, language evolution, language families and variability, language and software product lines); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc. Deadline for submissions: August 30, 2023 (artifacts).

☺ October 23-27 **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2023). Topics include: all practical and theoretical investigations of programming languages, systems and environments, targeting any stage of software development, including requirements, modeling, prototyping, design, implementation, generation, analysis, verification, testing, evaluation, maintenance, and reuse of software systems; development of new tools, techniques, principles, and evaluations.

October 23 12th **Workshop on Programming Languages and Operating Systems** (PLOS'2023), Koblenz, Germany. Topics include: domain-specific and type-safe languages for the OS; the design of language-specific unikernels; language-based approaches to crosscutting system concerns, such as security and run-time performance; PL support for system verification, testing, and debugging; the use of OS abstractions and techniques in language runtimes; verification and static analysis of OS components; critical evaluations of new programming language ideas in support of OS construction; experience reports on applying new language techniques in commercial OS settings; etc. Deadline for paper submissions: August 4, 2023.

October 24-27 (h) 28th IEEE **Pacific Rim International Symposium on Dependable Computing** (PRDC'2023), Singapore. Topics include: software and hardware reliability, resilience, safety, security, testing, verification, and validation; dependability measurement, modeling, evaluation, and tools; architecture and system design for dependability; reliability analysis of complex systems; dependability issues in computing systems (e.g. high performance computing, real-time systems, cyber-physical systems, ...); emerging technologies (autonomous systems including autonomous vehicles, human machine teaming, smart devices/internet of things); etc.

October 24-27 21st **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2023), Singapore. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware and software systems; program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent systems; analysis and verification of deep learning systems; verification in industrial practice; applications and case studies; etc.

November 08-10 21st **International Conference on Software Engineering and Formal Methods** (SEFM'2023), Eindhoven, the Netherlands. Topics include: software development methods (formal modelling, specification, and design; software evolution, maintenance, re-engineering, and reuse), design principles (programming languages; abstraction and refinement; ...), software testing, validation, and verification, security and safety (security, privacy, and trust; safety-critical, fault-tolerant, and secure systems; software certification), applications and technology transfer (real-time, hybrid, and cyber-physical systems; intelligent systems and machine learning; education; ...), case studies, best practices, and experience reports.

November 13-15 18th **International Conference on integrated Formal Methods** (iFM'2023), Leiden, the Netherlands. 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. Deadline for submissions: July 13, 2023 (PhD symposium).

November 13-17 (h) 18th **International Conference on Software Engineering Advances** (ICSEA'2023), Valencia, Spain. Topics include: trends and achievements; advances in fundamentals for software development; advanced mechanisms for software development; advanced design tools for developing software; software performance; software security, privacy, safeness; advances in software testing; specialized software advanced applications; open source software; agile and Lean approaches in software engineering; software deployment and maintenance; software engineering techniques, metrics, and formalisms; software economics, adoption, and education; etc. Deadline for submissions: August 10, 2023.

| | |
|---|---|
| December 04-07 | 30th **Asia-Pacific Software Engineering Conference** (APSEC'2023), Seoul, South Korea. Topics include: requirements and design (component-based software engineering; software architecture, modeling, and design; middleware, frameworks, and APIs; software product-line engineering; ...); testing and analysis (testing, verification, and validation; program analysis; program repairs; ...); formal aspects of software engineering (formal methods, model-driven and domain-specific engineering); software comprehension and traceability; dependability, safety, and reliability; software maintenance and evolution (refactoring, reverse engineering, software reuse, debugging and fault localization, ...); software repository mining; etc. Deadline for submissions: July 7, 2023 (papers), August 25, 2023 (Software Engineering Education track, Early Research Achievement track, Doctoral symposium, tutorials), August 30, 2023 (Software Engineering in Practice track), September 1, 2023 (student research competition). |
| December 05-08 | 43rd IEEE **Real-Time Systems Symposium** (RTSS'2023), Taipei, Taiwan. Deadline for submissions: September 6, 2023 (Brief Presentations track). |
| December 10 | Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day! |

## 2024

| | |
|---|---|
| January 17-19 | 19th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2024), Munich, Germany. Topics include: computer architecture, programming models, compilers and operating systems for general-purpose, embedded and cyber-physical systems. Areas include safety-critical dependencies, cybersecurity, energy efficiency and machine learning. Deadline for submissions: July 3, 2023 (workshops). |
| March 02-06 | IEEE/ACM **International Symposium on Code Generation and Optimization** (CGO'2024), Edinburgh, UK. Deadlines for paper submissions: September 1, 2023 (2nd round). |
| ♦ June 11-14 | 28th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC'2024), Barcelona, Spain. Sponsored by Ada-Europe. #AEiC2024 #AdaEurope #AdaProgramming |
| December 10 | Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day! |

# 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) 11-14 June 2024, Barcelona, Spain

**Conference Chair**
**Sara Royuela**
*sara.royuela@bsc.es*
*Barcelona Supercomputing Center, Spain*

**Journal-track Chairs**
**Bjorn Andersson**
*baandersson@sei.cmu.edu*
*Carnegie Mellon University, USA*

**Luis Miguel Pinho**
*lmp@isep.ipp.pt*
*ISEP & INESC TEC, Portugal*

**Industrial-track Chairs**
**Luciana Provenzano**
*luciana.provenzano@mdu.se*
*Mälardalen University, Sweden*

**Michael Pressler**
*Michael.Pressler@de.bosch.com*
*Bosch, Germany*

**Work-In-Progress-track Chairs**
**Alejandro R. Mosteo**
*amosteo@unizar.es*
*CUD Zaragoza, Spain*

**Ruben Martins**
*rubenm@andrew.cmu.edu*
*Carnegie Mellon University, USA*

**Tutorial Chair**
**Maria A. Serrano**
*maria.serrano@nearbycomputing.com*
*NearbyComputing, Spain*

**Workshop Chair**
**Sergio Saez**
*ssaez@disca.upv.es*
*Universitat Politècnica de València, Spain*

**Exhibition & Sponsorship Chair**
**Ahlan Marriott**
*ahlan@Ada-Switzerland.ch*
*White Elephant GmbH, Switzerland*

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

**Webmaster**
**Hai Nam Tran**
*hai-nam.tran@univ-brest.fr*
*University of Brest, France*

**Local Chair**
**Nuria Sirvent**
*nuria.sirvent@bsc.es*
*Barcelona Supercomputing Center, Spain*

## General Information

The **28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024)** will take place in Barcelona, Spain. The conference schedule comprises a journal track, an industrial track, a work-in-progress track, a vendor exhibition, parallel tutorials, and satellite workshops.

- Journal-track papers present research advances supported by solid theoretical foundation and thorough evaluation.
- Industrial-track contributions highlight the practitioners' side of a challenging case study or industrial project.
- Work-in-progress-track illustrates novel research ideas that are still at an initial stage, between conception and first prototype.
- Tutorials guide attenders through a hands-on familiarization with innovative developments or with useful features related to critical software.
- Workshops provide discussion forums on themes related to the conference topics.

## Schedule

| | |
|---|---|
| 15 January 2024 | Deadline for submission of journal-track papers |
| 26 February 2024 | Deadline for submission of industrial-track papers, work-in-progress papers, tutorial and workshop proposals |
| 22 March 2024 | First round notification for journal-track papers, and notification of acceptance for all other types of submissions |
| 11-14 June 2024 | Conference |

## Scope and Topics

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

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

- Formal and model-based engineering of critical systems
- High-Integrity Systems and Reliability
- AI for High-Integrity Systems Engineering
- Real-Time Systems
- Ada Language
- Applications in relevant domains

More specific topics are described on the conference web page, at http://www.ada-europe.org/conference2024.

# *Join Ada-Europe!*

Become a member of Ada-Europe and **support Ada-related activities** and the future **development of the Ada programming language**.

Membership benefits include **receiving the quarterly Ada User Journal** and a substantial **discount when registering for the annual Ada-Europe conference**.

To apply for membership, visit our web page at

http://www.ada-europe.org/**join**

# ADEPT 2022 Workshop: A Summary of Strengths and Weaknesses of the AADL Ecosystem

**Frank Singhoff[1], Jérôme Hugues[2], Hai Nam Tran[1], Gianluca Bardaro[3], Dominique Blouin[4], Marco Bozzano[5], Patrick Denzler[6], Pierre Dissaux[7], Eric Senn[8], Xiong Xu[9], Zhibin Yang[10]**

*1. University of Brest, Lab-STICC UMR CNRS 6285, Brest, France, firstname.lastname@univ-brest.fr*

*2. Software Engineering Institute, Carnegie Mellon University, USA, jhugues@andrew.cmu.edu*

*3. Politecnico di Milano, Piazza Leonardo Da Vinci 32, Milano (IT); email: {name.surname}@polimi.it*

*4. LTCI, Telecom Paris, Institut Polytechnique de Paris, Palaiseau, France, dominique.blouin@telecom-paris.fr*

*5. Fondazione Bruno Kessler, Via Sommarive 18, 38123 Trento, Italy, bozzano@fbk.eu*

*6. Institute of Computer Engineering, TU Wien, Vienna; patrick.denzler@tuwien.ac.at*

*7. Ellidiss Technologies, 24 quai de la douane, 29200 Brest, France; pierre.dissaux@ellidiss.com.*

*8. Lab-STICC, Université de Bretagne Sud, Lorient, France; eric.senn@univ-ubs.fr*

*9. SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China; University of Chinese Academy of Sciences, Beijing, China; xux@ios.ac.cn*

*10. School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China; yangzhibin168@163.com*

## Abstract

*The Architecture Analysis and Design Language (AADL) is a SAE Standard for the modeling of both the hardware and the software of embedded systems. The AADL standard is now mature and is today employed by numerous stakeholders in the domain of critical embedded real-time systems to address a large set of concerns: performances (latency, schedulability), safety, or security, ... The ADEPT workshop aims to present and report on current projects in the field of design, implementation, and verification of critical systems where AADL is a first-citizen technology. This article is a summary of the ADEPT 2022 workshop.*

*Keywords: AADL, critical embedded real-time systems, design, implementation and verification.*

## 1 Introduction

The Architecture Analysis and Design Language (AADL) is a SAE standards for the modeling of both the hardware and the software of embedded systems [1]. The AADL standard is now a mature standard for the modeling of critical embedded real-time systems. AADL is today employed by numerous stakeholders in the domain of critical embedded real-time systems to address a large set of concerns: safety [13], security [15], or performance (latency, schedulability) [14] but also code generation [12, 21]. One key strength of AADL as a language is the set of tools that provide those analysis capabilities.

The ADEPT workshop aims to present and report on current projects in the field of design, implementation and verification of critical systems where AADL is a first citizen technology. The ADEPT workshop is also an opportunity for AADL beginners to meet experienced AADL practitioners.

The ADEPT 2022 workshop was a full day workshop. A morning session was introducing new tools and was an opportunity for AADL beginners to discover the language, its tools, and its potential uses. The afternoon was dedicated to the presentation of success stories and returns of experience in the form of a discussion with the workshop attendees. The workshop was co-located with the 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022) at Ghent, Belgium.

In the sequel, we describe the 2022 workshop program in section 2. Section 3 presents a summary of the return of experience reported by the workshop participants. Then we conclude is section 4.

## 2 Workshop program

The workshop gathered 22 participants, with 8 presentations. It was organized in 4 sessions: 1) an introduction from the workshop organizers about the AADL standard, its ecosystem and the ongoing standardization activities, 2) a session introducing emerging AADL tools, 3) a session presenting success stories, case studies, and return of experience, and 4) a discussion between the workshop participants leading to a review of the current strengths and weaknesses of the AADL standards and tools.

The tool session was composed of 4 presentations. [2] addresses SysML v2 and AADL. The speakers show how SysML v2 constructs can be mapped to AADL entities, allowing verifications on SysML models by AADL tools such as AADLInspector. The 3 next presentations were examples of how formal methods can be combined with AADL [3, 4, 5]. [3] introduces MARS, a graphical modeling and verification tool. MARS inputs are models combining AADL and Simulink/Stateflow components. Both AADL and Simulink components are translated to hybrid communicating sequential processes for formal verification purposes. [4] describes C2AADL_Reverse, which is both an approach and a tool for model-driven reverse engineering. The proposal is to extract design artefacts from a multi-tasked C code in order to automatically produce AADL models. The produced model can then be used for formal verification or code generation. Finally, presentation [5] was dealing with the COMPASTA project in which the COMPASS [11] and the TASTE [10] tools are combined. Both are AADL oriented tools: TASTE is a set of tools bridged by AADL models while COMPASS provides formal safety analysis on models expressed with a dialect of AADL.

The case study session hosted 4 presentations [6, 7, 8, 9]. Two presentations were addressing robotic systems modeling and verification [7, 8]. They are both focussing on the ROS platform and show how AADL is able to model ROS nodes, network and device entities. In [7], AADL modeling of ROS entities targets early performance verification to predict required network bandwidth, end to end latencies or processor utilization. [8] proposes to generate a part of the C/C++ code for ROS architecture from similar AADL models. Network modelling is also a concern in the talk [6] where the AADL is used to model a gateway architecture in the context of fog computing. One of the motivations of this AADL model is to generate complex gateway configurations. Finally, the last talk illustrates the use of AADL in the context of a railways system with ALISA (Architecture-Led Incremental System Assurance) [9]. In this context, ALISA is used to both express and verify system requirements of a train control system.

## 3 Strengths and weaknesses of the AADL ecosystem

During the workshop, a session dedicated to a discussion about return of experience from workshop participants has been organized. Return of experience collected during this session is summarised in the sequel.

3 topics were discussed: topics related to the AADL language, its tools and its community.

About the AADL language, the main addressed topics covered the rationale to choose AADL. The participants also identified the language features that are missing today in AADL and the ones they wish for in the next versions of the standards.

Availability of tools is one of the motivations to select a design or a programming language. AADL tools used by the workshop participants have been enumerated during the workshop. Participants also reviewed the main issues they faced when using them. Such review is an opportunity to understand the future features workshop participants are waiting for in the next tool versions.

Finally, model based technologies such as AADL technologies strongly change how practitioners design and implement software. Having an active user community contributes to increasing stakeholders AADL skills and then contributes to successfully applying AADL on concrete projects. What AADL users expect from the user community and how this community would be organized were also topics discussed during the workshop.

In the sequel, we only describe the items reported by workshop participants that we believe are more important.

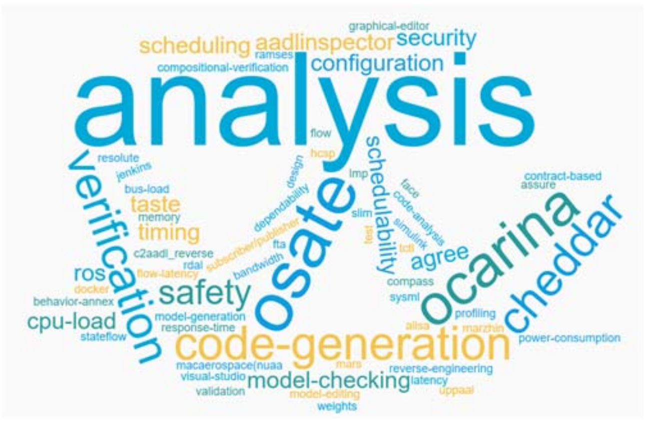


**Figure 1: User's intends and used tools**

### 3.1 Rationale to choose AADL

Workshop participants explained why they have chosen AADL for the work they presented. Sometimes, AADL was simply chosen because it was part of a larger toolset (e.g. TASTE) they are using. Sometimes also, as with any technologies, AADL was chosen simply because participants knew it before.

About the language itself, AADL was frequently selected because it has the ability to both model hardware and software components, with both textual and graphical syntaxes, and with features that bring software engineering good practices (e.g. inheritance, abstract components, modularity, …).

AADL's ability to model both static and dynamic aspects of embedded systems architecture and to predict its behavior is also an important rationale for most of the workshop participants. Participants aim to assess properties on the architecture behavior or to do design space exploration by investigating various architecture alternatives (e.g.

deployment options), to identify, for example, performance bottlenecks or safety/security vulnerabilities.

Last but least, the availability of AADL mature tools was an argument for several workshop participants. The list of AADL tools workshop participants are using is displayed in the word cloud of Figure 1. This figure also contains words representing the concerns of the workshop participants when using AADL. Not surprisingly, and it is easy to see it in Figure 1, participants expect analysis features on various aspects while simultaneously keeping mandatory features such as modeling and code generation services/tools. Finally, many tools exist for AADL, but one aspect which was highlighted by workshop participants is the ability of AADL to build heterogeneous tool-chains (e.g., TASTE, OSATE, AADLInspector).

### 3.2 AADL standard review

In the sequel, we enumerate several issues in the AADL standards which were pointed out by workshop participants. One of the workshop objectives was to identify the features that are currently missing in the AADL standards.

First, we must notice that several participants underlined the (too) large number of features already defined in the AADL standards. One of the expressed needs by workshop participants is to instead provide a better formal specification of the current AADL features and to define and support well-defined subsets of AADL standards. Some concepts are also expected to be simplified (e.g. flow specification is seen as cumbersome). Furthermore, workshop participants also missed proper documentation of standard constructs and options, to reduce the learning curve of AADL for beginners. We see there that improving the current AADL standard but with the same feature bounding box would be a first user requirement.

There are however few missing features that were identified by workshop participants. One of them is about safety and behavioral specification (e.g. specification of timed/hybrid behavior in state machines, bridge between behavioral and safety annexe). Workshop participants also remind that the standard proposes nothing today to model the physical environment of the system to implement.

Finally, AADL has extensions mechanisms that can be solutions for some of the requests discussed during the workshop, but the spirit of those mechanisms have probably to be more explained to practitioners. For example, workshop participants complain about the lack for the modelling of multi-core systems, machine learning components, ROS components, specific aspects of virtual bus or resource binding, while the current AADL standard provides extension mechanisms to allow users to define their own domain or application specific properties and property sets. Assessing that those extensions are able to cope with the requirements expressed by workshop participants stays an open question.

### 3.3 AADL tools issues and expected features

During the workshop, participants have also made a return of experience on the AADL tools they are using. One of the main reported issues is the level of compliance to the standard. Most of the tools cover only a part of the standard and as it is difficult to extend them by users, such limits restrict tool applicability. Tool interoperability would be probably better experimented by tool developers.

An important identified lack is about the relationships between AADL declarative models and instance models. There is a need of bi-directional transformation/flow between AADL declarative and instance models. Most of the tools only provide transformations from declarative models to instance models, but few provide the reverse after analysis for example. Second, an interest for transformation from or to a graphical representation is also pointed out.

Of course, AADL tools, as any software, are subject to bugs, deprecated features or release, poor documentation, unmaintained tools, and it was also pointed out by workshop participants.

### 3.4 AADL community

The last topic discussed during the workshop dealt with the AADL community and what users expect from it.

First, several participants highlighted that it is difficult to reach other AADL users. There is a demand for Internet forums animated to share experience, material, problems and solutions. Workshop participants expressed the needs of workshops, open source teaching materials, case studies, or working examples, in fact any materials that contribute to reducing learning time for AADL newcomers.

Few repositories and websites providing a part of such AADL material exist however: model repositories and examples with OpenAADL and AADLib [16], AADL cook books [17], AADL labs or teaching materials [18, 19, 20], … Those repositories and websites probably have to be better advertised but also updated to cope with user’s expectations.

Workshop participants also raised the interest to continue the AADL users' days that were taking place during the AADL standardization committee meetings. It shows that there is also a need for meetings devoted to users where open source tools/materials can be presented, demonstrated. An annual workshop as ADEPT could play that role.

## 4 Conclusion

AADL is a set of SAE international standards that aims to improve the quality of the critical embedded systems design. The AADL standards are now mature. The objective of the ADEPT workshop was to encourage discussion between members of the AADL community. ADEPT may be a location to share experiences on AADL and its ecosystem. In this article, we summarized the discussions during the first edition of the ADEPT workshop co-located with the 26th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2022) at Ghent, Belgium.

Obviously, we cannot state as representative the return of experience presented in this article as it is the opinion of 22 workshop participants only. However, the workshop shows that AADL is an active standard currently used in various domains in Europe. The workshop pointed out also that the community lacks events to share experiences on the standards and its tools.

## Acknowledgments

We would like to thank Ellidiss Technologies who supported the registration fees for the speakers of the workshop.

## References

[1] *Architecture Analysis and Design Language (AADL) AS-5506c:* Tech. Rep. The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report (2017), Version 2.2.

[2] Jean-Charles Roger, Pierre Dissaux. *AADL modelling with SysML v2.* Ada User journal, volume 44, number 1, March 2023, pages 63-66. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[3] Xiong Xu, Shuling Wang, Bohua Zhan, Xiangyu Jin, Naijun Zhan, Jean-Pierre Talpin. *Unified graphical co-modeling, analysis and verification of cyber-physical systems by combining AADL and Simulink/Stateflow.* Ada User journal, volume 44, number 1, March 2023, pages 67-70. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[4] Zhibin Yang, Zhikai Qiu, Yong Zhou, Zhiqiu Huang, Jean-Paul Bodeveix, Mamoun Filali. *C2AADL_Reverse: A model-driven reverse engineering approach for development and verification of safety-critical software.* Ada User journal, volume 44, number 1, March 2023, pages 71-74. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[5] Alberto Bombardelli, Marco Bozzano, Roberto Cavada, Alessandro Cimatti, Alberto Griggio, Massimo Nazaria, Edoardo Nicolodi, Stefano Tonetta, Gianni Zampedri. *COMPASTA: Integrating COMPASS Functionality into TASTE.* Ada User journal, volume 44, number 1, March 2023, pages 75-78. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[6] Patrick Denzler, Daniel Ramsauer, Daniel Scheuchenstuhl, Wolfgang Kastner. *Experiences Modeling a OPC UA / DDS Gateway in AADL in the Context of Fog Computing.* Ada User journal, volume 44, number 1, March 2023, pages 79-79. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[7] Gianluca Bardaro, Matteo Matteucci. *Modelling robot architectures with AADL.* Ada User journal, volume 44, number 1, March 2023, pages 80-83. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[8] Eric Senn, Lucie Bourdon. *Modeling ROS based applications with AADL.* Ada User journal, volume 44, number 1, March 2023, pages 84-87. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[9] Dominique Blouin, Paolo Crisafulli, Françoise Caron, Cristian Maxime. *An Introduction to ALISA and an Experience Report on its Usage for an Industrial Railway System Case Study.* Ada User journal, volume 44, number 1, March 2023, pages 88-91. Also presented in the ADEPT 2022 workshop, Ghent, Belgium, 2022.

[10] Perrotin, Maxime, et al. *TASTE: An open-source tool-chain for embedded system and software development.* Embedded Real Time Software and Systems (ERTS2012). 2012, France, Toulouse.

[11] Bozzano, M., Cavada, R., Cimatti, A., Katoen, J. P., Nguyen, V., Noll, T., & Olive, X. *Formal verification and validation of AADL models.* In ERTS2 2010, Embedded Real Time Software & Systems 2010, May, France, Toulouse.

[12] Hugues, J., Zalila, B., Pautet, L., & Kordon, F. *From the prototype to the final embedded system using the Ocarina AADL tool suite.* ACM Transactions on Embedded Computing Systems (TECS), 7(4), 1-25, 2008.

[13] Delange, J., & Feiler, P. *Architecture fault modeling with the AADL error-model annex.* In 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (pp. 361-368). IEEE, 2014.

[14] Singhoff, F., Legrand, J., Nana, L., & Marcé. *Scheduling and memory requirements analysis with AADL.* ACM SIGAda Ada Letters, 25(4), 1-10, 2005.

[15] Ellison, R., Hudak, J., Kazman, R., Woody, C., & Householder, A. *Extending aadl for security design assurance of cyber physical systems.* CMU/SEI, USA, 2015.

[16] AADLib repositories, http://www.openaadl.org/

[17] AADL cook book, http://beru.univ-brest.fr/AACB/

[18] Cyber-Physical Systems Design & Analysis course teaching, by Georgia Institute of Technology, https://www.udacity.com/course/cyber-physical-systems-design-analysis--ud9876

[19] CPS-IoT Summer School 2022, https://mem4csd.telecom-paristech.fr/blog/index.php/training-schools/cps-iot-summer-school-2022/

[20] SEA UE Teaching week 2022, http://beru.univ-brest.fr/cheddar/contribs/educational/ubo/SEA-UE_TEACHING_WEEK_2022/LAB/tp.html

[21] F. Cadoret, E. Borde, S. Gardoll and L. Pautet, "Design Patterns for Rule-Based Refinement of Safety Critical Embedded Systems Models," 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems, Paris, France, 2012, pp. 67-76, doi: 10.1109/ICECCS20050.2012.6299202.

# Software-based Security Approach for Networked Embedded Devices

*José Ferreira, Alan Oliveira, André Souto, José Cecílio*
*LASIGE, Departamento de Informática, Faculdade de Ciências da Universidade Lisboa; email: fc53311@alunos.ciencias.ulisboa.pt,{aodsa,ansouto,jmcecilio}@ciencias.ulisboa.pt*

## Abstract

*As the Internet of Things (IoT) continues to expand, data security has become increasingly important for ensuring privacy and safety, especially given the sensitive and, sometimes, critical nature of the data handled by IoT devices. There exist hardware-based trusted execution environments used to protect data, but they are not compatible with low-cost devices that lack hardware-assisted security features. The research in this paper presents software-based protection and encryption mechanisms explicitly designed for embedded devices. The proposed architecture consists of two parts: the Agent, which is designed to work with low-cost, low-end devices without requiring modifications to the underlying hardware, and the Computing Module, which is designed for slightly more computationally powerful devices. The Computing Module enables devices to write data in protected memory and continuously verifies its integrity to provide protection. Additionally, it utilizes the Agents located on the device to safeguard device applications against attacks by requesting the Agent to generate an application code signature and validating it. The proposed solution is an alternative data security approach for low-cost IoT devices without compromising performance or functionality. Our work underscores the importance of developing secure and cost-effective solutions for protecting data in the context of IoT.*

*Keywords: IoT Security, Trusted Execution Environment, Code Protection, Memory Integrity.*

## 1 Introduction

As a result of the Internet of Things (IoT) popularization, millions of embedded devices are being deployed and connected to the worldwide network [1]. These devices allow IoT to extend the boundaries of the Internet. At the same time, they connect digital processes to the physical world [2]. Although the resulting systems create added-value services for the respective applications, they inherently also bring vulnerabilities [3]. The threats must be mitigated since these systems might collect and process critical and sensitive data.

Among the techniques proposed to protect data are the Trusted Execution Environments (TEEs). TEEs are designed to provide mechanisms to protect applications (code and critical data), ensuring confidentiality and integrity [3]. However, most existing TEEs proposals rely on hardware [3], such as Trusted Platform Modules (TPMs) [4], Intel SGX [5] and ARM TrustZone [6]. Due to cost and size constraints, those hardware features mainly exist on high-end platforms and are unavailable on cost-effective and low-end embedded devices [3].

Alternatively, software-based approaches are being proposed, such as PISTIS [3], Security MicroVisor ($S\mu V$) [2], SofTEE [7], and Virtual Ghost [8]. These solutions have some advantages when compared with hardware-assisted ones. One advantage is related to update costs. It is easier and cheaper to update a software-based TEE. Another advantage is hardware portability, given that those TEEs do not require specific hardware features (such as ARM TrustZone or Intel SGX) [7]. Despite these advantages, most of those software-based secure architectures do not consider hardware-based attacks in their scheme, i.e., they do not consider scenarios in which the attacker has access to the device where the application is running and can change its code.

In the literature, it is possible to find solutions hybrid architectures, i.e., architectures that combine hardware and software modules to achieve the best of both solutions [9, 10, 11, 12]. Although those solutions offer strong security guarantees for applications running on low-end devices, to be implemented in those devices, they require hardware modification [3]. Hardware modification is impracticable in real-world scenarios, considering that every device needs the addition of customized hardware [3], and it is difficult for legacy devices to take advantage of them [7]. Moreover, the hybrid solution approaches usually consider that hardware-based attacks are out of scope.

Considering the lack of hardware security features in low-cost embedded systems and the benefits offered by software-based TEEs, this work aims to develop a software-based security approach for networked embedded devices (SbS4NED) that provides a set of lightweight mechanisms to protect software and data integrity (continuously verifying the integrity of memory) and offer correction in case of unexpected changes. The application code will also be protected using encryption. This way, it becomes tamper resistant and offers more reliability to the verification process. Moreover, it will be supported by lightweight cryptography algorithms presented at the National Institute of Standards and Technology (NIST) competition [13]. In particular, Xoodyak [14], one of NIST's finalists, will be used to encrypt data.

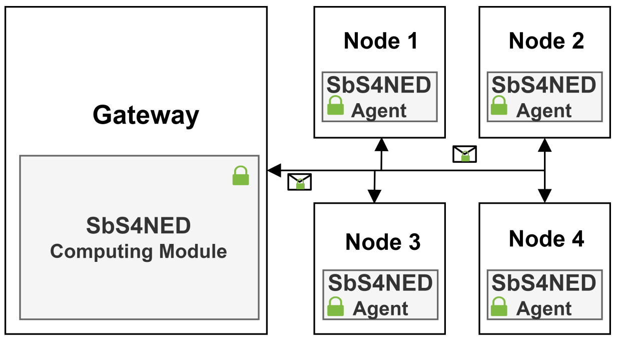


**Figure 1: SbS4NED architecture. SbS4NED Computing modules (CM) on the Gateway and SbS4NED Agents on each node connected to the Gateway.**

## 2 Thread Model and Assumptions

Defining the assumptions about the attacker's capabilities and goals, the system's components and interactions, and the security goals that need to be achieved are essential. In this work, we consider adversaries with the following capabilities:

- The adversaries have access to the device. They may modify the application code running on the device to read or change the data the application handles.
- The adversaries can sniff the network, modify messages exchanged between devices, and perform man-in-the-middle attacks.
- Software-based adversaries may be present on the device where the architecture will be deployed. Their goal may be to change the data available in memory and consequently control the entities that rely on data accuracy.

We assume that the SbS4NED is correctly installed on the devices by a trusted party. We also assume it is bug-free, encrypted, and working as expected. Therefore, the adversary can not surpass the code, and the verification process carried out by its components. The final assumption is that each device has mechanisms to compute the encryption key used to protect the local files where SbS4NED keys are stored.

## 3 Software-based security Architecture

As mentioned in the Introduction, this work aims to build a software-based tamper-resistant solution that protects the software and data in networked embedded devices. Driven by this goal, we design the SbS4NED proposed architecture.

Figure 1 shows a high-level description of the proposed architecture, where the SbS4NED Computing Module (SbS4NED_CM) runs inside the Gateway. It is responsible for monitoring applications running on the nodes connected to the Gateway. Each node will have an agent (SbS4NED_Agent) generating the signature of the application code running on the node and sending it to the SbS4NED_CM for code integrity check. The SbS4NED_CM and the node code's application are encrypted to increase security and to offer more protection to SbS4NED_CM internal processes. Moreover, the messages exchanged between SbS4NED_CM and its agent are also encrypted. Next, we describe the SbS4NED components:

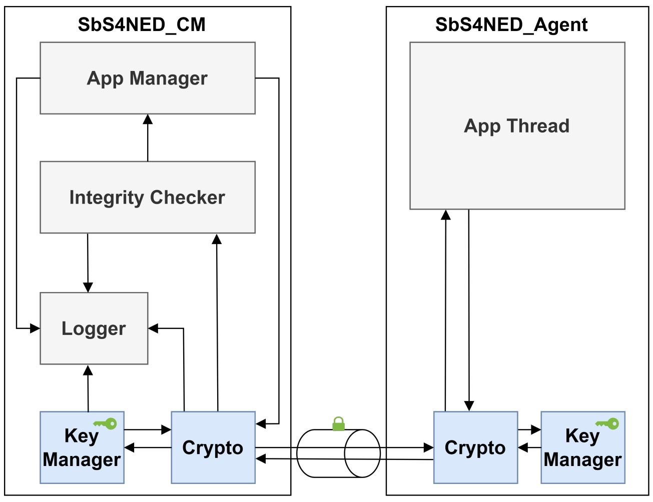


**Figure 2: SbS4NED Modules. CM-specific and Agent-specific modules, along with Common modules (Key Manager and Crypto)**

- `App Manager` – It interacts with the applications deployed in the node and aims to perform application updates and send and receive data from the nodes.
- `Key Manager` – This component is responsible for managing (*i.e.*, generating and renewing) the keys used internally by SbS4NED_CM and for external communication (with a SbS4NED_Agent running on the node). It uses Diffie-Hellman (DH) key-exchange protocol for external communication to generate or renew the key.
- `Crypto` – Provides cryptography services inside the SbS4NED_CM and the SbS4NED_Agent. It can encrypt, decrypt, and compute the message authentication code (MAC). In the SbS4NED_CM side, the `App Manager` can also use this component to encrypt the compiled app code before sending it to the node. This way, secure code update is ensured.
- `Integrity Checker` – Designed for memory integrity checking. It writes the data from App Manager in the memory and holds the (randomized) position where it is written. The Integrity Checker is also responsible for remotely checking the integrity of the nodes' code.
- `Logger` – It is responsible for keeping the log files updated regarding the memory integrity state, which app the data came from, which nodes are connected, and any network activity that must be logged to easily detect if an attacker is trying to join the network or injecting any data on the network.
- `App Thread` – It is used for executing the application code developed by the user. It offers an API to interact with the node's underlying software and hardware layers. All the interaction must be done using the API to ensure, for instance, that the exchanged data is encrypted.

The system architecture of SbS4NED_CM and its agents is illustrated in Figure 2. The figure provides an overview of the individual components and their interconnections within the system.

The Key Manager and Crypto components are the essential modules used in the SbS4NED_CM and also in its agents. These components are deployed on both sides of the system, providing the necessary encryption services and ensuring that data transmitted between the agents and SbS4NED_CM remain secure and confidential.

The App Manager, Integrity Checker, and Logger are part of SbS4NED_CM. These must be deployed in the Gateway. The App Manager plays a crucial role in dealing with the application code deployed in the agents, providing the necessary services to manage, update, and configure them. On the other hand, the Integrity Checker ensures that the code and messages transmitted by the agent are authentic and have not been tampered with.

## 3.1 Data Protection

Memory integrity is nowadays a crucial security concern. The integrity of the data stored in memory is essential to ensure the system's proper functioning and to prevent unauthorized access or manipulation of sensitive information. When data is written in the memory, two pieces of information are stored: the value ($v$) that can be accessed by any other external entity, such as an actuator, and the data integrity ($I$) needed to check the integrity of the data. Data integrity $I$ is computed using the MAC, and its purpose is to ensure that the data in memory remains untampered and unmodified. The formula to compute it is $I = \text{MAC}(v \oplus t)$, where $t$ can be a timestamp or a random number known only to the integrity checker. The position of the value $I$ in memory is arbitrary, and only the Integrity Checker knows where it is placed. The choice of MAC over hash functions is because MAC uses a secret key to generate the authentication code. Assuming that SbS4NED_CM has exclusive access to this private key, only authorized accesses to SbS4NED_CM can generate the correct MAC result. In contrast, anyone can compute the hash value by identifying the function used. A secret key prevents anyone from computing the hash value and faking the integrity data. This way, SbS4NED_CM provides a strong level of security against malicious attacks to the memory and an easy and efficient verification of the memory's integrity.

## 3.2 Application Protection

Besides confidentiality protection provided by encryption, the application code is locally stored in the node and decrypted only during execution. Application code also has integrity protection to ensure it remains unmodified even when other parties can access the node. To check the code's integrity, a challenge-response protocol is used, in which the SbS4NED_CM will send $enc(t)$, an encrypted challenge to the node, where $t$ can be a timestamp or a value randomly generated by the SbS4NED_CM. The node has to reply with the hash of the whole application code ($ac$) XOR-ed with $t$, i.e., $enc(hash(ac \oplus t))$. Then, the SbS4NED_CM checks the validity of the result. If the node fails the validation, SbS4NED_CM could force an update to restore the node application. If the node gives no response, SbS4NED_CM assumes that the node is lost or compromised and its data is dropped.

## 3.3 Keys Renewal

A renewal cycle mechanism can extend the system's lifetime. Therefore, before any application update, the key used to encrypt the application code and for communication between the SbS4NED_CM and node is renewed using a DH protocol. However, the application could take a long time without needing an update. Therefore, the node is provided with an encryption application code, and the SbS4NED_CM can initialize the DH key exchange with the node to generate the new key even when there is no call of the functionally. The generated key will be used to renew the encrypt of the application code and in further communication with SbS4NED_CM.

## 3.4 Encryption Algorithms

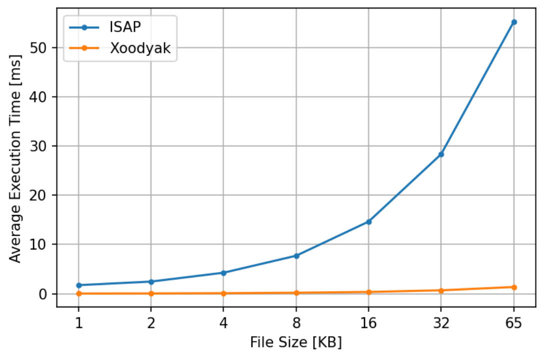
Since the proposed architecture uses an encryption algorithm and targets low-end devices, the algorithms that can be used must be lightweight, including the cryptographic ones. Thus, this architecture will use NIST lightweight encryption algorithms to protect code and data during message exchange [13]. Although by the time of writing this manuscript, the final stage of the NIST competition comprises ten finalists, we are only interested in algorithms that can deal with stream encryption. The main reason is that we want the encrypted code and messages to have the same size as the original ones. Among the ten finalists, few support stream encryption natively. For the SbS4NED architecture, Xoodyak and ISAP schemes with keyed mode association are considered. Since the SbS4NED architecture is modular, other algorithms may also be used.

## 4 Proof of concept

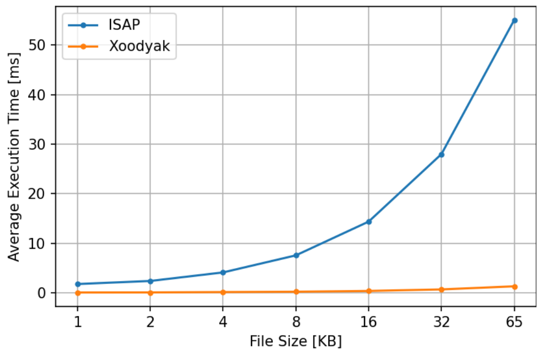
To verify and characterize the proposed architecture, we are currently implementing it. We plan to deploy the architecture in a prototype, enabling system testing in a distributed environment. The prototype will consist of a Gateway on which the SbS4NED_CM will be running, with connections to multiple nodes where the SbS4NED agents and applications will be deployed. By conducting tests in a distributed environment, we can ensure that the architecture can effectively handle the communication and data exchange requirements between the Gateway and the nodes. Additionally, we will be able to identify potential issues or limitations during deployment, which will help us refine the architecture further.

During the early stages of designing the proposed architecture, we conducted experiments using two NIST lightweight encryption algorithms, ISAP and Xoodyak, to determine which would be more suitable for our research. Our experiment used a Raspberry Pi Model 3+ platform with a Quad-core @1.4GHz and 1GB LPDDR2 SRAM. We tested both algorithms for their execution time and memory usage using the same key length of 16 bytes, a nounce of 16 bytes, and file sizes from 1 to 65 kilobytes (kB) (Figure 3). The tests also included both the encryption (Figure3a) and decryption (Figure 3b) processes.

Our experiment shows that Xoodyak was more suitable for our research than ISAP in terms of both average execution time and memory usage. The average execution time for Xoodyak was consistently lower, especially for larger file sizes (generally, Xoodyak is 30 to 60 times faster than ISAP with a maximum standard deviation of 0.004 ms). Moreover,



**(a) [ISAP] and [Xoodyak] Encryption.**



**(b) [ISAP] and [Xoodyak] Decryption.**

**Figure 3: [ISAP] and [Xoodyak] Encryption and Decryption Average Execution Time [ms] on Raspberry Pi 3+. Based on Different File Sizes.**

both algorithms required about 370 kb of RAM to perform encryption and decryption tasks. It was noticed that file size does not affect the algorithm's memory usage. In summary, our experiments with ISAP and Xoodyak algorithms, conducted on a Raspberry Pi Model 3+ platform, helped us to determine which algorithm is more suitable for our research.

## 5 Conclusion

This work proposes a software-based secure execution environment architecture that is lightweight, requires no hardware modifications and is resistant to the most common hardware attacks. Currently, this architecture is being implemented as a proof of concept. After the implementation, tests will be conducted to analyze its performance and characterize its efficiency.

## Acknowledgments

This work was supported by the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020), and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).

## References

[1] M. Ammar and B. Crispo, "Verify&revive: Secure detection and recovery of compromised low-end embedded devices," in *Annual Computer Security Applications Conference*, (New York, NY), p. 717–732, ACM, 2020.

[2] M. Ammar, B. Crispo, B. Jacobs, D. Hughes, and W. Daniels, "S$\mu$v—the security microvisor: A formally-verified software-based security architecture for the internet of things," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 5, pp. 885–901, 2019.

[3] M. Grisafi, M. Ammar, M. Roveri, and B. Crispo, "PISTIS: Trusted computing architecture for low-end embedded systems," in *31st USENIX Security Symposium*, (Boston, MA), pp. 3843–3860, Aug. 2022.

[4] T. C. Group, "TCG specification architecture overview." `https://trustedcomputinggroup.org/wp-content/uploads/TCG\_1\_4\_Architecture\_Overview.pdf`, 2007. [Online - Accessed on 11-11-2022].

[5] V. Costan and S. Devadas, "Intel sgx explained." Cryptology ePrint Archive, Paper 2016/086, 2016. `https://eprint.iacr.org/2016/086`.

[6] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "Trustzone explained: Architectural features and use cases," in *2nd IEEE Int. Conf. on Collaboration and Internet Computing*, pp. 445–451, 2016.

[7] U. Lee and C. Park, "Softee: Software-based trusted execution environment for user applications," *IEEE Access*, vol. 8, pp. 121874–121888, 2020.

[8] J. Criswell, N. Dautenhahn, and V. Adve, "Virtual ghost: Protecting applications from hostile operating systems," *SIGARCH Comput. Archit. News*, vol. 42, p. 81–96, feb 2014.

[9] I. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified Hardware/Software Co-Design for remote attestation," in *28th USENIX Security Symposium*, (Santa Clara, CA), pp. 1429–1446, 2019.

[10] K. Eldefrawy, A. Francillon, D. Perito, and G. Tsudik, "Smart: Secure and minimal architecture for (establishing a dynamic) root of trust," in *19th Annual Network and Distributed System Security Symposium, February 5-8, San Diego, USA* (ISOC, ed.), (San Diego), 2012.

[11] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, (New York, NY, USA), ACM, 2014.

[12] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[13] NIST, "Lightweight cryptography." `https://csrc.nist.gov/projects/lightweight-cryptography`. Accessed: 2022-12-04.

[14] J. Daemen, S. Hoffert, S. Mella, M. Peeters, G. Van Assche, and R. Van Keer, "Xoodyak, a lightweight cryptographic scheme," *NIST*, 05 2021.

# Cooperative Autonomous Driving in Simulation

***Gonçalo Costa, José Cecílio, António Casimiro***
*LASIGE, Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa; email: fc53352@alunos.ciencias.ulisboa.pt,{jmcecilio, casim}@ciencias.ulisboa.pt*

## Abstract

*Autonomous driving is an area that has been growing in recent years. However, cars are unprepared to cooperate with others nearby, wasting resources and computational power. Thus, cooperative autonomous driving reveals its importance in the future. In this work-in-progress paper, we define, implement and test an architecture for a simulation environment where cooperative autonomous driving protocols can be tested. Additionally, a Manoeuvre Negotiation Protocol is implemented. This protocol will make an existing autonomous driving (AD) stack more resilient in real driving scenarios, improving its robustness and safety.*

*Keywords: Autonomous Driving stack, Simulator, Manoeuvre Negotiation Protocol, SVL, Apollo*

## 1 Introduction

With the increase in the number of vehicles present on the roads [1], it became necessary to research autonomous driving, not only to allow better traffic management on the road and thus reduce travel time but also to reduce the number of accidents and, consequently, the amount of money dispended for health recovering [2].

According to the Society of Automotive Engineers (SAE), there are five levels of automation [3]. At level 1, most components are controlled by the human driver, and only some essential functions are automated (computer controlled). In contrast, at level 5, the vehicle is fully autonomous, with all controls being automated, without any human driver intervention or dependence [2, 4, 5]. Currently, the highest level of automation used in commercialized vehicles is between level 2 and level 2.5, in which vehicles are autonomous up to a certain point, requiring human intervention in adverse or extreme situations [4].

Vehicle to Everything (V2X) is a concept that is very common in the area of autonomous driving and refers to communication between vehicles (V2V) and communication with infrastructures (V2I). The V2X system is crucial for identifying and analyzing obstacles and phenomena, which are then communicated to other vehicles and infrastructures. This way, drivers can learn information without observing it in person, which improves driving safety [6]. Subir et al. [7] reinforce using V2V and V2I communication, addressing the topic of Cooperative Collision Avoidance. This article indicates the advantages of using broadcast messages instead of performing a specific routing.

However, over these years of evolution towards autonomous driving, there has been a significant increase in the number of sensors and cameras in vehicles, increasing the number of cables and data buses crucial for passing and sharing data. Furthermore, the code required for all these components to work correctly is extensive and increasingly complex, which can put reliability at risk. Finally, the increasing technology in vehicles requires more computational power [4] since they comprise a lot of sensors that generate a considerable amount of data to be processed quickly.

This approach of putting as much technology as possible into vehicles refers to scenarios where each vehicle reacts to the surrounding environment. Despite being a viable approach for the present, we must consider that in a future where all vehicles are considered autonomous, there will be a considerable waste of computation since all the vehicles will be performing the same processing of the environment surrounding them [4].

The advantages of cooperative autonomous driving systems have already been demonstrated and discussed in [7]. However, its implementation is not addressed. One of the main difficulties is the costs associated with installing and testing those systems in actual vehicles, in which the prices of cars and infrastructure are too high [7, 8, 9, 10, 11]. Another difficulty associated with the implementation is the high number of hours that would have to be devoted to testing the algorithms on the roads, with algorithms that require hundreds of hours of testing in different conditions (e.g., atmospheric conditions). This makes the process very complicated to carry out [8, 9, 10].

Considering all difficulties, the one that makes the entire implementation process very difficult refers to the safety conditions of humans, in which many tests involve pedestrians and, in general, all tests carried out on the road can compromise the life of any pedestrian who is present in the vicinity [8, 10, 11].

According to the ISO/PAS 21448:2019 [12] standard, several safety measures were created to guarantee safe conditions while testing autonomous driving algorithms and protocols. Thus, using simulators of real environments to test algorithms and protocols became necessary since we can virtually test any condition without creating dangerous situations. The main advantage of simulators is the ability to change reality for the different tests that the algorithms need to be trained and evaluated [8, 10, 11]. The work in [10] used the CARLA simulator to train a driving policy through Reinforcement Learning to test it in the real world later. As such, they recreated the route in the simulator and trained the system according to the real-world scenario. The work done in [13] also presents a protocol for vehicle coordination and implements it using the V-REP simulator. The author then conducted some tests on the protocol, namely implementing purposeful communication failures, to verify the protocol's robustness. All these works indicate methods to test algorithms and protocols, maintaining the security of people and infrastructures.

This work will explore the advantages of cooperative driving systems, where vehicles can cooperate in the surrounding environment. We will use a simulator to implement and test a cooperative protocol for manoeuvre interception. The protocol was designed to guarantee greater safety conditions for each vehicle while reducing the construction complexity and code necessary for correct operations [14]. This work helps to remove the human factor in driving through vehicle control and decision-making, improving vehicle efficiency and safety. This work is also motivated by the need to test the protocol in different scenarios to observe its behavior. Using failure scenarios to analyze their impact on the protocol is crucial. It is also essential to consider real-world factors such as message losses, latency variations, and malicious behaviors that can influence the autonomous driving system [4, 5, 7, 8, 10, 11].

This work uses a realistic simulator (SVL [15]) to demonstrate immersively how the entire architecture works. It allows the simulation of a driving scenario, considering other vehicles, pedestrians, traffic signs, and traffic rules. Furthermore, each vehicle corresponds to an instance of the Apollo autonomous driving stack [16], along with the protocol proposed in [14]. A network simulator is also integrated to recreate real conditions for introducing failures.

## 2 Autonomous Driving Protocol

The Manoeuvre Negotiation Protocol used [14] is based on a solution where vehicles have different priorities, depending on the manoeuvre they intend to perform. However, to avoid a scenario where vehicles wait indefinitely to manoeuvre, a lower-priority vehicle can increase its priority and carry out the manoeuvre it wants to perform. Suppose the time to complete the manoeuvre is less than the time the higher-priority vehicle takes to reach the intersection. In that case, the vehicle can increase its priority and make the intersection.

The protocol was designed to handle specific conditions, like Priority Violation. It is considered that there has been a Priority Violation when a vehicle that has increased its priority cannot complete the intersection, causing vehicles with higher priorities to be unable to perform normal traffic behaviour. To prevent this, the protocol predicts the time for the vehicle to reach the intersection and pass the intersection. Suppose the time interval of the passing vehicle intersects the time interval of the vehicle present on the road of the intersection. In that case, the protocol will not allow the vehicle to cross the intersection.

Each vehicle ($p_i$) also has a membership that indicates to other vehicles, with higher priority, that it intends to perform a manoeuvre.

The protocol updates the membership every $T_M$ unit of time. It is also responsible for calculating which vehicles have higher priority than $p_i$ and which can reach the intersection during the execution of the manoeuvre. This way, the protocol is prepared for different types of intersections (ex, three-way or four-way intersections). Timestamps are used to check the correctness of the membership to guarantee security. Finally, the value of Manoeuvre Opportunity (*MO*) is determined, which only becomes *True* if all vehicles with the highest priority are within the communication range. The vehicle is in an admissible crossing opportunity if the membership is empty and fresh.

This protocol is also responsible for describing vehicles' states and messages when carrying out a manoeuvre and scenarios where messages may not reach the intended destination due to network failures. Before any vehicle wants to carry out a manoeuvre, it must connect to the server and store its information (e.g., location). Then, this information is used to create the membership. When a vehicle intends to perform a manoeuvre, it invokes the *tryManoeuvre* procedure, which generates a request tag, including the timestamp when the request was made, the requester ID, and the manoeuvre to be performed.

Next, the vehicle checks its state, and if it is in the *NORMAL* (no manoeuvre is being performed) or *TRYGET* state (agent intends to execute a manoeuvre), it invokes the Membership Algorithm (*MA*). If the vehicle can perform the grant request after invoking the *MA*, it sends it and starts a timer named *tRETRY*. Suppose the vehicle receives all responses with the *GRANT* message (sent when the agent accepts the required manoeuvre). In that case, the vehicle changes its state to *EXECUTE* and executes the manoeuvre. Otherwise, it sends a *RELEASE* message (sent when the vehicle intends to revoke the GRANT) to the agents that granted him, changes its state to *TRYGET*, and starts the *tRETRY* command for a new round, avoiding a deadlock situation. If the vehicle does not receive all the responses within the *tRETRY* time, the vehicle changes its state to *TRYGET* and executes the *tryManoeuvre* procedure again.

In a scenario where the vehicle is in the *GRANT* state (vehicle gives in to another to perform a manoeuvre) but wants to perform a manoeuvre, it switches to the *GRANTGET* state (vehicle gives a GRANT message but intends to perform a manoeuvre) and will invoke the *tryManoeuvre* procedure when it receives a *RELEASE* message.

This protocol also covers scenarios where communication can fail. If the *RELEASE* message does not reach the vehicles, they check the vehicle's last position to whom the *GRANT* message was sent and inferred, using sensors, whether it is outside the intersection or not. The protocol can also discard messages sent in previous rounds to avoid interfering with the current round.

## 3 Autonomous Driving Simulator

Regarding the simulator, it was decided to choose the SVL simulator [15] because it is open source, largely customizable, and has a realistic graphics engine that can be easily extended. In addition, the SVL allows the creation of modules that can be used to establish communication between vehicles, being

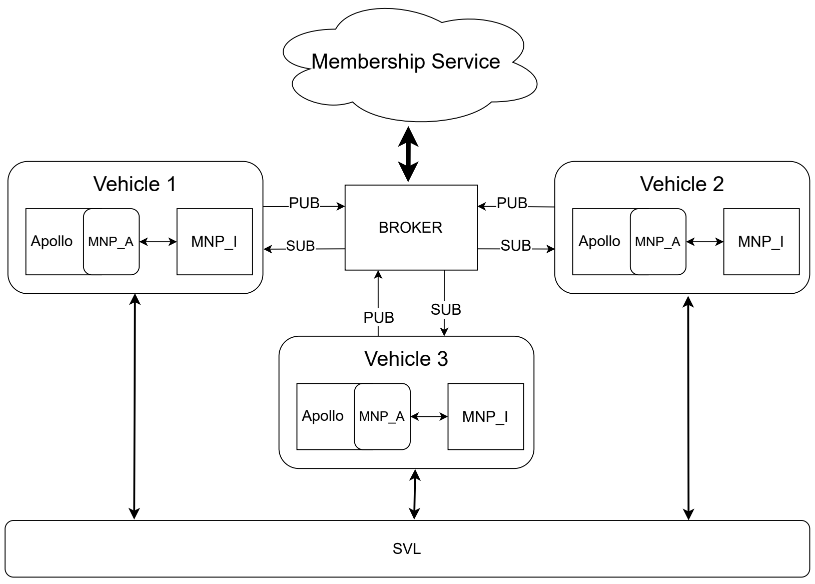


**Figure 1: Architecture of the simulator**

crucial for the work. Regarding the simulator's architecture, SVL has a section responsible for loading the map, vehicles, sensors, and environmental settings. The SVL is then responsible for sending the results obtained by the sensors to the AD Stack. After receiving inputs from the SVL, the Autonomous Driving (AD) Stack is responsible for applying the protocol and updating the vehicle status in the SVL. This update corresponds to the manoeuvre or strategic route changes by the vehicle. There is also a Visual section that receives the output of the SVL, corresponding to the vehicle circulating on the road according to the actuators provided by the AD stack.

Regarding the AD stack, the Apollo stack is used since it is easily incorporated with the SVL, which makes the implementation of the protocol much easier. Apollo runs inside a docker container, making it possible to create multiple instances of Apollo, each corresponding to an autonomous vehicle.

Figure 1 represents the system architecture implemented in this work. It comprises modules corresponding to the Manoeuvre Negotiation Protocol, the Apollo AD stack, and the network simulator used to conduct communication tests between the different vehicles.

In our architecture, each vehicle comprises an Apollo AD stack, a Manoeuvre Negotiation Protocol Agent (MNP_A) and a Manoeuvre Negotiation Protocol Instance (MNP_I). The MNP_A fetches data from the Apollo AD stack and sends it to MNP_I. Then, MNP_I updates the membership, applies the protocol, and returns the protocol commands to the Apollo AD stack through the MNP_A. All the MNP_I and the membership are connected to a broker. This broker implements pub/sub mechanisms to support the integration of multiple vehicles in a seamless way. All vehicles registered on the broker can use the protocol since the messages exchanged are broadcasted to a specific topic that everyone is subscribed to. The protocol also defines specific message types to update the membership. Access to these topics is restricted to the Membership Service.

Lastly, all the vehicles are connected to the SVL simulator to represent their status.

## 4 Implementation of the Simulator

Implementing the architecture defined in Figure 1, requires docker since the Apollo AD stack runs inside a docker container. In our implementation, there are two containers per vehicle: one that supports and runs Apollo and another container responsible for the protocol's operation. In each Apollo instance, the Planning and Control modules connect to the MNP_A module to receive information about the vehicle, such as speed, position, acceleration, and trajectories.

The connection between MNP_A and MNP_I is made by sending UDP messages. Then, MNP_I uses MQTT topics to communicate with the broker and other vehicles to create safe manoeuvres. Each MQTT message includes specific information concerning the intention of each vehicle. For instance, if a vehicle intends to start a manoeuvre, it must publish a message with the vehicle identification (vehicle ID), a timestamp, a manoeuvre code, and the trajectory.

All instances are subscribed to the same topic. The vehicles filter each message, and if it corresponds to a message sent by itself, it is discarded. If the vehicle ID is different, the message is processed by the MNP_I module, allowing it to change the values of acceleration, velocity, and trajectory of the vehicle. If necessary, it sends updated information to the MNP_A module, which will update the information inside the Apollo container by changing the information in the Control module and completing the execution flow.

Finally, the communication between Apollo and SVL simulator is done through a network bridge. This bridge refers to the local host if the entire simulation is performed on one machine. Otherwise, the bridge refers to the IPv4 address of the machines running the Apollo stack. In this way, SVL supports multiple systems connected simultaneously.

The entire architecture can be implemented in a single machine. However, it requires a machine with a considerable amount of graphics and processing power to process the sensor results in the AD stack and to render the world in which the cars are. In addition, in the implementation carried out in this work, multiple autonomous vehicles will be used. They will receive and process data, increasing the requirements for a machine to process the simulation. Thus, it is important to spread the processing across several machines when needed.

## 5 Conclusion

This work proposes an architecture to develop a simulator for testing cooperative autonomous driving protocols. The solution comprises integrating an autonomous driving stack and the SVL simulator that represents the environment and the physical conditions/status of the sensors presented in the vehicles, as well as integrating a cooperative autonomous driving protocol. In the prototype developed, two vehicles were considered, and we concluded that the simulator requires a machine with a considerable amount of graphics and processing power, which may suggest spreading the processing across several machines if more vehicles are considered. We intend to test the functioning of the protocol by evaluating possible collisions between vehicles when going through the intersection. We also plan to insert hundreds of vehicles in the same intersection to observe the protocol's behavior and latencies to reach a consensus regarding access to the intersection. We also intend to test the functioning of the architecture by adding several vehicles to verify the performance of the membership service and the latency in the generation of memberships. In addition, we intend to compare the behavior of vehicles in a simulation without using the architecture and with the use of the architecture to verify the improvements associated with the safety and organization of the vehicles. Finally, tests that measure the consumption of machine resources when the architecture is running are essential to verify its efficiency.

## Acknowledgments

This work was supported by the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020), and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).

## References

[1] J. Dargay and D. Gately, "Income's effect on car and vehicle ownership, worldwide: 1960–2015," *Transportation Research Part A: Policy and Practice*, vol. 33, no. 2, pp. 101–138, 1999.

[2] S. Mariani, G. Cabri, and F. Zambonelli, "Coordination of autonomous vehicles: taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 1, pp. 1–33, 2021.

[3] SAE, "Taxonomy and Definitions for Terms Related to Cooperative Driving Automation for On-Road Motor Vehicles." `https://www.sae.org/standards/content/j3216_202107`, 2023. [Online; accessed 16-May-2023].

[4] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1243–1274, 2018.

[5] I. Yaqoob, L. U. Khan, S. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2019.

[6] A. Abacus, "Vehicle-to-everything (V2X) communication – the design engineer's guide." `https://www.avnet.com/wps/portal/abacus/solutions/markets/automotive-and-transportation/automotive/communications-and-connectivity/v2x-communication/`, 2023. [Online; accessed 7-February-2023].

[7] S. Biswas, R. Tatchikou, and F. Dion, "Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety," *IEEE communications magazine*, vol. 44, no. 1, pp. 74–82, 2006.

[8] J. Seymour, Q.-H. Luu, *et al.*, "An empirical testing of autonomous vehicle simulator system for urban driving," in *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*, pp. 111–117, IEEE, 2021.

[9] D. Zhao, Y. Liu, C. Zhang, and Y. Li, "Autonomous driving simulation for unmanned vehicles," in *2015 IEEE Winter Conference on Applications of Computer Vision*, pp. 185–190, IEEE, 2015.

[10] B. Osinski, A. Jakubowski, P. Zięcina, P. Miłoś, C. Galias, S. Homoceanu, and H. Michalewski, "Simulation-based reinforcement learning for real-world autonomous driving," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6411–6418, IEEE, 2020.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*, pp. 1–16, PMLR, 2017.

[12] ISO, "ISO/PAS 21448:2019(en) Road vehicles — Safety of the intended functionality." `https://img.auto-testing.net/testingimg/202003/19/071723321.pdf`, 2023. [Online; accessed 7-February-2023].

[13] J. P. V. Pinto *et al.*, *Design and implementation of a protocol for safe cooperation of self-driving cars.* PhD thesis, 2019.

[14] A. Casimiro, E. Ekenstedt, and E. M. Schiller, "Membership-based manoeuvre negotiation in autonomous and safety-critical vehicular systems," *arXiv preprint arXiv:1906.04703*, 2019.

[15] SVL, "Introduction." `https://www.svlsimulator.com/docs/getting-started/introduction/`, 2022. [Online; accessed 9-December-2022].

[16] Apollo, "Apollo Platform." `https://developer.apollo.auto/developer.html`, 2022. [Online; accessed 12-December-2022].

# Achieving Crash Fault Tolerance in Autonomous Vehicle Autopilot Software Stacks Through Safety-Critical Module Rejuvenation

***Federico Lucchetti, Marcus Voelp***

*Critical and Extreme Security and Dependability Group (CritiX), Interdisciplinary Centre for Security Reliability and Trust, University of Luxembourg, Luxembourg; email: {federico.lucchetti,marcus.voelp}@uni.lu*

## Abstract

*Autonomous driving vehicles (ADV), have been in recent years, victims of their own success. Through their use of increasingly sophisticated sensor modalities and deep learning capabilities, ADVs have not only learned how to probe their chaotic environment with higher granularity coupled with smooth trajectory execution but also inherited all the vulnerabilities that were hiding behind these new features. Ensuring the safety of ADVs is crucial since a simple fault along their underlying autopilot software stack can lead to catastrophic accidents with the loss of human lives. Therefore we propose a crash-fault tolerant scheme that can be triggered whenever a crash fault of the safety critical submodules of the autopilot software stack is detected, which executes an emergency trajectory and effectively steers the car into a safe spot where the autopilot can be rejuvenated. We implement and evaluate the efficacy of this recovery scheme in the Apollo ADV software stack in conjunction with the SVL simulator. Keywords: autonomous driving vehicles, rejuvenation, crash fault tolerance, simplex architecture, apollo software stack.*

## 1 Introduction

Autonomous driving vehicles (ADVs) have become more advanced in recent years, incorporating advanced features such as deep neural networks and intricate obstacle prediction modules. These developments are crucial to the success of computer vision algorithms, precise ADV trajectory planning, and smooth control algorithms, and have raised the level of driving automation to new heights. However, this increasing sophistication also comes with a downside. The growing complexity of ADVs increases their vulnerability to malicious intrusions and introduces new potential faults, which can lead to dangerous and even deadly outcomes.

Another limitation is that the autonomous driving software stack is typically regarded as reliable and expected to withstand accidental failures and cyber-attacks, which lately have become increasingly more advances. This assumption places an additional constraint on the development of autonomous driving technology, as it must be designed to be highly robust and resistant to potential security breaches.

Indeed, ADVs have been involved in numerous tragic incidents over the past two decades. One major contributing factor to these accidents has been unintended accelerations (UA), which have resulted in the deaths of 89 individuals [1]. Such behaviors can have two possible origins: they may result from an accidental internal fault or the absence of a fail-safe mechanism, or they may be intentionally provoked by a malicious attacker [2].

The adoption of ADVs on a large scale depends on convincing human drivers of their reliability. Therefore, the resilience of ADVs must play a critical role in ensuring their effective adoption by the general public. It is inevitable that faults will occur at any level, so it is crucial to equip ADVs with mechanisms that enable them to tolerate these faults. In the event of faults, a responsibility gap arises, in which it is unclear who can be held responsible for any unintended catastrophic outcomes. This gray zone is even wider due to the over-reliance of modern ADV modules on artificial neural networks (NN) whose safety and reliability properties are seldom studied in conjunction with the entire ADV software stack [3].

In addition to their black-box nature, NNs are susceptible to common faults that can originate from either software or hardware issues [4]. In the former, malicious intruders can reprogram, evade, or data-poison NNs during either the inference or training phase [5, 6, 7]. In the latter case NNs can for instance be targets of single-event upsets leading to permanent or transitory faults such as stuck-at or bit-flip types which can alter the parameter space of the NN or cause an erroneous computation of hidden layers' activation functions [8].

Similarly, sensors are not immune to attacks. Malicious actors can modify the lane-keeping system by installing dirty road patches, causing the ADV to drift out of its lane [9]. Jamming the camera modules or executing LIDAR spoofing attacks in order to inject false obstacle depth can lead to false sensor data and cause the ADV's data processing chain to compute erroneous control commands [10, 11]. In these cases, the health of the sensors remains uncompromised, and traditional fault detection schemes are unable to detect the attacks.

Autonomous driving software stacks, such as Apollo Baidu, typically consist of a series of interconnected modules that process information sequentially in a event-triggered manner. Sensor values are gathered and processed in the prediction module, then forwarded to the prediction module for obstacle trajectory prediction. A planning module computes the safest and shortest trajectory given the constraints forwarded by the prediction module. The ADV trajectory coordinates are translates into control commands by the control module and sent to the electronic control unit (ECU) for actuation. Due to the downstream interdependence of these modules and the causal interlinking of the computation and safe execution of control commands, the failure of an intermediary module can propagate throughout the information processing chain and result in unforeseen behaviors.

To mitigate the risk of single points of failure in ADV software, efforts have been made to employ redundancy by gathering data from multiple sources, such as RGB cameras, LIDAR, and RADAR, and fusing it to minimize the impact of a faulty device [12]. However, redundancy comes at an additional computational cost, and certain modules, such as GPU-resource greedy NNs, cannot be easily replicated. Other have developed adaptive control algorithms and equipped sensor fusion modules with fault detection capabilities to enhance the overall fault tolerance of ADVs [13]. However, validating ADV software in a real physical environment is costly and not scalable for all possible driving scenarios. Therefore, interfacing physics simulators like SVL [14] with ADV software stacks is crucial to ensure quality assurance in the automotive sector as required by the evolving standard ISO 21448: Safety of the Intended functionality [15].

### 1.1 Related Work

The studies referenced in [12] and [13] investigated fault tolerance in ADV systems. The former focused on fusing data from different sensor modalities to mask potentially faulty outputs, while the latter developed a model adaptive control algorithm with fault detection capabilities to enhance overall fault tolerance. Abad et al. [16] studied the safety conditions for recovering software-faulty modules in cyber-physical systems, while Abdi et al. [17] proposed a system-wide restart method that leveraged system inertia to prevent destabilization. The efficacy of redundancy was studied and applied to individual neurons in trained NNs to increase fault tolerance [18], and Khunasaraphan et al. [19] developed a technique for quickly restoring weights and recovering the entire NN after fault detection.

In this work we plan to design a resilience scheme that recovers the full safety critical features of the ADV software stack through rejuvenation after having triggered upon fault detection an emergency trajectory execution mechanisms which steers the ADV into a safe spot.

The recovery method proposed in this paper belongs to the category of shallow recovery methods, which aim to repair faulty components of a CPS with minimal or no operation on the system states. For instance, Abad et al. developed a technique that restarts a failed component and replaces it with a healthy one [16], while Shin et al. suggest leveraging redundancy to fuse the output of multiple replicas and isolating and restarting the origin of the faulty contribution upon attack detection [20].

## 2 Emergency Recovery Scheme

### 2.1 System Model

We refer to a standard ADV architecture and which can be delineated as a succession of cascading modules:

- *Perception system* comprising sensors (cameras, LIDAR, localization) and computer vision tools to first fuse different sensor modalities and then detect, classify surrounding obstacles.
- *Prediction module* predict through the use of trained neural networks, the trajectories of previously detected obstacles.
- *Planning task* outputs a spatio-temporal ADV trajectory latest prediction output and a selected destination point.
- *Control task* computes the required steering, braking and throttling commands in order to execute the received ADV trajectory in a stable fashion.
- *Electronic control unit (ECU)* actuates the computed control commands.

### 2.2 Fault Model

In analyzing the ADV architecture shown in the top half of Figure 2, it becomes clear that each module within the system represents a single point of failure. This means that any fault occurring in one module can result in either an incorrect computation by subsequent modules or a delay and/or absence in the transmission of information, ultimately leading to the disruption of the ECU's ability to generate accurate and timely control commands. GPU greedy algorithms such as obstacle classification and prediction that power the perception and prediction modules, make this part of the software stack not only safety-critical to the safe maneuver execution of the ADV but are also highly vulnerable to potential faults. We consider in this work only software based crash fault i.e. non-responsiveness in the perception and/or prediction modules or time delayed attacks [21] performed on any of the task which all together can lead to missing or delayed information forwarded to the subsequent modules. We assume that this type of fault can be detected with high coverage by setting a hard deadline on the periodic execution time of the control task.

### 2.3 Emergency Maneuver

In order to tolerate the type of crash fault laid out in out fault model without the need to investigate its source, we propose to instantiate a parallel and more lightweight software stack following a simplex architecture [22]. The latter is devoid of the fault-prone perception-prediction module complex and is composed of a simpler planning module and a replica of the original control module. We refer to the lightweight planning module as the simplex-planning module whose task is to compute at every moment a potential trajectory to the nearest safe spot e.g. an emergency lane on a highway or a parking spot (see Figure 1).

A switch reads at every processing cycle the output of both the original and simplex stack, and by default forwards the commands of the original module to the ECU. If a crash of the original stack is detected via a missing or delayed control

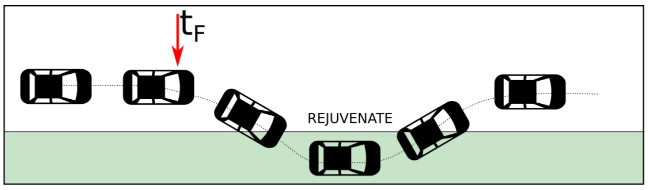


**Figure 1: Emergency parking. A module crash is detected at time $t_F$ which triggers the simplex-planning module to compute a trajectory into the safe spot area (green) where the whole autopilot software stack can be rejuvenated.**

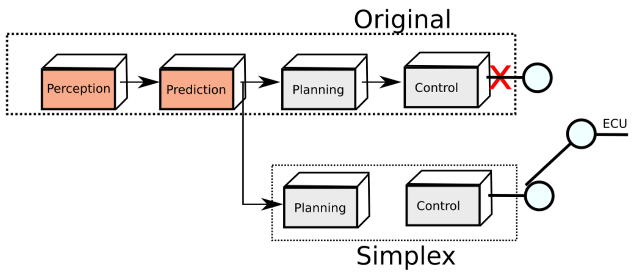
command, the switch forwards the control commands of the simplex stack to the ECU which steers the ADV into a safe spot. Once the car is at a given safe spot, a reboot signal is sent to the original software stack and consequently triggering a rejuvenation of the latter. This is schematically represented on Figure 2. While freshly rebooting a new stack during which all the software components have been re-instantiated from a trusted source (e.g. original NN weights are loaded into GPU memory and control algorithms are rejuvenated with new internal states) we exclude intermittent software faults to be resolved by this procedure.

To ensure the safe execution of this type of emergency maneuver, it is necessary to assume that the ADV is always near a safe lane, such as on a highway, and therefore cannot be performed in highly congested areas like city centers. With this assumption, it becomes feasible to compute an emergency trajectory at each processing cycle. However, in order to guarantee the dependability of switching from the original to the simplex stack devoid of perception system, the most recent computed trajectory has a short expiration time which should be significantly longer than the crash-fault detection time of the original stack. The specific duration of this expiration time depends on the speed of the ADV and the distance between the current and safe spot position. This will be investigated in the evaluation section of this manuscript.

We make the standard assumption that both stack, original and simplex are isolated and diverse enough, for example through obfuscation, to ensure that they fail independently with high coverage. Implementation-wise, this will be achieved via containerization (see next section). Our design employs a hybrid architecture, in which we differentiate the fault model of our trusted components in the simplex stack. While the components of the original stack are susceptible to crash failures, simplex-planning and the replicated control module must not fail. We justify this requirement based on their simplicity, as both components have considerably less number of lines of code than those in the original stack. Specifically, we assume that techniques such as ECC and scrubbing are in place to correct the effects of accidental faults in the stored data.

## 3 Implementation and Evaluation

The scheme proposed in the system model above will be implemented in the Apollo ADV software stack and simulated using the SVL physics simulator.



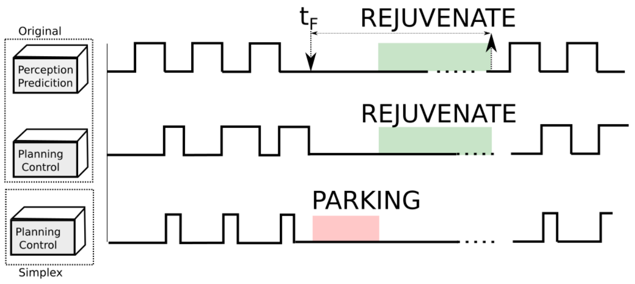


**Figure 2: Top: Switching between the original and the simplex autopilot software stack once a crash (red boxes) in the original stack cause a delayed response from the original control module to the ECU. Bottom: Temporal logic of the recovery scheme presented above in which a fault is detected at time $t_F$ and the parking scenario by the simplex stack is executed by the ECU. Green area denotes the time where the original stack is rejuvenated after which the normal driving conditions are restored.**

Apollo utilizes containers to provide isolation and protection for its components. Containers provide a restricted execution environment with communication capabilities between containers and are hosted on top of a Linux-based operating system within Apollo. In a deployed system, we assume that critical component containers will be directly hosted on top of a real-time operating system (RTOS) capable of providing the necessary isolation. However, the RTOS in these architectures represents a single point of failure that will need to be addressed in the future (as demonstrated in the Midir architecture [23]). For demonstration purposes, we will implement both the original and simplex architecture of the software stack in two isolated containers, in which the inter-process communication will be assured via a simple TCP socket. Moreover, we plan to run the switching mechanisms between the two stacks in a third container for further separation.

In order to demonstrate our approach, we plan to design a set of diverse driving scenarios in SVL comprising a simple cruise control along a highway and a more complex situation inside a congested city area with multiple pedestrians and other vehicles. We will evaluate the efficacy qualitatively i.e. the ability of the ADV to avoid sudden crashes with surrounding obstacles. We will simulate a crash fault of the original stack by stopping the processing of any of the original stack submodules.

As we have laid out in out fault model section, the dependability of the emergency maneuver can only be guaranteed if the crash fault detection delay is significantly shorter than the expiry time of the planned emergency trajectory. By simulating a whole range of scenarios, we will study the dependency of the crash-fault detection delay and the ADV speed at the moment of the crash on the efficacy of the emergency maneuver.

## 4 Acknowledgments

This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH project).

## References

[1] "Toyota "unintended acceleration" has killed 89," May 2010.

[2] A. Lima, F. Rocha, M. Völp, and P. Esteves-Veríssimo, "Towards safe and secure autonomous and cooperative vehicle ecosystems," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pp. 59–70, 2016.

[3] Z. Peng, J. Yang, T.-H. P. Chen, and L. Ma, "A first look at the integration of machine learning models in complex autonomous driving systems," 2020.

[4] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[5] G. F. Elsayed, I. Goodfellow, and J. Sohl-Dickstein, "Adversarial reprogramming of neural networks," *arXiv preprint arXiv:1806.11146*, 2018.

[6] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1625–1634, 2018.

[7] H. Aghakhani, D. Meng, Y.-X. Wang, C. Kruegel, and G. Vigna, "Bullseye polytope: A scalable clean-label poisoning attack with improved transferability," in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 159–178, IEEE, 2021.

[8] B. S. Arad and A. El-Amawy, "On fault tolerant training of feedforward neural networks," *Neural Networks*, vol. 10, no. 3, pp. 539–553, 1997.

[9] T. Sato, J. Shen, N. Wang, Y. Jia, X. Lin, and Q. A. Chen, "Dirty road can attack: Security of deep learning based automated lane centering under {Physical-World} attack," in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 3309–3326, 2021.

[10] M. Panoff, R. G. Dutta, Y. Hu, K. Yang, and Y. Jin, "On sensor security in the era of iot and cps," *SN Computer Science*, vol. 2, no. 1, pp. 1–14, 2021.

[11] C. Zhou, Q. Yan, Y. Shi, and L. Sun, "Doublestar: Long-range attack towards depth estimation based obstacle avoidance in autonomous systems," *arXiv preprint arXiv:2110.03154*, 2021.

[12] M. Darms, P. Rybski, and C. Urmson, "Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 1197–1202, IEEE, 2008.

[13] K. Geng and S. Liu, "Robust path tracking control for autonomous vehicle based on a novel fault tolerant adaptive model predictive control algorithm," *Applied Sciences*, vol. 10, no. 18, p. 6249, 2020.

[14] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta, E. Agafonov, T. H. Kim, E. Sterner, K. Ushiroda, M. Reyes, D. Zelenkovsky, and S. Kim, "SVL Simulator: A High Fidelity Simulator for Autonomous Driving," *arXiv e-prints*, p. arXiv:2005.03778, May 2020.

[15] I. Iso, "Pas 21448-road vehicles-safety of the intended functionality," *International Organization for Standardization*, 2019.

[16] F. A. T. Abad, R. Mancuso, S. Bak, O. Dantsker, and M. Caccamo, "Reset-based recovery for real-time cyber-physical systems with temporal safety constraints," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2016.

[17] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 10–21, IEEE, 2018.

[18] L.-C. Chu and B. W. Wah, "Fault tolerant neural networks with hybrid redundancy," in *1990 IJCNN international joint conference on neural networks*, pp. 639–649, IEEE, 1990.

[19] C. Khunasaraphan, T. Tanprasert, and C. Lursinsap, "Recovering faulty self-organizing neural networks: By weight shifting technique," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, vol. 3, pp. 1513–1518, IEEE, 1994.

[20] J. Shin, Y. Baek, J. Lee, and S. Lee, "Cyber-physical attack detection and recovery based on rnn in automotive brake systems," *Applied Sciences*, vol. 9, no. 1, p. 82, 2018.

[21] K. Xiahou, Y. Liu, and Q. Wu, "Robust load frequency control of power systems against random time-delay attacks," *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 909–911, 2020.

[22] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 99–107, IEEE, 2009.

[23] I. P. Gouveia, M. Völp, and P. Esteves-Verissimo, "Behind the last line of defense: Surviving soc faults and intrusions," *Computers & Security*, vol. 123, p. 102920, 2022.

# Symbolic Refinement for CPS

***Dionisio de Niz, Lutz Wrage***

*Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA, USA; email: {dionisio,lwrage}@sei.cmu.edu*

## Abstract

*In this paper we present an analysis contract approach that takes advantage of efficient domain-specific analysis algorithms, enable incremental analysis of architectural model refinements, and implement assume-guarantee reasoning in symbolic domains in SMT.*

## 1 Introduction

There is an increasing understanding that assuring large Cyber-Physical Systems (CPS) must start as early as possible to avoid re-work cost that increases exponentially the later problems are discovered. Such an assurance involves a variety of verification procedures that evaluate claims both during development but especially during the final acceptance evaluation. This final evaluation involves, in most commercial (e.g., FAA and FDA) and defense systems (Operational Test and Evaluation), a certification authority that must determine if the final system is fit and safe to operate.

Early assurance requires early design evaluation and verification procedures that first must be conducted with limited information and second generate results that must be preserved across design refinements that lead all the way down to the final implementation. In this paper we present an approach that enables the symbolic integration of verification results at different levels of refinement applied to architectural models. To do this we developed contracts that enable the determination of the level of refinement and the selection of the analyses corresponding to this level. We call our approach *Symbolic Assurance Refinement* (SAR).

### 1.1 Symbolic Refinement

SAR encodes assume-guarantee contracts that wrap model analyses to create a contract argumentation tree that defines, at each level, how specific contracts discharge claims from either a root verification plan or assumptions of other contracts. This argumentation is encoded as a constraint satisfaction problem expressed in Satisfiability Modulo Theories (SMT) form, which allows us to validate the final claims of the verification plan with an SMT solver.

To handle different level of refinement, SAR allows the omission of design details, leaving them as proof obligations that allow us to (1) figure out if the analysis contracts applied to the current level of refinement lead to contradictions and if (2) at the end of the refinement, our complete model still has remaining proof obligations to discharge.

### 1.2 Related Work

Assume/guarantee reasoning is based on Hoare triples [1]. This approach evolved into more abstract domains with the development of contract algebras [2]. Contracts have also been used in Assume/Guarantee reasoning over components in AADL models [3]. However, component contracts reason about properties of the values that AADL components communicate through their ports to other components and the computation that occurs inside a component that transform the input values into output values. This is indeed a more traditional way of thinking about contracts that perhaps is easier to map to a Hoare triples. In contrast, analysis contracts reason about properties of analysis algorithms applied to (e.g., AADL) models, not to the components of the model. This is because our goal is to reason about how multiple analyses work together to prove top-level assurance claims instead of how properties on values generated by model components discharge properties of top-level components. From this point of view, an analysis that uses component contracts to verify value transformation properties is just another analysis that we integrate and that would have its own analysis contract.

Previous work in analyses contract started with [4], where contracts were defined for resource allocation models. These contracts were defined in Alloy [5] and the analyses algorithms were implemented in Mathematica and included in the AADL models. Analyses contracts were later extended [6] first to remove the bounded verification limitations of Alloy, implementing the contracts specification with a mixture of SMT and LTL [7] with a verification in Z3 and SPIN [8]. This work also extended the analyses beyond resource allocation to domains such as thermal dissipation and security. Later, the authors in [9] created an implementation of analysis contracts with a special emphasis on lower-level analysis assumptions within the same domain.

In [10], the authors present a contract model close to analysis contracts with a synthesis approach to combine multiple contracts that restrict the design space out of pre-crafted parts. Their approach works at a more abstract level closer to [2] and it is applied at the assurance case level and reuse of assurance case patterns, but provides no connection to domain-specific analysis algorithms.

## 2 Architectural Verification of CPS

Architectural models allow the description of early designs that can be analyzed to find and correct errors that otherwise would be discovered in later stages of development. The Architecture Analysis and Design Language (AADL) [11] is a modeling language that allows us to describe the software execution architecture of Cyber-Physical Systems. AADL captures the software execution elements like processes, threads, connections, flows, etc. and the hardware architecture including devices, processors, networks, and storage as well as the relationship between software and hardware that affect their execution like thread deployment, scheduling algorithms, connection allocation to networks, bandwidth, CPU capacity, error models, system modality etc. AADL models have been used in multiple DARPA [12,13] and US Army programs [14] with multiple analyses to verify critical software and system properties.

The analyses used on AADL models are typically applied by system designers who define the high-level architectural structures that are then incrementally refined. These designers are not experts in all analyses domains (e.g., real-time scheduling, control theory, safety, security, etc.). They are expected to execute analyses that automatically validate different properties without knowing the internals of these analyses, i.e., they used them as black boxes. Unfortunately, most analyses make implicit assumptions about the models that, if not met, will void their results. When this occurs, the analyses fail or produce invalid results, leaving the designers in the dark about the cause of these failures given their lack of expertise in the analysis domain.

As an example consider a real-time system that we want to analyze for schedulability. When applying the rate-montonic schedulability bound analysis to validate if all tasks (a.k.a. threads) will finished before their deadline, two sample assumptions are that (1) the deadlines of all tasks should be equal to their respective periods (a.k.a., implicit deadlines) and (2) that for any two tasks $\tau_i$ and $\tau_j$ in the system if $\tau_i$'s period is larger than $\tau_j$'s then $\tau_j$'s priority should be higher than $\tau_i$'s (known as rate-montonic priority assignement).

Analysis contracts are designed to explicitly model the assumptions of the analysis used. Then, our verification framework evaluates whether these assumptions are met in the architectural model (e.g., by the priorities, periods, and deadlines of the tasks from our example) preserving the validity of the analysis results. This evaluation is performed in multiple ways, including ensuring that the assumptions from different analysis do not contradict each other. The structure of the assumption validation and their dependencies are captured in an assurance argumentation that we discuss in Section 3.1.

## 3 Analysis Contracts

To evaluate the guarantees and assumptions that verification analyses make about architectural models we created analysis contracts [6]. In our previous work we defined these contracts between the analyses and the model for a fully finished architectural model.

In this paper we extend our previous work to (1) enable model refinement, (2) enable automatic analyses selection based on assumption satisfaction including level of refinement, and (3) enable contracts for exact and approximate analysis.

An analysis contract follows Hoare triples [1] of the form $\{P\}S\{P'\}$ where both $P$ and $P'$ are Boolean predicates over variables derived from architectural models, and $S$ is another predicate over the model. In traditional program verification, Hoare triplets encode that if $P$ holds, the execution of a statement $S$ will make $P'$ hold. In contrast to the traditional use of Hoare triples, the statements $S$ are left as an uninterpreted black box assumed to be correct. This approach allows us to re-use proofs for analysis algorithms without the need to re-encode these proofs. For instance, we have created contracts for schedulability where the post-condition $P'$ encodes that the worst-case response time of a task will never exceed its deadline. This post condition can be applied to utilization-bound tests where a response time is never calculated, or a response time test where the response time is calculated explicitly. In the future we plan to connect analysis proof mechanization such as PROSA [15] to the analysis contracts to close the loop.

To enable the reuse of domain-specific analysis algorithms we allow the implementation of predicates over model variables (e.g., thread periods, priorities, execution time, thread-to-processor bindings, errors, error propagations, etc) as imperative functions (imperative predicate) that return a Boolean to test a condition. The imperative predicate is then paired up with a symbolic formula that encodes the condition in a symbolic interpretation of the analysis domain (e.g., $\forall \tau_i \in \tau : R_i \leq D_i$ – with $R_i$ as the response time and $D_i$ its deadline). To encode this interpretation we defined a symbolic domain as the tuple $D_d = (V_d, S_d)$ with $V_d$ as the set of symbolic variables and $S_d$ as the set of predicates on $V_d$ that define the invariants of the domain.

An analysis contract is defined as a tuple $C_i = (V_d, Q, I, A, G, N)$ where, $Q$ is the set of model variables over which predicates are applied. $I$ is a set of input assumption predicates over $Q$ that defines whether the analysis predicate $N$ can run or not. This is necessary to ensure that the imperative function does not fail due to lack of data or incorrect data. $A$ is the set of pairs $(p_j, a_j)$ where $p_j$ is a predicate implemented as a call to a Boolean function in an imperative language or a reference to yet another contract and $a_j$ is an assertion over the domain variables $V_d$, modeling $p_j \implies a_j$. $G$ is an assertion over $V_d$ that is asserted as $I \implies (\bigwedge_j a_j | (p_j, a_j) \in A \wedge N \implies G)$.

We encode contracts as an AADL annex (an AADL extension mechanism to define sublanguages [11]) with the template shown in Listing 1.

```
annex contract {**
    contract <name> {
    queries
        <model var> =
            <query to obtain model data>
    domains
        <domain reference>
    input assumptions
        <Bool func to check data consistency>(
            <model vars>)
    assumptions
        <Bool func>(<model vars>)
        –> <symbolic assertion>
    analysis
```

```
            <Bool func>(<model vars>)
            –> <symbolic guarantee>
        }
    **};
```

**Listing 1: Contract Template**

## 3.1 Contract Argumentation

When Hoare triples are used to create contracts for program verification, these contracts are connected through the control flow of the program, and the predicates are evaluated against the value transformations of the program statements. In contrast, in an architectural model we do not have a unique control flow given that each analysis defines its own behavioral model where predicates can be evaluated. For instance, a schedulability model defines a task model as an infinite sequence of jobs that arrive with some specific periodicity. Each job consumes some limited amount of processing time from the processor it runs on, and has a timing requirement on when to finish such processing.

Analysis contracts are connected through an assurance argument flow. That is, first we start with the top-level claims needed to assure a system. From there we connect the analysis contracts that discharge such claims. These contracts, in turn, encode a number of assumptions (pre-conditions) that must be discharged for the analysis results to be valid. We then connect these assumptions to further contracts whose guarantees can discharge the assumptions of the previous analysis. This creates a *contract argumentation* that is used to prove the top-level claims.

The contract argumentation encoding starts with a verification plan that defines the set of top-level claims that must be verified in a model together with the analysis contracts that must be executed to prove these claims. The verification plan is defined as the tuple $P = (K, D)$ where $K$ is the set of pairs $(k_i, C_i)$ with $k_i$ as a predicate over variables of one of the domains $V_i \in D_i.V | D_i \in D$ and $C_i$ is the contract use to discharge the predicate. In turn, each contract specify a set of assumptions that must be validated in order for the results of the analysis to be valid. These assumptions can be satisfied either by simple model value comparisons, a complex behavioral validation that involves another analysis contract, or a imperative predicate that validates some data from the model. In order to combine multiple analysis to satisfy a claim or assumptions we have a special contract call *contract argument* of the form $\alpha = (C_\alpha, F_\alpha, G_\alpha, V_\alpha)$ where $C_\alpha$ is the set of contracts combined in the argument, $F_\alpha$ is the Boolean formula that combines the contracts and $G_\alpha$ is a predicate over domain variables $V_\alpha$ that is asserted (guarantee) if the formula holds.

## 3.2 SMT Encoding of Contract Argumentation

We implemented the contract argumentation as a constraint satisfaction problem in SMT. This is created using z3py (SMT encoded in python statements) as the conjunction of a set of constraints obtained from the contract argumentation. More specifically, we obtain the SMT encoding from the verification plan as described in Algorithm 1. In this algorithm `replG4C(F)` replaces the contract names with their respective guarantees within the Boolean formula $F$.

**Algorithm 1** getSMTEncoding($Plan$)

```
 1: $F \leftarrow \{k_i | (k_i, C_i) \in Plan.K\}$
 2: $T \leftarrow \{C_i | (k_i, C_i) \in Plan.K\}$
 3: while $T \neq \emptyset$ do
 4:     select $t$ from $T$ and remove it from $T$
 5:     if $t$ is argument then
 6:         $T \leftarrow T \cup t.C_\alpha$
 7:         $F \leftarrow F \cup (\text{replG4C}(t.F_\alpha) \implies t.G_\alpha)$
 8:     else if $\bigwedge_{i \in t.I} i$ then
 9:         for $p \in \{p | (p, a) \in t.A\}$ do
10:             if $p$ is contract then
11:                 $T \leftarrow T \cup p$
12:                 $F \leftarrow F \cup (\bigwedge_{\forall \{a | (p,a) \in t.A\}} a)$
                        $\wedge N \implies t.G)$
13:             else
14:                 $F \leftarrow F \cup (p \implies a)$
15:             end if
16:         end for
17:     end if
18: end while
19: return $F$
```

### *3.2.1 Discharging Claims on Partial Models*

The traditional use of SMT for program verification involves the creation of a constraint satisfaction problem by declaring constraints on symbolic variable valuations, and evaluating a high level claim. This evaluation is typically done by negating such a claim and asking the solver if there is any possible value assignment that will make this negated claim true, i.e., a counter example. If the solver cannot find a counter example then we know that the claim has been proven.

Looking for counter examples requires a full model. However, when we allow partial models, as in our case, we look for whether the constraints that the partial model creates still allow for value assignments that satisfy our top-level claim. Hence, for a partial model, we do not negate the claim and use the solver to find models that cannot possibly be completed to satisfy the top level claim.

## 4 Model and Assurance Refinement

The partial model verification presented before allows us to verify partial models. This is important to allow early analysis but also verify that subsequent refinements do not invalidate previous analysis.

AADL has explicit mechanisms to refine a model[1]. With these mechanisms it is possible to add details to a model. For instance, assume we have a model where we are interested in the end-to-end latency of a data flow through the system. In an early model it is possible to define the flow as the sequence of AADL flow specifications through components and their expected delays without specifying threads, scheduling, and other details. At this stage, then, we can apply an analysis that simply adds up the intermediate delays to get the end-to-end latency.

Later on, as the model is refined to include details of threads and their scheduling, we use a more sophisticated analysis

[1]Not discussed here for brevity.

that can verify whether or not the threads meet intermediate deadlines and whether the final more precise end-to-end latency based on such deadlines can still be met.

### 4.1 Determining Pending Proof Obligations

The verification of partial models implicitly disregards constraints over symbolic variables for which we do not have data yet. For instance, consider the constraint that the priorities assigned to the threads follow the rate-monotonic priority assignment (i.e., the shorter the period the higher the priority) needed for a utilization bound test. If no priorities have been assigned, then the verification of the partial model will find priorities that can satisfy this constraint. We call these constraints proof obligations whose verification is deferred to a later stage when the corresponding data is available.

At the end of our design process we should have a complete model with all the details necessary to verify all the claims. At this stage, we follow the approach used for program verification by negating the claims and searching for counter examples. Any counter examples that we find informs us that we are either still missing information in the model such that some proof obligations cannot be discharged, or that we have all the detail but the model does not satisfy all requirements.

## 5 Assumption-Based Path Selection

Our contract argumentation allows us to select argumentation paths based on assumption satisfaction. We can select the path based on (1) the level of refinement of the model or (2) the design choices made to satisfy different assumptions.

### 5.1 Refinement-Based Path Selection

Path selection based on refinement level is implemented through `input assumptions`. More specifically, input assumption check if there is enough data in the model to run a certain analysis contract. Missing data can lead to the choice of an alternative contract. This choice is encoded as a contract argument where the Boolean formula is a disjunction of the two contracts where each contract check for complementary input assumption conditions. Listing 2 shows an example.

```
annex contract {**
verification plan myPlan {
    claims
        EndToEndDelayArgument–>
        And([E2EResp[i] <= E2ELatency[i]
                for i in range(len(E2EResp))]
}

argument EndToEndDelayArgument {
    argument
        Or(E2ESched, E2ESFlowSpec)–>
            And([E2EResp[i] <= E2ELatency[i]
                    for i in range(len(E2EResp))]
}

contract E2ESched {
    input assumptions
        allSchedDataPresent()
        ...
}

contract E2EFlowSpec {
    input assumptions
        notAllSchedDataPresent()
        ...
}
**}
```

**Listing 2: Path Selection Based on Refinement**

This structure allows us to select the argumentation path based on amount of detail a model has.

### 5.2 Design-Choice Selection

The other way to select an argumentation path is by assumptions on design choices. For instance, it is possible to use the rate-monotonic bound analysis if we choose to use rate-monotonic priorities (among other things). However, if for some reason we decide to not follow rate-monotonic priorities, it is not possible to use the bound to analyze schedulability but it is possible to use the response time test. This is encoded in the argument presented in Listing 3.

```
annex contract {**
argument schedulability
    argument
        Or(RMBound, RTA)
        ...
}

contract RMBound {
    assumptions
        RMPriorities(periods, priorities )
    analysis
        RMBoundTest(...)
}

contract RTA {
    assumptions
        ...
    analysis
        RTATest (...)
}
**}
```

**Listing 3: Path Selection Based on Assumptions**

## 6 Exact / Over-Approximation Contracts

Another important aspect of domain-specific analyses is that some of them perform an exact analysis while others perform an over-approximation. This fact was not captured in previous work [6] where we considered all analyses as over-approximations. More specifically, when an analysis verification returns true then we can infer the guarantee, that is, the analysis implies the guarantee as presented in Section 3. Furthermore, when the analysis returns false, we cannot conclude anything about the guarantee given that there could exist a more exact analysis that can still verify the guarantee, hence the single implication.

An exact analysis, on the other hand, is not only able to prove that a guarantee holds if it returns true, but also is able to assert the negation of the guarantee if it returns false. This is because this analysis has been proven to be exact, i.e., if it fails to verify the guarantee we can get a counter-example that exists in the model. In our annex we capture an exact analysis using a double implication arrow <-> instead of the simple implication arrow -> in the assumptions and analysis of a contract shown in lines 13 and 16 of Listing 1, as well as lines 4 and 11 of Listing 2 for assumptions, analysis, verification plan and argument respectively. Similarly, the SMT generation algorithm is modified to replace the simple implication $\Longrightarrow$ in lines 7, 12, and 14 with a double implication $\Longleftrightarrow$ in the exact case.

## 7 Conclusion

In this paper we presented our work in progress on the development of an efficient analysis contract argumentation approach that takes advantage of efficient domain-specific analysis algorithms, enables incremental analysis of model refinements and encodes assume guarantee reasoning in symbolic domains in SMT. This approach has allowed us to describe dependencies related to assumptions, design choices, and refinement level that automatically selects the appropriate analysis whose preconditions can be met or report errors if no options are available to validate the claims of a verification plan.

## 8 Acknowledgement

The following markings MUST be included in work product when attached to this form and when it is published. For purposes of blind peer review, markings may be temporarily omitted to ensure anonymity of the author(s). Copyright 2023 Carnegie Mellon University. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. Carnegie Mellon© is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM23-0179

## References

[1] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, 1969.

[2] A. Benveniste and et al., *Contracts for System Design*, 2018.

[3] D. Cofer and et al., "Compositional Verification of Architectural Models," in *NASA Formal Methods*, 2012.

[4] M.-Y. Nam, D. de Niz, L. Wrage, and L. Sha, "Resource allocation contracts for open analytic runtime models," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, 2011, pp. 13–22.

[5] D. Jackson, "Alloy: A language and tool for exploring software designs," *Communications of the ACM*, vol. 62, pp. 66–76, 08 2019.

[6] I. Ruchkin, D. De Niz, S. Chaki, and D. Garlan, "Contract-based integration of cyber-physical analyses," in *EMSOFT*, 2014.

[7] Y. Kesten, A. Pnueli, and L.-o. Raviv, "Algorithmic verification of linear temporal logic specifications," in *Automata, Languages and Programming*, K. G. Larsen, S. Skyum, and G. Winskel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 1–16.

[8] G. Holzmann, "The model checker spin," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[9] G. Brau, J. Hugues, and N. Navet, "Towards the Systematic Analysis of Non-Functional Properties in MBE for RTES," *Science of Computer Programming*, 2018.

[10] T. E. Wang, Z. Daw, P. Nuzzo, and A. Pinto, "Hierarchical contract-based synthesis for assurance cases," in *NASA Formal Methods: 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24–27, 2022, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 175–192.

[11] "Architecture Analysis and Design Language (AADL)," SAE International, USA, Standard, Mar. 2009.

[12] "High-Assurance Cyber Military Systems," https://www.darpa.mil/program/high-assurance-cyber-military-systems, accessed: 2023-02-26.

[13] "Cyber Assured Systems Engineering (CASE)," https://www.darpa.mil/program/cyber-assured-systems-engineering, accessed: 2023-02-26.

[14] "Applying AADL Expertise to Future Vertical Lift Modeling," https://www.sei.cmu.edu/publications/annual-reviews/2021-year-in-review/year_in_review_article.cfm?customel_datapageid_315013=335714, accessed: 2023-02-26.

[15] F. Cerqueira, F. Stutz, and B. B. Brandenburg, "Prosa: A case for readable mechanized schedulability analysis," in *ECRTS*, 2016.

# Towards a Methodology to Design Provably Secure Cyber-Physical Systems

***Felipe Lisboa Malaquias, Georgios Giantamidis, Stylianos Basagiannis***
*Collins Aerospace, Applied Research and Technology Centre, Ireland; email: {firstname.lastname}@collins.com*
***Simone Fulvio Rollini***
*Collins Aerospace, Applied Research and Technology Centre, Italy; email: simonefulvio.rollini@collins.com*
***Isaac Amundson***
*Collins Aerospace, Applied Research and Technology Centre, United States; email: isaac.amundson@collins.com*

## Abstract

*The inordinate financial cost of mitigating post-production cybersecurity vulnerabilities in cyber-physical systems (CPS) is forcing the industry to rethink systems design cycles: greater attention is being given to the design phase – with the goal of reducing the attack surface of systems at an early stage (i.e., before silicon tape out). Fortunately, formal methods have advanced to the point that they can address such needs and contribute towards achieving security certification. However, new methods and tools focusing on industrial scalability and usability for systems engineers are required. In this ongoing research paper, we describe a framework that will help systems engineers to: a) design cyber-assured CPS using a Model Based Engineering (MBE) approach; b) formally map security requirements to different hardware and software blocks in the model; and c) formally verify security requirements. Based on the nature of each requirement, our framework collects formal correctness evidence from different tools: while high-level architectural properties are suitable for a contract- or ontology-based reasoning, more complex properties with rich semantics require the use of model checking or theorem proving techniques.*

*Keywords: Formal Methods, Cybersecurity, Cyber-Physical Systems, Model Checking, Theorem Proving.*

## 1 Introduction

In 2020, researchers estimated that there were 12 billion active *Internet of Things* (IoT) devices, and that number would at least double within five years [1]. The attack surface for IoT devices is large, and an attacker may use several attack vectors to compromise a device, ranging from physical side-channel attacks to programming bugs like buffer and arithmetic overflows [2].

Considering that many of these IoT devices are used in critical *Cyber-Physical Systems* (CPS), security breaches can lead to seriously hazardous outcomes, both in terms of human life and financial loss. A well-publicised example of an IoT security breach is the remote hijacking of a Jeep on a U.S. highway [3]: white hat hackers were not only able to manipulate non-critical systems (e.g. display, air conditioning), but they were also able to control the engine and brakes.

Given the high stakes, guaranteeing that these devices correctly implement the appropriate security countermeasures is crucial. Furthermore, in order to achieve a high degree of trust regarding security claims, engineering teams must go beyond dynamic testing of software and hardware components and tackle the problem with *formal methods* tools.

While in the past formal methods encountered a strenuous barrier regarding industry adoption, theoretical advances led by academia, software engineering best practices, and the exponential growth of computational power have all contributed to reaching the current state-of-the-art: tools are now sophisticated enough to abstract from mathematical theories and provide user-friendly interfaces for systems engineers. This is highly beneficial, given the fast-paced needs of the industry.

Although recent research has successfully proposed frameworks for formal reasoning about cybersecurity in CPS, no comprehensive framework exists for modelling and formally verifying general-purpose CPS, such as IoT devices. Therefore, we propose a framework that will allow engineers to write requirements (not only security-related but also safety, timing, and functional requirements), design system architectures, and gather, on the same model, formal evidence that the stated requirements are satisfied either by the architectural model or by specific component implementations.

## 2 Related Work & Background

### *2.1. Model-Based Engineering & AADL*

Model-Based Engineering (MBE) has emerged as a key set of methodologies to design complex systems [4]. One widely-adopted MBE technology is the *Architecture Analysis & Design Language* (AADL) [5]. Initially developed for avionics

This document contains no technical information subject to EAR or ITAR. Work is funded under the REWIRE Horizon Europe project (REWiring the composItional security veRification and assurancE of systems) with grant agreement No. 101070627.

applications, AADL has since been used to design a wide range of embedded real-time system architectures, largely due to its language constructs for specifying both software and hardware configurations. Moreover, AADL has a reference implementation called OSATE [6], which is an open-source modelling environment that comes with a few built-in analysis tools such as flow control and schedulability. Because OSATE is based on the Eclipse framework, creating new analysis plugins is relatively straightforward.

AADL includes an *annex* mechanism for extending the base grammar, thereby supporting new language features and analyses. One such annex is the *Assume Guarantee REasoning Environment* (AGREE) [7], which is a compositional assume-guarantee-style formal analysis tool. AGREE attempts to prove properties about one layer of the architecture using properties allocated to subcomponents. The composition is performed in terms of formal assume-guarantee contracts that are provided for each component. Assumptions describe the expectations the component has on its inputs and the environment, while guarantees describe bounds on the component's behaviour. The model checker then attempts to find any model execution traces that violate these contracts using one of several Satisfiability Modulo Theories (SMT) solvers. If the model checker covers all reachable states in the model without finding a violation, the model is proven to satisfy its contracts.

Another important annex for reasoning over AADL models is Resolute [8], which includes a language for embedding assurance cases in AADL models and a tool for evaluating the validity of the associated evidence. An assurance case is a structured argument, supported by evidence that a system will operate as intended in a specified environment. Because high-assurance products generally undergo certification at the system level, there is a natural mapping between a system design and the corresponding assurance argument. Resolute takes advantage of this alignment by enabling the specification of the assurance argument directly in the AADL model. The assurance case can then be instantiated and evaluated with elements specified in the model. The resulting assurance case can be viewed in the modelling environment, or exported to graphical tools such as AdvoCATE [9]. Resolute assurance cases are at the core of our approach, and we describe them in greater detail in Section 3.

Our choice of using AADL over other MBE languages such as SysML [10] is informed by multiple factors. First, AADL was designed for specifying *hierarchical* system architectures, enabling the composition of systems from subsystems, and refinement from abstract to concrete types. It includes first class objects for representing components that comprise embedded systems such as memory, buses, processors, threads, subprograms, and data. SysML, on the other hand, is more abstract and thus better-suited for early stages of system engineering. Second, AADL has a sufficiently rigorous run-time semantics, enabling a wide range of analyses that would otherwise not be possible. And third, AADL's annex support cannot be overstated. The ability to extend the language in order to perform new types of analyses is critical in the rapidly evolving – and heavily regulated – CPS design space.

### 2.2. Cyber-Assured Systems Engineering Framework

Cofer et al. recently developed BriefCASE [11], an AADL-based framework for designing, building and assuring cyber-resilient systems. In that work, high-level security requirements are mapped to seL4 microkernel [12] features via a (very) trustworthy tool chain. Although they did succeed at creating a framework for crafting formally verified secure applications, their work did not focus heavily on hardware security, which plays a fundamental role in protecting a wide range of CPS including IoT devices. In contrast, we propose a framework that allows system engineers to specify a wide range of system requirements and map them to the appropriate software or hardware block.

For example, one might require a platform capable of performing *trusted boot* to verify the authenticity of an over-the-air firmware update, or a platform capable of executing hardware-implemented *crypto-primitives* (e.g., symmetric or asymmetric encryption, hash functions, etc.). While the former security goal could be achieved through the use of a hardware *Root of Trust* (RoT) acting as the *Trusted Platform Module* (TPM), the latter would require *Instruction Set Extensions* (ISEs) or memory-mapped crypto-accelerators. These solutions are outside the scope of what BriefCASE currently offers.

Another important work that introduces a tool aimed at formal reasoning about CPS is KeYmaera X [13], a theorem prover for differential dynamic logic (dL). KeYmaera X introduces important advances in formal verification of CPS – its logic is well suited for reasoning about discrete and continuous dynamics, which are useful to encode functional and safety properties of CPS. It has been used, for instance, to model and verify the safety of flight collision avoidance software [14] and train controls with air pressure brakes [15]. In comparison, our work focuses instead on characterising software/hardware running in CPS and its underlying cybersecurity properties.

Finally, VRASED [16] is a HW/SW co-design that implements a formally verified Remote Attestation protocol. To achive that ultimate goal, VRASED relies on different formal verification techniques: a custom hardware module (used to reset the internal state of the micro-controller studied in their paper in case the code to be attested is compromised) has its correctness specified with Linear Temporal Logic (LTL) and checked with NuSMV – a model checker; the overall soundness and correctness of the resulting system is modelled and proved in a theorem prover; and finally, they also make use of pre-verified cryptographic code [17]. While VRASED is an impressive effort, their architecture is bare-metal and specific to a family of micro-controllers, such as MSP430. The goal of our work, contrarily, is to leverage MBE tools in conjunction with formal verification techniques to formally reason about more generic, richer architectures (e.g. with RISC-V cores and Trusted Execution Environments (TEE)).

*We thus propose building a framework that: a) provides the tools to formally reason about security solutions implemented by both software and hardware; and b) maps security requirements to evidence gathered not only from AADL and its plugins, but also external tools, such as Coq.*

## 3 Framework Description

Figure 1 presents an abstract architectural overview of the tool-chain, under development at the time of writing. Note that, while some of the tools are embedded within OSATE, some are called by Resolute as an external source of formal correctness evidence.

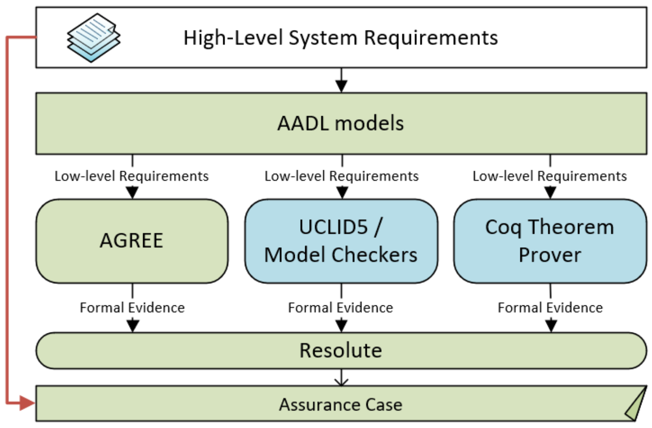


**Figure 1: Toolchain. Legend: 🟩 OSATE tools 🟦 External tools**

The framework we propose is intended to be used according to the following workflow:

1. Security requirements are specified;
2. System architecture is modelled in AADL;
3. Formal analysis of model is performed using AGREE to verify the design satisfies security properties;
4. Hardware and Software components are implemented manually, or through verified synthesis [18];
5. Where possible, formal analysis is performed on component implementations – this could be done by a variety methods (e.g., model checking and theorem proving);
6. Component implementations are integrated into a system build;
7. System testing is performed;
8. An assurance case is generated using Resolute, confirming that security goals are support by evidence (maintained by the framework).

In the following, we detail some of the steps presented above.

***Requirement Specification (Step 1)***

Consider three illustrative high-level security requirements, which are motivated by industrial use-case scenarios for embedded systems controlling safety-critical operations of CPS:

*R1: "The TEE shall provide the necessary mechanisms to enable the isolated execution of sensitive functions in enclaves"*

*R2: "The crypto schemes that will be used for the secure communication of devices shall be provably secure, based on well-accepted underlying assumptions"*

*R3: "The system shall include mechanisms that detect replay attacks and can tell if newer messages or part of them are unauthorised repetitions of previously authorised exchanges"*

***Architecture Modelling (Step 2)***

Based on the security requirements from Step 1 (as well as other high-level requirements), engineers use AADL to model a system architecture that captures the appropriate security solutions. Here, for illustrative purposes, consider the following architecture:

- *On the hardware layer*: a RISC-V core, such as the 2-stage pipeline 32-bit Ibex core[1] or the 6-state 64-bit Arianne core[2], and a hardware RoT (such as the OpenTitan[3]), which is assumed to have a crypto co-processor capable of enhancing the performance of functions such as AES and SHA-256.
- *On the firmware layer*: a custom-tailored Keystone TEE [19], configured to use not only the RISC-V Physical Memory Protection (PMP) Registers – primitives for ensuring memory isolation between secure enclaves – but also custom RISC-V instructions to access crypto co-processors and the RoT;
- *On the software layer*: a set of secure applications (Keystone enclave application), such as attestation agents, evidence collectors, and SW/FW update agents.

Figure 2 shows an abstraction of the proposed architecture.[4]

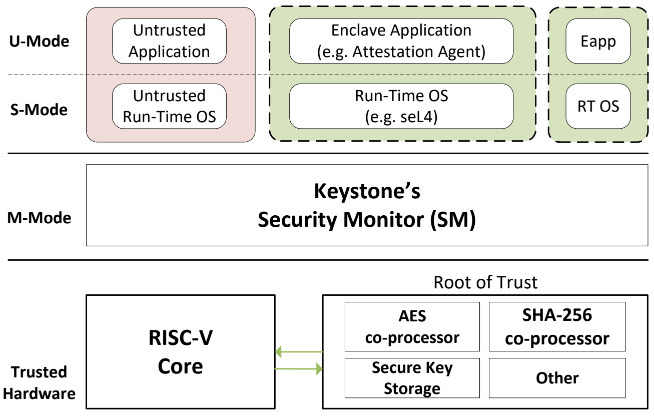


**Figure 2: Abstraction of the example architecture.**

Modelling software and hardware components in AADL is a straightforward and well-documented process: software is described using components such as *processes* and *threads*, while hardware components include *processors*, *buses*, and *memories*. Modelling the TEE and its properties in AADL however is still an open problem and we consider it to be part of our research.

Note that the architecture proposed above is merely illustrative and its security implications are not the focus of this paper. Rather, here our focus is on providing an overview of

[1]https://ibex-core.readthedocs.io/en/latest/
[2]https://github.com/lowRISC/ariane
[3]https://opentitan.org/
[4]The architecture used as example here is similar, in some sense, to Keystone's architecture [19], although not entirely. Keystone, in this case, would have to be configured by the *Keystone Programmers* [19] to use the custom proposed hardware.

a methodology to formally verify a diverse range of security requirements.

***Translation of Higher-Level Requirements into Low-Level Specifications (Step 5)***

In order to determine whether individual component implementations satisfy their requirements, a necessary prerequisite is the translation of those requirements into a formal representation that facilitates (semi-)automated, computer-aided analysis/reasoning.

For low-level functional requirements characterising component behaviour, this representation can, for example, be a form of modal logic, such as *Linear Temporal Logic* (LTL) or *Computation Tree Logic* (CTL) [20], or *differential dynamic logic* (dL) [21].

For intermediate-level non-functional requirements, approaches for describing constraints on system architecture / component interconnections (or other desired properties of a non-functional nature) can vary. For example, in cases where properties are quite abstract and their satisfaction is difficult to define, so called soft-goal approaches are more appropriate [22], while in cases where properties are more concrete, a (constraint) logic-based approach is preferred [23]. Requirements *R1*, *R2*, and *R3* fall under the latter category. *R1*, for instance, can be formalised in first-order logic as follows:

$$\begin{aligned}&\forall e \in Enclaves,\\&\forall f \in Functions, sensitive(f) \implies\\&allocated\_on(f,e) \implies\\&ensured\_isolated\_execution(f,e)\end{aligned} \tag{1}$$

In natural language, *R1* is said to be satisfied if any arbitrary sensitive function $f$ is allocated inside of a secure enclave $e$.

Development of automated requirement formalisation tools is out of scope for this work. Furthermore, it will likely be the case that not all requirements can be formalised in a manner that permits automated reasoning. For now, we assume that translating high-level requirements into low-level specifications is a manual task to be performed by the framework user and verified through manual review.

***Formal Verification of Requirements (Step 5) and System Assurance (Step 8)***

Here, we detail how formal analysis can be performed (Step 5) considering requirements *R1*, *R2*, and *R3*; and how we can use Resolute to generate assurance cases, which can be used to provide confidence that cybersecurity requirements have been satisfied in both the design and implementation.

Requirement *R1* can be checked using Resolute, since it is structural in nature. We can use Resolute to "traverse" the model and check that a component implementing the desired functionality is present, cannot be bypassed, and has been implemented appropriately. In the proposed architecture, Keystone's Secure Monitor (SM) [19] implements enclave isolation by manipulating RISC-V *Physical Memory Protection* (PMP) registers, and thus, "*isolated execution of sensitive functions in enclaves*" is achieved.

Listing 1 is a first attempt at writing an assurance argument for requirement *R1* in Resolute. In the listing, `SW.Impl` is a rather simplified representation of a software process hosted in a system equipped with Keystone, where `Eapp` is a sensitive function. The Resolute goal `Iso_Exec` traverses the model to check that: 1) the enclave where `Eapp` executes is properly implemented and initialised, 2) `Eapp` executes on the enclave, and 3) Applications on U-Mode or S-mode cannot bypass the Security Monitor (see Figure 2). We omit the connections between threads for conciseness.

**Listing 1: Verifying *R1* with Resolute.**

```
process implementation SW.Impl
    subcomponents
        Eapp : thread Eapp.Impl;
        RT : thread RT.Impl;
        Enclave : thread Enclave.Impl;
        SM : thread SM.Impl;
    annex resolute {**
        argue Iso_Exec (this.Eapp, this.
            Enclave, this.SM)
    **};
end SW.Impl;

goal Iso_Exec (eapp : component, encl :
     component, sm : component) <=
    strategy: "Reason about architecture";
    enclave_exists(encl) and
    enclave_initialized(encl) and
    allocated_on(eapp, encl) and
    sm_not_bypassed(sm, encl)
```

The second requirement, *R2*, is more difficult. Informally, "provably secure" is naturally more complex than "shall provide". Here, for simplicity, let us first assume that the "*crypto-schemes that will be used for the secure communication of devices*" can be simply reduced to AES and SHA-256, as these are the co-processors specified in the architecture. Realistically, this is a strong assumption, since a real device would also require asymmetric encryption schemes, which are mostly implemented in software.

We can assume that the cryptographic algorithms themselves (AES and SHA-256) are secure by design and focus our efforts in proving that the actual hardware implementations are correct against a high-level formal specification of these algorithms. For that goal, considering that the architecture proposes the use of hardware co-processors, we could either: a) perform typical verification techniques, such as property checking with SystemVerilog Assertions (SVA) on existing HW IPs or b) produce correct-by-design *Register Transfer Level* (RTL) code using Coq *Domain Specific Languages* (DSLs), such as Kôika [24]. At this phase of our research, we explore the second option, as depicted in Listing 2. Notably, we prove that the hardware implementation of the RISC-V standardised crypto custom instructions [25] is correct against the instruction semantics, expressed in SAIL [26].

**Listing 2: Verifying *R2* with Resolute.**

```
system implementation HW.Impl
    subcomponents
        AES : processor AES.Impl;
        SHA_256 : processor SHA_256.Impl;
    annex resolute {**
        argue Correct_By_Design (this)
    **};
end HW.Impl;

goal Correct_By_Design (sys : system) <=
    ** "RTL code is provably correct" **
    forall(proc : processors(sys)) .
        analysis("coq", proc)
```

Finally, *R3* requires that the architecture includes mechanisms for detecting intruder operations such as message replay attacks. In such an attack, a malicious agent intercepts a message and/or controls its delivery to the intended target, thereby disrupting system operations or obtaining unauthorised information. If a protection mechanism is enabled (e.g., by including timestamps), then a model checker could explore whether or not a successful replay-attack state is reachable in which the protection would fail to detect the repetition of the message within a specific time threshold.

Devices that are part of communication networks are commonly modelled together with an intruder model (e.g., Dolev-Yao [27]) that can perform attack operations against eavesdropped messages. Model checking using SPIN [28] or OFMC [29] can provide evidence that proves the absence of a series of such attacks. Recent approaches also involve the usage of TAMARIN [30] in an attempt to verify cryptographic protocols using adversaries within the tool itself. In the case of *R3*, since the requirement can be directly modelled in the language of a model checker, the Resolute assurance argument can take the form of a simple predicate: $safe\_against\_replay\_attacks()$, supported by evidence generated by the model checking tool.

## 4 Conclusion & Next Steps

We propose a framework aimed at modelling and formally verifying cyber-assured CPS under the following design principles: a) security requirements are allocated to either system software or hardware, b) the system is modelled in a language (e.g., AADL) with sufficiently rich semantics that enable formal analysis, c) formal methods are applied at multiple points in the development workflow (compositional reasoning at the architecture level, verified synthesis for component generation, model checking and theorem proving of hardware and software component implementations, etc.), and c) an assurance case is generated that substantiates cybersecurity claims with evidence from formal analyses (and other workflow processes managed by the framework).

Currently, we focus our efforts on modelling Keystone and custom hardware in AADL – an open problem – since AADL has not been previously used to model TEEs. A subsequent challenge is how to scale the approach to model larger systems.

## References

[1] K. L. Lueth, “State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time,” *IoT Analytics*, Nov 2021.

[2] K. Keerthi, I. Roy, A. Hazra, and C. Rebeiro, “Formal verification for security in IoT devices,” *Security and Fault Tolerance in Internet of Things*, pp. 179–200, 2019.

[3] A. Greenberg, “Hackers remotely kill a Jeep on the highway-with me in it,” *Wired*, Jul 2015.

[4] P. de Saqui-Sannes and J. Hugues, “Combining SysML and AADL for the design, validation and implementation of critical systems,” in *ERTS2 2012*, p. 117, 2012.

[5] P. H. Feiler, D. P. Gluch, and J. J. Hudak, “The architecture analysis & design language (AADL): An introduction,” tech. rep., Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[6] P. Feiler, “The open source AADL tool environment (OSATE),” tech. rep., Carnegie Mellon University Software Engineering Institute, 2019.

[7] D. Cofer, A. Gacek, S. Miller, M. W. Whalen, B. LaValley, and L. Sha, “Compositional verification of architectural models,” in *NASA Formal Methods: 4th International Symposium, NFM 2012, Norfolk, VA, USA, April 3-5, 2012. Proceedings 4*, pp. 126–140, Springer, 2012.

[8] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, “Resolute: an assurance case language for architecture models,” *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 19–28, 2014.

[9] E. Denney and G. Pai, “Tool support for assurance case development,” *Automated Software Engineering*, vol. 25, September 2018.

[10] M. Hause *et al.*, “The SysML modelling language,” in *Fifteenth European Systems Engineering Conference*, vol. 9, pp. 1–12, 2006.

[11] D. Cofer, I. Amundson, J. Babar, D. Hardin, K. Slind, P. Alexander, J. Hatcliff, G. Klein, C. Lewis, E. Mercer, *et al.*, “Cyberassured systems engineering at scale,” *IEEE Security & Privacy*, vol. 20, no. 3, pp. 52–64, 2022.

[12] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, “seL4: formal verification of an OS kernel,” in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009* (J. N. Matthews and T. E. Anderson, eds.), pp. 207–220, ACM, 2009.

[13] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, “KeYmaera X: An axiomatic tactical theorem prover for hybrid systems,” in *CADE* (A. P. Felty and A. Middeldorp, eds.), vol. 9195 of *LNCS*, pp. 527–538, Springer, 2015.

[14] R. Cleaveland, S. Mitsch, and A. Platzer, “Formally verified next-generation airborne collision avoidance games in ACAS X,” *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 1, pp. 1–30, 2023.

[15] S. Mitsch, M. Gario, C. J. Budnik, M. Golm, and A. Platzer, “Formal verification of train control with air pressure brakes,” in *RSSRail* (A. Fantechi, T. Lecomte, and A. Romanovsky, eds.), vol. 10598 of *LNCS*, pp. 173–191, Springer, 2017.

[16] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, “Vrased: A verified hardware/software co-design for remote attestation.,” in *USENIX Security Symposium*, pp. 1429–1446, 2019.

[17] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, “Hacl*: A verified modern cryptographic library,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1789–1806, 2017.

[18] E. Mercer, K. Slind, I. Amundson, D. Cofer, J. Babar, and D. Hardin, “Synthesizing verified components for cyber assured systems engineering,” in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 205–215, 2021.

[19] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, “Keystone: An open framework for architecting trusted execution environments,” in *Proceedings of the Fifteenth European Conference on Computer Systems*, pp. 1–16, 2020.

[20] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, “Model checking and the state explosion problem,” *Tools for Practical Software Verification: LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures*, pp. 1–30, 2012.

[21] A. Platzer, “Differential dynamic logic for hybrid systems,” *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[22] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional requirements in software engineering*, vol. 5. Springer Science & Business Media, 2012.

[23] J.-P. Katoen, T. Noll, H. Wu, T. Santen, and D. Seifert, “Model-based energy optimization of automotive control systems,” in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 761–766, IEEE, 2013.

[24] T. Bourgeat, C. Pit-Claudel, and A. Chlipala, “The essence of Bluespec: a core language for rule-based hardware design,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 243–257, 2020.

[25] B. Marshall, G. R. Newell, D. Page, M.-J. O. Saarinen, and C. Wolf, “The design of scalar AES instruction set extensions for RISC-V,” *Cryptology ePrint Archive*, 2020.

[26] A. Armstrong, T. Bauereiss, B. Campbell, A. Reid, K. E. Gray, R. Norton-Wright, P. Mundkur, M. Wassell, J. French, C. Pulte, *et al.*, “ISA semantics for ARMv8-a, RISC-V, and CHERI-MIPS,” 2019.

[27] D. Dolev and A. Yao, “On the security of public key protocols,” *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[28] G. J. Holzmann, “The model checker SPIN,” *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[29] A. A. et al, “The AVISS security protocol analysis tool,” in *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, vol. 2404 of *Lecture Notes in Computer Science*, pp. 349–353, Springer, 2002.

[30] D. A. Basin, C. Cremers, J. Dreier, and R. Sasse, “Tamarin: Verification of large-scale, real-world, cryptographic protocols,” *IEEE Secur. Priv.*, vol. 20, no. 3, pp. 24–32, 2022.

# Automatic Test Value Generation for Ada

***L. Creuse, M. Eyraud, V. Garèse***
*Adacore; email: {creuse,eyraud,garese}@adacore.com*

## Abstract

*This article introduces novel tools to automatically generate pertinent Ada values in order to produce higher quality tests for Ada subprograms. A first tool will generate valid Ada values based on a structural analysis of the types of the parameters of the subprogram under test following various customizable strategies. Those values will then be filtered in order to satisfy the specifications of the subprogram, and new coverage criteria for executable specifications will be used to assess the relevance of the generated testsuite. This first set of values will then be used as seeds both for a fuzzing process, and a symbolic execution campaign, from which values of interest will be then extracted. This integrated process will enable users to generate a high value starting test corpus, which can then be expanded upon by domain-specific tests.*

*Keywords: Ada, generation, fuzzing, symbolic execution, automatic testing,*

## 1 Introduction

Software occupies a predominant place within critical systems, whether civil or military: aeronautics, railways, space, financial, medical. Failures and vulnerabilities in such software can have dramatic consequences in human, material or environmental terms. The Ada language is widely used in the development of such software. It is therefore essential to be able to ensure a high level of quality and security for the software developed in Ada. Whatever validation and verification techniques are used throughout the development process, the essential method for revealing run-time errors is testing.

Testing a program consists of executing it on a finite set of input values, so as to detect discrepancies between its actual behavior and the expected behavior. Although testing is the most widely used software validation method in the industrial context, tests are written manually most of the time. Writing these tests, as well as maintaining them, is one of the difficult and costly points for verification and validation teams.

The field of test generation has undergone major developments in recent years, which have seen the previously separate technologies of structure-aware random generation, fuzzing and symbolic execution being combined into a single system.

There are tools that generate Ada test input values [1, 2, 3], with various configurable generation strategies. However, there are limitation with these approaches, such as a lack of generation of discriminated records, with or without variant parts or a lack of integration with other forms of dynamic analysis, such as fuzzing campaigns and symbolic execution.

The tools being developed aim to lift some of these limitations by generating values for a wider range of types, to port test generation techniques not yet available to the Ada ecosystem and to combine various dynamic analysis methods to create a single consolidated test corpus. It is important to note that the work remains restricted to a subset of the Ada types. Notably, tagged types and access types are not in the scope of the project.

As shown on Figure 1, the idea is to combine several existing techniques for generation of test values and make them cooperate. These techniques include random generation based on types, described in section 3, fuzzing (see section 5) and symbolic execution (section 5.2). In addition, as contracts, in particular pre and postconditions, are first class citizens in Ada, the tool should take them into account to guide the generation process.

This development is being led by AdaCore, conjointly with Thales Research and Technologies, as part of the RAPID initiative [4].

## 2 Related work

The main motivation for developments centered around test generation is the discovery of vulnerabilities, mainly in C or C++ code bases, languages well known for their security flaws. This breakthrough in test generation techniques has been driven by the undeniable success of fuzzing as the main technique for vulnerability discovery, supplanting static analysis for this activity in many domains, and becoming one of the leading tools in black hat and white hat hacking.

In parallel, the academic symbolic execution platforms KLEE [5] and CBMC [6], dedicated to the C language, have seen their use grow in industry. These platforms are now co-developed by communities of academics and industrialists. Other symbolic execution platforms continue to be developed (SymCC [7], Driller, Munch) which also offer integration with fuzzing tools.

Finally, random testing is probably the least expensive and simplest test generation technique to implement. However, the relevance of the generated tests is strongly linked to the constraints on the data, expressed through types or explicit preconditions, which requires an additional programming effort. The founding tool of this approach, QuickCheck [8] initially developed for the Haskell language, has given many successors, such as recently Hypothesis [9] for the Python language or proptest [10] for the Rust language. Some of

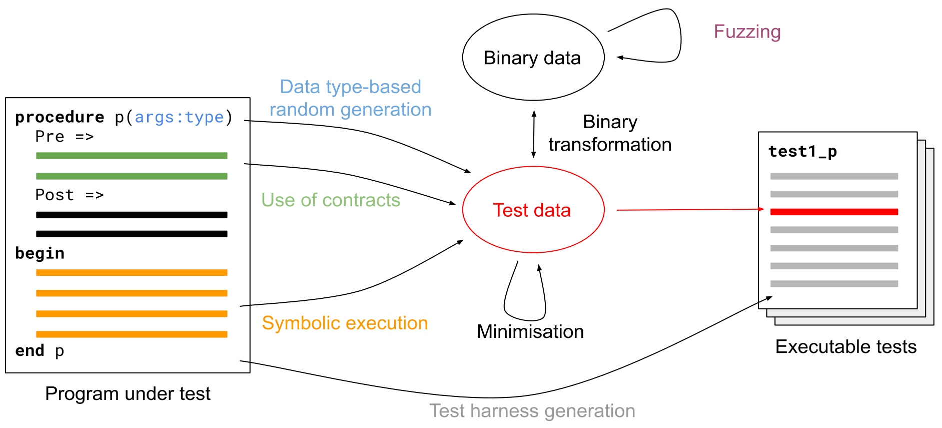


**Figure 1: Generation methods and interactions among the tools**

these property based-testing tools have started integrating with fuzzing engines to ease the use of the fuzzer and make it protocol-aware, such as Hypofuzz [11].

As these different test generation techniques have gained sufficient maturity for languages such as C, it now seems necessary to share them with the Ada community. Theoretically, the tools available for C can be applied to Ada code, but they require adaptations on several points, both for the kind of uses this development is aimed at, and languages specificities in the Ada type model. In addition to the necessary switch to Ada tooling, the techniques available for C are generally aimed at testing complete programs, and not unit testing. Moreover, they are generally used for applications exposing a simple interface (reading a binary file for example), and not for embedded code that must support, for example, multiple inputs from sensors, which are more difficult to test by fuzzing. To this end, we will rely on the existing technologies GNATtest, GNATcoverage and GNATfuzz developed by AdaCore. GNATtest is a unit test harness generation tool for Ada, generating for a given project a test harness based on the AUnit unit test library. GNATCoverage is a structural coverage analysis tool for Ada, C and C++, compatible with the most stringent civil avionic standards (DO-178c). GNATfuzz is an AFL++ based high level fuzzing tool for Ada, simplifying the fuzzing process by automatically generating a test harness, generating a starting corpus and running the fuzz test campaign.

## 3 Type-based test value generation

### 3.1 Library for type introspection

The first approach to generating values relies on the detailed type system available in Ada. The rich typing system can be used to express various invariants of a subprogram, but they can also be used as constraints for semi-random value generation. The solution being developed first extracts, using the Libadalang code analysis library, a high level representation of the types of all the parameters of the subprogram under test (SUT), including bounds for scalar types, and the possible shapes a record can take depending on the values of its discriminants expressed in each of the variant choices.

### 3.2 Generation of values

The type representation is then used to generate values within the type constraints, with various strategies the user will be able to configure. The strategies currently implemented in the prototypes include a uniformly distributed random generation for scalar types, unless they are used as parameters for strategies for compound types. For arrays, the size of the array is currently randomly generated between 0 and 10[1] elements to avoid generating objects too large to be allocated on the stack but this value and record types use biased strategies to generate discriminant values in order to explore all the shapes of the record. For instance, consider the discriminated record type in Figure 2.

```
type My_Record (I : Integer) is record
   case I is
      when 0 =>
         Comp : Integer;
      when others =>
         null;
   end case;
end record
```

**Figure 2: Example record with unbalanced variant**

If we were to do random generation, the odds of producing an instance with the Comp component would be too low. We thus coerce the strategy generation for I to pick in one of the value samples ({O} or the rest) or to be random (to keep some uniformity).

### 3.3 Challenges

The main challenge with this approach is that it is generally not possible to fully determine the characteristics of a type through a single approach. A static analysis of the code, with the help of libadalang, will allow to determine for a record, which are the components of the record depending on the values of the discriminants, but it won't make it possible to determine the bounds of defined scalar type whose bound are not statically known. Conversely, while it is possible to determine the bounds of scalar at runtime with the `'First` and `'Last` attributes, Ada lacks dynamic introspection capabilities, making it impossible to determine the list of components for a given record object. The types in Figure 3 illustrate such a case: It is impossible to determine the list of components of Rec_Type at runtime, but we cannot statically determine the bounds of Component_Type. This was solved by

[1]The choice for this value has not undergone extensive testing, and was introduced to aid the usability of the first prototypes. Finer strategy customization later on will allow the user to set this bound to a more meaningful value

```
function From_Env return Positive;

type Component_Type is range 0 .. From_Env;
type Rec_Type is record
   X : Component_Type;
end record;
```

**Figure 3: An example of types with non-static bounds**

```
Component_Type_Repr : constant Integer_Repr :=
  (Name => "Component_Type",
   First  => Component_Typ'First,
    Last => Component_Type'Last);

Rec_Type_Repr : constant Record_Type_Repr :=
  (Name => "Rec_Type",
   Components =>
     (1  => (Name => "X", Typ => Component_Type_Repr)));
```

**Figure 4: An example of representation for types with non-static bounds**

generating a partial type representation (namely the nature of the type, array index types, record discriminant and component types as well as variant part information) as an Ada support library, which when executed extracts the missing information (scalar type bounds), which can then be used to generate relevant values. A pseudo-code example of what would be generated for the types in Figure 3 is represented in Figure 4.

This dynamic introspection allows us to implement builtin generic strategies that will work for a variety of types (e.g. the same random strategy can be used for all of the record types). Such strategies need to generate values that can be of a variety of types, thus in a format generic enough to represent all kind of Ada values. The format picked there was the JSON format, as it has a structural advantage over e.g. a string format. The generated support library also contains JSON serializers (converting from JSON to the Ada type and the opposite) to be able to use the values that the generic strategies generate. An alternative could have been to generate specialized generators (that would return a value of the Ada type) but this was deemed much less convenient as it makes it harder to extend the built-in generators, and make code generation harder as the generators contain some intelligence.

This "two-stage" generation first analyzes the SUT and generates a support library and then execute generators from this support library to produce testcases values. It will allow the user to add custom generation strategies for subprogram parameters for which specific invariants are not captured in the type definition, in the form of an Ada subprogram. These strategies would be encoded as aspects, attached to a particular subprogram to drive its testing, or to a type, to provide a default generation strategy, should the tool-provided ones not fully match the intended testing pattern.

An example would correspond to Figure 5. This syntax, which is still being refined, is inspired by property-based testing tools such as Hypothesis [9].

```
function Generate_Integer return Integer;

function Foo (I, J : Integer) return Boolean
   with Generation => (Strategies => (I => Generate_Integer)
                      Nb_Tests => 10);
```

**Figure 5: Example of custom strategy specification**

Here, we specify for the parameter I a dedicated custom strategy using a function implemented by the user. The strategy of the parameter J is left unspecified, meaning that our tool will pick the default (random) strategy. As random generation can yield an arbitrary number of values, the number of tests would be configurable through the Generation.Nb_Tests aspect.

One could also use a built-in strategy but that is not the one used by default, e.g. the Builtin_Sample strategy which picks an arbitrary value from a sample, as seen in Figure 6.

```
function Foo (I, J : Integer) return Boolean
   with Generation =>
      (Strategies => (I => Builtin_Sample ([1, 2, 3]))
       Nb_Tests => 10);
```

**Figure 6: Example of strategy specification**

Another challenge we face when generating values for complex record types, or subprograms with a high number of parameters, is to handle the combination of all the values generated for the various parameters and components. We will be implementing various combinatorial strategies, in addition to random strategies (which are the default in the current implementation) to let the user choose how the individual generated values need to be combined, such as N-way testing.

One could imagine enhancing the previous strategy specification syntax to account for enumerative strategies as in Figure 7:

```
procedure Initialize (S : in out State);
function Has_Next (S : State) return Boolean;
function Generate_Integer
   (S       : in out State;
    Success : out Boolean) return Integer;

function Foo (I, J : Integer) return Boolean
   with Generation => (Strategies => (I => Generate_Integer),
                       Driving    => Enumerate);
```

**Figure 7: Example of custom enumerative strategy specification**

A custom enumerative strategy would need a state (to keep track of what has been generated before), and coercing the generation engine to enumerate the values produced by the (enumerative) strategies instead of producing a fixed number of tests, which is what the aspect Generation.Driving would specify.

## 4 Using subprogram specifications to their full extent

The generated test corpus will aim, through the various strategies available, to generate relevant inputs for the SUT, but some of the inputs generated solely based on the types of the input parameters may not be in conformance with the full specifications of the SUT.

Ada has first class support for executable precondition and postconditions, which take the form of a Boolean expression, such as in Figure 8, and are evaluated on each call of the subprogram to which they are attached.

Taking advantage of the executable specifications that can be attached to Ada subprograms, the type based generation tool will extract simple relational constraints on the input parameters to be taken into account during generation, filter

```
function Swap (Arr : Array_Type; I, J : Index_Type) with
   Pre  => I in Arr'First .. Arr'Last
             and then J in Arr'First .. Arr'Last
   Post => Arr (I) = Arr'Old (J)
             and then Arr (J) = Arr'Old (I);
```

**Figure 8: Example of preconditions and postconditions**

input values not validating the precondition, and use the post-condition to detect abnormal executions during the generated executable tests.

### 4.1 Measuring Contract coverage

Simply validating a post-condition does not however indicate if all the outcomes specified in the subprogram contracts are tested or not, in particular if the post-condition is a complex expression.

Contract coverage analysis performed by the tool GNATcov already makes use of three levels of detail for the coverage analysis of Ada code. For decisions, they heavily rely on checking that Boolean expressions have been evaluated to both True and False. However, contracts are expected to never evaluated to False, making the regular criteria unfit to be used for checking the proper coverage of contracts.

We are thus developing new coverage criteria, based on the proposals by C. Comar et al [12], so that the tool will be able to assess quality of the generated tests.

In keeping with the possibility to choose between three levels of requirements to achieve coverage of Ada code, contract coverage would be achieved according to three levels of strictness:

- Assertion True Coverage (ATC): The expression as a whole has been evaluated True at least once.
- Assertion True Condition Coverage (ATCC): All the expression conditions have been evaluated at least once as part of a complete expression evaluation to True. Different conditions may have been evaluated as part of different outer expression evaluation instances.
- Assertion True Path Coverage (ATPC): All the paths leading to a True outcome within the expression's BDD were taken.

## 5 Generating values through fuzzing

While the values generated from the structural analysis of the SUT parameter's type may generate a highly valuable starting corpus, its generation does not take into account the implementation of the SUT. As such, critical values on which particular behaviors could be observed may not be represented in the corpus. A fuzzer is a powerful tool which can find vulnerabilities in subprograms it operates on by mutating the input values of the SUT, but it can also be used to explore more paths in a subprogram, and thus help build a quality test corpus which maximizes structural coverage. Fuzzers however rely on a pre-existing test corpus to operate, and the higher the structural coverage is attained by the starting corpus, the less mutations are required to find the remaining unexplored execution paths.

### 5.1 Efficiently fuzzing Ada values

We are integrating our test generation tool with GNATfuzz, a tool developed around AFL++ [13] for Ada code. Since there currently are no custom AFL++ mutators for Ada values, the contents of the inputs to the SUT must be serialized in a file, which will undergo the mutations strategies in AFL++, then these variables need to be read from disk, and passed to the SUT to check if new paths or vulnerabilities are found with the mutated values.

AFL++ operates by flipping bits of the binary representation of the input values for the SUT. As there are currently no structure-aware mutators implemented for Ada, these bit flips may happen on alignment padding bits. These bit-flips are thus useless, and render the fuzzing process less efficient. To solve this, we have developed a binary representation for Ada values that aims at minimizing the number of padding bits.

Moreover, as an Ada variable may encode some structural information, such as array length or discriminant values in the binary representation for that variable, it is important to adapt what part of the binary representation of a value can be mutated during the fuzzing process, and which should remain unchanged. If the length of an array is mutated, and grows larger, it won't be possible to read back the added component values, as they do not exist. We have thus split the binary representation for Ada values, separating the immutable components from the mutable ones, to be used in GNATfuzz to ensure only the relevant components are mutated during a fuzzing campaign.

### 5.2 Using symbolic execution to find edge-cases

Structure-aware values generation and fuzzing combined are not always enough to explore all the execution paths of the SUT. Symbolic execution is a third method that can help find input values for the SUT that will reach previously unexplored execution paths. GNATFuzz integrates SymCC [7], a compiler-based symbolic execution tool that will be used in conjunction with the AFL++ fuzzing loop, when the later cannot find input values that explore new paths. Given the costly nature of a symbolic execution run of the SUT, this alternating use of the AFL++ loop and of SymCC help minimize the number of symbolic execution runs, while still exploring all the relevant execution paths.

As Ada contracts can contain any valid Boolean expression, including calls to other subprograms, arbitrary code can be executed as part of the evaluation of a pre-condition or a post-condition. It is thus not always possible to extract simple relations between the various formals of the SUT. However, given the executable nature of contracts in Ada, and the compiler-based nature of SymCC, it can be used to explore branches that have not been executed in the precondition and postcondition to generate input values of higher interest, that will increase coverage for the various Assertion criteria.

## 6 Consolidating the test-cases in a coherent test-suite

Each value generation phase is handled by a specific tool, which requires communicating test case values amongst each other. To do so, we are defining two test case exchange

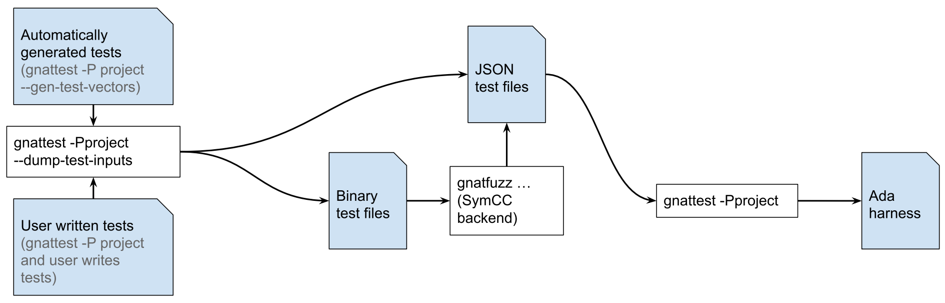

Automatically generated tests (gnattest -P project --gen-test-vectors)
gnattest -Pproject --dump-test-inputs
User written tests (gnattest -P project and user writes tests)
JSON test files
Binary test files
gnatfuzz … (SymCC backend)
gnattest -Pproject
Ada harness

**Figure 9: Integration workflow among the various tools**

format. The first one is in plain text so it can be reviewed by any human, which serves as the exchange medium for the tools. The second one is a binary dump of the input data, as presented in Section 5. The type based generation creates tests cases in the text based format, which can then be either incorporated into a test harness generated by GNATtest, and then executed. The test cases are also passed to the fuzzer as a starting corpus in the form of binary input dumps. The tests of interest generated by the fuzzer are themselves converted to the text format to be inserted back into the test harness. If GNATfuzz is used in conjunction with SymCC, the values generated by it are also fed back into the source test harness for inspection by the user.

This test exchange format can also be used to extract test values from existing GNATtest harnesses, by instrumenting the SUT to convert its input parameter to the format, which can then be used in the fuzzing phase and symbolic execution phase to augment the pre-existing testsuite. Figure 9 gives an overview of the interactions of the tools and how they each contribute to the testsuite generation.

Many of the tests generated by the three approaches combined may be redundant, which is why a testsuite minimisation tool is being worked on, keeping only the test-cases yielding unexpected behaviors, and measuring the impact on coverage on the SUT to find the minimal amount of tests that achieve the highest coverage possible.

## 7 Conclusion

The tools presented in this article have the potential to significantly help reduce the testing efforts for Ada subprograms, by both introducing new ways to generate input values based on the strong typing system present in Ada, and by combining multiple dynamic analysis techniques to create a high quality and high coverage consolidated test-suite, either starting from a pre-existing GNATtest harness, or from the type of the parameters of the subprogram under test, as well as its specifications.

Future work may include lifting the limitations concerning tagged types, and investigate the possibility to use abstract interpretation tools to generate input sets that would satisfy the precondition of the SUT.

## References

[1] Midoan Software Engineering Solutions, “Mika: Test data generation for ada.” http://www.midoan.com/mika.html.

[2] QA systems, “Adatest95: Automated unit & integration testing for ada.” https://www.qa-systems.com/tools/adatest-95/.

[3] IBM, “Real-time rational-testing.” https://www.ibm.com/docs/en/rtr/8.0.2?topic=rational-test-realtime-overview.

[4] Direction Générale de l’armement, “RAPID homepage.” https://www.defense.gouv.fr/aid/deposez-votre-projet/rapid-regime-dappui-a-linnovation-duale.

[5] C. Cadar, D. Dunbar, and D. R. Engler, “Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs.,” in *OSDI* (R. Draves and R. van Renesse, eds.), pp. 209–224, USENIX Association, 2008.

[6] K. Khazem and M. Tautschnig, “Cbmc path: A symbolic execution retrofit of the c bounded model checker,” in *Tools and Algorithms for the Construction and Analysis of Systems* (D. Beyer, M. Huisman, F. Kordon, and B. Steffen, eds.), (Cham), pp. 199–203, Springer International Publishing, 2019.

[7] S. Poeplau and A. Francillon, “Symbolic execution with SymCC: Don’t interpret, compile!,” in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 181–198, USENIX Association, Aug. 2020. https://www.usenix.org/conference/usenixsecurity20/presentation/poeplau.

[8] K. Claessen and J. Hughes, “Quickcheck: A lightweight tool for random testing of haskell programs,” *SIGPLAN Not.*, vol. 46, p. 53–64, may 2011.

[9] D. R. MacIver, Z. Hatfield-Dodds, *et al.*, “Hypothesis: A new approach to property-based testing,” *Journal of Open Source Software*, vol. 4, no. 43, p. 1891, 2019.

[10] Proptest-Rs, “Proptest.” https://proptest-rs.github.io/proptest/intro.html.

[11] Zac Hatfield-Dodds, “Hypofuzz: adaptive fuzzing of property-based test suites.” https://zhd.dev/phd/hypofuzz.html.

[12] C. Comar, J. Guitton, O. Hainque, and T. Quinot, “Structural coverage criteria for executable assertions,” in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[13] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse, “AFL++: Combining incremental steps of fuzzing research,” in *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, USENIX Association, Aug. 2020.

# Mechanization of the Ravenscar Profile in Coq

***J. Hugues***
*Carnegie Mellon University/Software Engineering Institute; email: jhugues@andrew.cmu.edu*

## Abstract

*The Ravenscar profile has been added to the Ada language as part of the 2005 revision. It is a pragmatic definition of concurrency patterns for real-time systems for mono-core processor. In this paper, we report on an ongoing effort to mechanize a toy language derived from IMP that embeds the Ravenscar profile constructs as they are specified in the Ada Reference Manual. We define the denotational semantics of the language and derive two interpreters for the language.*

*Keywords: Coq, mechanization, Ravenscar*

## 1 Introduction

The semantics of a programming language can be expressed in multiple ways using either structural, reduction, or natural semantics, see [1] for a survey of the various styles. Defining the semantics of sequential programming languages is a well-mastered exercise, with numerous variations for imperative or functional programming languages. They usually focus on the type system. Process algebras complement these approaches to express concurrency in programming languages [2]. These approaches are usually out of touch of actual programming languages that implement a model of concurrency.

The Ada programming language has been one of the first languages with built-in concurrency features. Ada builds on the notion of task, protected object (akin to the Hoare monitor, see §9 in [3] for a discussion). This model of concurrency can be restricted to meet the constraints of high-integrity systems, for instance the Ravenscar profile, introduced in Ada 2005.

In [4], the authors propose an operational semantics for the Ada Ravenscar profile. This fundational work provides only definitions, but no associated proof. Being a paper exercise, its consistency and correctness is not asserted.

In this paper, we report on an ongoing effort to mechanize the Ada Ravenscar profile using the Coq theorem prover. Our contributions cover 1) the definition of a toy imperative language to mimic the key constructs of Ada concurrent languages for control flows and concurrency, 2) the definition of its semantics using an axiomatic and a reduction-style semantics, 3) the definition of a simulator for this language, and a proof of its correctness. Finally, we propose 4) a simulator of multiple Ravenscar programs, each representing a thread.

This paper is organized as follows: in section 2, we introduce the Ravenscar profile; section 3 introduces the Coq theorem prover. In section 4, we first introduce the IMP language, and then its extension to support Ravenscar constructs for concurrency (section 5), and then conclude.

## 2 The Ada Ravenscar profile

The Ada Ravenscar profile is a normative subset of the Ada 2005 language [5]. This profile defines guidelines to implement deterministic real-time systems. Ravenscar restricts a system's constructs to periodic and sporadic tasks that interact through one-way asynchronous communications. This model of computation restricts constructs to those that allow for determinism, scheduling analysis, and memory boundedness. It has been initially defined for the Ada language and later adapted to other languages and APIs.

This profile is organized into four categories:

- *Static Existence Model* are restrictions that guarantee the system is comprised of a set of tasks with static properties. This ensures some properties of the task set are invariant and enable scheduling analysis using classical Liu and Layland or Joseph & Pandia schedulability test [6]. The supported task set supports only sporadic or periodic tasks;
- *Static Synchronization and Communication Model* defines the resources to support the task set (activation based on the arrival of the event) and shared resources. This model uses a static set of Ada protected objects that use the Immediate Ceiling Priority Protocol.
- *Deterministic Memory Usage* restrictions disallow implicit memory allocation by the Ada runtime. Additional Ada High-Integrity restrictions may forbid the secondary stacks or all forms of dynamic memory allocation.
- *Deterministic Execution Model* further restricts Ada constructs to fully adhere to the task model of the Response-Time Analysis and forbid situations that would create a synchronization point with a complex dispatch trigger such as the conjunction of multiple input events.

## 3 The Coq Theorem Prover

Coq [7] is a proof assistant, or Interactive Theorem Prover (ITP). A developer may use the Coq language, Gallina, to write mathematical definitions, executable algorithms, and theorems within an Interactive Development Environment (IDE). Coq has been used to prove non-trivial mathematical theorems and also develop formally certified software and hardware. One interesting feature of many ITPs is the capability to generate certified code (e.g. OCaml or Haskell) from Coq definitions. In this context, "certified" means that there exists a proof script – the certificate – that connects the software produced and the proofs that accompany it. Many references exist to learn more about the Coq ITP such as

Pierse et al. [8]. In the following, we introduce a minimal set of notions prior to introducing our Coq development.

Coq relies on Gallina, a functional language that acts as a specification language to describe types, functions, and proofs of some statement. Coq's core follows the rules of the Calculus of Inductive Constructions. The basic types in Coq are either propositions, `Prop`, or sets `Set`. All other types are built on top of these two types. `Prop` is the type of logical propositions. It denotes the class of terms representing proofs. `Set` represents typical data types.

In Figure 1, we show some fragments of Coq. First, we define the `statements` that is the list of statements we use in our mechanization (see section 4). Then, we define the `Legal_Periodic_Body_Spec` lemma that assesses that the function `Legal_Periodic_Body` is decidable. The syntax uses sumbool. A sumbool, which is written $A + B$, is the informative disjunction "A or B", where A and B are logical propositions. The script proof asserts that we can either build a proof of A or a proof of B. The proof is done by induction overs the program p and proceeds by simplification and basic reasoning steps on the terms produced using Coq's default proof tactics.

```
Inductive statements : Type :=
  (* A sequential execution step *)
  | COMP (WCET : Time)
  (* Delay until some time *)
  | DELAY_UNTIL_NEXT_PERIOD
  (* Sequence of statements *)
  | SEQ (s1: statements) (s2: statements)

Lemma Legal_Periodic_Body_dec: forall p,
  { Legal_Periodic_Body p } +
    {   Legal_Periodic_Body p }.
Proof.
  induction p ; simpl ; auto.
  destruct p1 ; auto ;
  apply dec_sumbool_and ; auto.
Qed.
```

**Figure 1: A Coq inductive type**

Coq has a rich standard library that defines regular types such as booleans, integers, lists, and typical results on them. Coq type system support polymorphisms and Java-like interfaces (typeclasses). Because `Prop` and `Set` are both types, they can be part of expressions. One of the key aspects of Coq is that a proposition is more powerful than a boolean expression. The type `bool` is computational: one can define functions or perform case analysis. On the other hand, the type `Prop` supports universal quantification over elements, that is typical exists ($\exists$) and for all ($\forall$) quantifiers.

Writing proofs can be a repetitive task. Coq provides a rich library of tactics that encode specific reasoning steps. A typical example is `auto` that will prove basic propositions by applying well-known facts and computations. Coq's set of tactics can be extended by the user using the LTAC language.

## 4 The IMP language

In [4], the authors propose an operational semantics for the Ada Ravenscar profile. Yet, the authors opted to abstract the Ada language and define Ada concurrency constructs using concepts such as synchronizers and exchangers that are runtime abstraction used in their implementation of a Ravenscar runtime. Also, their work provides only definitions, but no proof of consistency or correctness of their approach.

In the following, we opted to extend a well-studied imperative language called IMP and extend it with Ada Ravenscar constructs for concurrency in section 5.

### 4.1 Sequential IMP

IMP is a toy language used to represent a basic imperative programming language. It is presented in depth in [9]. We present here the key elements of the language. There are three types of statements in IMP:

- arithmetic expressions AEXP (denoted by $a, a_0, a_1, \ldots$)
- Boolean expressions BEXP (denoted by $b, b_0, b_1, \ldots$)
- commands COM (denoted by $c, c_0, c_1, \ldots$)

A program in the IMP language is a command in COM.

Let VAR be a countable set of variables. Elements of VAR are $x, x_0, x_1 \ldots$. Let $n, n_0, n_1, \ldots$ be integers and $\overline{n}$ be an integer constant symbol representing the number $n$. The BNF grammar for IMP is

$$\begin{array}{rcl}
\textsc{AExp} & ::= & \overline{n} \mid x \mid (a_0 \oplus a_1) \\
\textsc{BExp} & ::= & \textbf{TRUE} \mid \textbf{FALSE} \mid (a_0 \odot a_1) \mid (b_0 \oslash b_1) \mid (\neg b) \\
\textsc{Com} & ::= & \textbf{SKIP} \mid x := a \mid (c_0 \; ; \; c_1) \mid \\
 & & (\textbf{IF}\ b\ \textbf{THEN}\ c_1\ \textbf{ELSE}\ c_2) \mid (\textbf{WHILE}\ b\ \textbf{DO}\ c\,) \\
\oplus & ::= & + \mid * \mid - \mid / \\
\odot & ::= & \leq \mid = \\
\oslash & ::= & \vee \mid \wedge
\end{array}$$

It relies on common notations for arithmetic and boolean operators, assignments, and control flows.

From this BNF, we can then define the notion of *configuration* of a program. A *configuration* is a pair $< c, \sigma >$, where $c \in Com$ is a command and $\sigma$ is a store. A *store* (also known as a *state*) is a function $Var \to \mathbb{Z}$ that assigns an integer to each variable. The set of all stores is denoted $\Sigma$.

Intuitively, the configuration $< c, \sigma >$ represents an instantaneous snapshot of reality during a computation, in which $\sigma$ represents the current values of the variables, and $c$ represents the next command to be executed.

As defined, IMP is a low-level language, close to an assembly language, with registers being substituted with more general variables. We claim it is suitable for expressing a language like Ada: an Ada program is turned into a simpler form with a similar power of expression as part of the compilation process.

### 4.2 Structural Operational Semantics of IMP

Small-step semantics specifies the operation of a program one step at a time. There is a set of rules that we continue to apply to configurations until reaching a final configuration $<$ **SKIP**$, \sigma >$ (if ever). We write $< c, \sigma > \to < c', \sigma' >$ to indicate that the configuration $< c, \sigma >$ reduces to $< c', \sigma' >$ in one step, and we write $< c, \sigma > \overset{*}{\to} < c', \sigma' >$ to indicate that $< c, \sigma >$ reduces to $< c', \sigma' >$ in zero or more steps. Thus $< c, \sigma > \overset{*}{\to} < c', \sigma' >$ iff there is a $k \geq 0$ and configurations $< c_0, \sigma_0 >, \ldots, < c_k, \sigma_k >$ such that $< c, \sigma > = < c_0, \sigma_0 >$, $< c', \sigma' > = < c_k, \sigma_k >$, and $< c_i, \sigma_i > \to < c_{i+1}, \sigma_{i+1} >$ for $0 \leq i \leq k-1$.

We define auxiliary small-step operators $\to_a$ and $\to_b$ for arithmetic and Boolean expressions, respectively, as well as $\to$ for commands. The types of these operators are

$$\begin{aligned} \to \quad &: \quad (Com \times \Sigma) \to (Com \times \Sigma) \\ \to_a \quad &: \quad (AExp \times \Sigma) \to \mathbb{Z} \\ \to_b \quad &: \quad (BExp \times \Sigma) \to \mathbf{B} \end{aligned}$$

One can derive the following rules for evaluating expressions.

- Constants: $\overline{< \overline{n}, \sigma > \to_a n}$
- Variables: $\overline{< x, \sigma > \to_a \sigma(x)}$
- Operations: $\dfrac{< a_0, \sigma > \to_a n_0 \quad < a_1, \sigma > \to_a n_1}{< a_0 \oplus a_1, \sigma > \to_a n_0 \oplus n_1}$

The rules for evaluating Boolean expressions and comparison operators are similar.

The definition of the semantics of commands follows a similar approach. First, we define an operator for updating a store:let $\sigma[n/x]$ denote the store that is identical to $\sigma$ except possibly for the value of $x$, which is $n$.

$$\sigma[n/x](y) \quad \triangleq \quad \begin{cases} \sigma(y), & \text{if } y \neq x, \\ n, & \text{if } y = x. \end{cases}$$

The semantics of all commands is presented in Figure 2. Note that there is no rule for **SKIP**, since $<$ **SKIP**$, \sigma >$ is a final configuration. This closes the definition of IMP.

## 5 IMP-Ravenscar

An Ada program is defined by three sets: $\{\mathcal{S}, \mathcal{T}, \mathcal{P}\}$ representing the sets of subprograms, threads, and protected objects[1]. An Ada task $\mathcal{T}_i \in \mathcal{T}$ is defined by the subprogram $\mathcal{S}_i$ it executes and real-time parameters: its runtime state (IDLE, RUNNING, or READY), its priority and its period. The latter is not an explicit parameter in the Ada programming language but is used for, e.g. scheduling analysis or simulation. Similarly, an Ada protected object $\mathcal{P}_i$ is defined by the set of subprograms from $\mathcal{S}$ it can execute and a unique entry, a subprogram whose execution is guarded by a boolean expression and a priority value for the Immediate Ceiling Priority Protocol concurrency policy[2]. Note that the Ada Ravenscar profile does not allow for dynamic allocation of tasks or protected objects. Similarly, Ada tasks are not allowed to terminate.

[1]We omit elements like global variables or interrupt handlers for clarity.

[2]i.e. when a thread enters the protected object, its priority is immediately raised to this value

### 5.1 Semantics of IMP-Ravenscar

Concurrency in the Ada Ravenscar profile relies on two patterns. First, absolute delays to suspend the execution of a task until a future date is reached. Second, communication through a protected object function or entry. We extend the set of commands accordingly.

$$\begin{aligned} \text{COM'} \quad ::= \quad & \textbf{DELAY\_UNTIL} \mid \textbf{PO\_ENTRY) (e)} \\ & \mid \textbf{PO\_FUNCTION) (f)} \mid \text{COM} \end{aligned}$$

The state of an Ada thread (T) is defined by a command denoting the subprogram it executes, its store used for local variables ($\sigma$), the time for the next dispatch of a thread (ND). We define in the figure 3 the rules for these commands. By convention, $\mathcal{T}[a ::= < expr >]$ indicates that member a of T is updated by the value of expr, $\mathcal{P}[a]$ fetches the value.

The semantics follows the description of the Ada language: for *delay until*, the next dispatch time of the task is updated by the value of period and the task becomes idle. The execution of a protected function can be done immediately, the priority of the thread is updated to the ceiling priority. This is made possible thanks to the simplification of the Ravenscar profile [10]. Protected entries follow a similar approach: if the guard evaluates to true, the execution proceeds, otherwise the thread becomes IDLE.

### 5.2 Validity of IMP-Ravenscar program

The previous rules defined the semantics of Ada Ravenscar programs in basic mathematical terms. They have a direct translation in Coq. We sketch some of the results we proved[3]: IMP-Ravenscar is deterministic, one can assert a program terminates or not. We also defined the notion of valid periodic and sporadic programs as decidable properties. Per the Ravenscar profile, a task never terminates. A periodic thread executes an infinite loop. It can be suspended only by one delay until per cycle. A sporadic thread also executes an infinite loop, it enforces a minimum inter-arrival time (period) between two cycles and waits on one protected object entry to receive events from another thread.

In addition, IMP-Ravenscar inherits properties from the IMP language: it is defined by both a denotational and a reduction semantics. We defined an evaluation function for a program, allowing to compute its final state. We also implemented a continuation-passing style semantics (CPS) that is equivalent to the initial semantics. In CPS, the command to be executed is explicitly decomposed into a sub-command under focus, where computation takes place; a context that describes the position of the sub-command in the whole command; or, equivalently, a continuation that describes the parts of the whole command that remain to execute once the sub-command terminates. This allows us to execute a subprogram for $x$ steps. In other words, to stop or preempt the execution of a program. All proofs are technical, but follow the same pattern of a proof by induction on either the command to be executed or the number of steps to be computed.

[3]The full mechanization is available at `https://github.com/Oqarina/oqarina/blob/main/src/MoC/ravenscar.v`

- Assignments: $\dfrac{< a, \sigma > \rightarrow_a n}{< x := a, \sigma > \rightarrow < \textbf{SKIP}, \sigma[n/x] >}$
- Sequences: $\dfrac{< c_0, \sigma > \rightarrow < c_0', \sigma' >}{< c_0; c_1, \sigma > \rightarrow < c_0'; c_1, \sigma' >}$ $\dfrac{}{< \textbf{SKIP}; c_1, \sigma > \rightarrow < c_1, \sigma >}$
- Conditionals: $\dfrac{< b, \sigma > \rightarrow_b true}{< \textbf{IF } b \textbf{ THEN } c_0 \textbf{ ELSE } c_1, \sigma > \rightarrow < c_0, \sigma >}$ $\dfrac{< b, \sigma > \rightarrow_b false}{< \textbf{IF } b \textbf{ THEN } c_0 \textbf{ ELSE } c_1, \sigma > \rightarrow < c_1, \sigma >}$
- While statements: $\dfrac{}{< \textbf{WHILE } b \textbf{ DO } c, \sigma > \rightarrow < \textbf{IF } b \textbf{ THEN } (c; \textbf{WHILE } b \textbf{ DO } c) \textbf{ ELSE SKIP}, \sigma >}$

**Figure 2: Small-step semantics for IMP commands**

- Delay: $\dfrac{}{< \textbf{DELAY\_UNTIL}, \mathcal{T} > \rightarrow < \textbf{SKIP}, \mathcal{T}[nd ::= nd + period, state := IDLE] >}$
- PO_Function: $\dfrac{}{< \textbf{PO\_FUNCTION (f)}, \mathcal{T}, \mathcal{P} > \rightarrow < f, \mathcal{T}[priority ::= \mathcal{P}[priority]] >}$
- PO_Entry: $\dfrac{\mathcal{P}[guard] \rightarrow_b true}{< \textbf{PO\_ENTRY (f)}, \mathcal{T}, \mathcal{P} > \rightarrow < f, \mathcal{T}[priority ::= \mathcal{P}[priority]] >}$ $\dfrac{\mathcal{P}[guard] \rightarrow_b false}{< \textbf{PO\_ENTRY (f)}, \mathcal{T}, \mathcal{P} > \rightarrow < f, \mathcal{T}[state := IDLE] >}$

**Figure 3: Small-step semantics for Ravenscar commands**

### 5.3 Simulation of IMP-Ravenscar program

The previous definitions allow one to reason on the execution of one program. We implemented a simulation procedure for a Ravenscar system that leverages the CPS-style semantics. We are concerned with mono-core systems, therefore the simulation procedure is simple. We first elect the next thread to be executed and compute the time of dispatch of the next thread that may preempt it (e.g. because of a higher-priority thread awaking). We then simulate this thread for the corresponding amount of time. We repeat the process until we reach the end of the simulation.

## 6 Conclusion

In this paper, we sketched how to mechanize the Ada Ravenscar profile by extending IMP, a canonical imperative language. Although Ada has a richer semantics, its control flow mechanisms can be expressed in both denotational and reduction semantics, allowing one to derive a simulator that conforms by construction to this semantics. This work is the first step in fully defining the semantics of the Ravenscar profile in the Coq theorem prover. We plan to finalize this effort and demonstrate other canonical results like the absence of deadlock that comes from the usage of ICPP and mono-core systems; we will also evaluate the connection with formal scheduling analysis tools like PROSA.

## Acknowledgments

Copyright 2023 ACM. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.
DM23-0191

## References

[1] Y. Zhang and B. Xu, "A survey of semantic description frameworks for programming languages," *SIGPLAN Not.*, vol. 39, pp. 14–30, Mar. 2004.

[2] J. Baeten, "A brief history of process algebra," *Theoretical Computer Science*, vol. 335, no. 2, pp. 131–146, 2005. Process Algebra.

[3] *Ada 95 Rationale The Language - The Standard Libraries*. Lecture Notes in Computer Science, 1247, Berlin, Heidelberg: Springer Berlin Heidelberg, 1st ed. 1995. ed., 1995.

[4] I. Hamid and E. Najm, "Operational semantics of ada ravenscar," in *Reliable Software Technologies – Ada-Europe 2008* (F. Kordon and T. Vardanega, eds.), (Berlin, Heidelberg), pp. 44–58, Springer Berlin Heidelberg, 2008.

[5] B. Dobbing, A. Burns, and T. Vardanega, "Guide for the use of the of the Ravenscar Profile in High Integrity Systems," tech. rep., 2003.

[6] M. Chetto, ed., *Real-Time Systems Scheduling 1: Fundamentals*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Aug. 2014.

[7] Y. Bertot and P. Castéran, *Interactive theorem proving and program development - Coq'Art: The calculus of inductive constructions*. Texts in theoretical computer science. An EATCS series, Springer, 2004.

[8] B. C. Pierce, A. Azevedo de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, and B. Yorgey, *Logical Foundations*. Software Foundations series, volume 1, Electronic textbook, May 2018.

[9] T. Nipkow and G. Klein, *Concrete Semantics - With Isabelle/HOL*. Springer, 2014.

[10] J. A. de la Puente, J. F. Ruiz, and J. Zamorano, "An open ravenscar real-time kernel for GNAT," in *Reliable Software Technologies - Ada-Europe 2000, 5th Ada-Europe International Conference, Potsdam, Germany, June 26-30, 2000, Proceedings* (H. B. Keller and E. Plödereder, eds.), vol. 1845 of *Lecture Notes in Computer Science*, pp. 5–15, Springer, 2000.

# Worst Case Execution Time and Power Estimation of Multicore and GPU Software: A Pedestrian Detection Use Case

***Ivan Rodriguez Ferrandez, Alvaro Jover Alvarez, Matina Maria Trompouki, Leonidas Kosmidis, Francisco J. Cazorla***

*Barcelona Supercomputing Center (BSC) and Universitat Politecnica de Catalunya (UPC); email: {ivan.rodriguez, alvaro.jover, matina.trompouki, leonidas.kosmidis, francisco.cazorla}@bsc.es*

## Abstract

*Worst Case Execution Time estimation of software running on parallel platforms is a challenging task, due to resource interference of other tasks and the complexity of the underlying CPU and GPU hardware architectures. Similarly, the increased complexity of the hardware, challenges the estimation of worst case power consumption. In this paper, we employ Measurement Based Probabilistic Timing Analysis (MBPTA), which is capable of managing complex architectures such as multicores. We enable its use by software randomisation, which we show for the first time that is also possible on GPUs. We demonstrate our method on a pedestrian detection use case on an embedded multicore and GPU platform for the automotive domain, the NVIDIA Xavier. Moreover, we extend our measurement based probabilistic method in order to predict the worst case power consumption of the software on the same platform.*

## 1 Introduction

Applications with real-time requirements such as the ones used in automotive and other safety critical systems, require the computation of worst case execution time (WCET). However, the increased complexity of modern platforms targeting these domains, which includes multiple cores and GPUs, makes the computation of the WCET a very challenging task.

Conventional timing analysis methods such as static timing analysis require detailed information about the hardware architecture which are not available for these platforms, especially for GPUs. Similarly, traditional measurement-based methods (MBTA) are unable to deal with the uncertainty introduced by shared caches and GPUs. Measurement-Based Probabilistic Timing Analysis (MBPTA) [1] can address this complexity, allowing to compute WCET on multicores [2].

However, MBPTA requires certain properties to be present in the platform in order to be used, namely *independent and identically distributed (i.i.d)* execution times ensuring that execution conditions at analysis match the ones at system operation. These properties require either specially designed hardware [3] [4] or software randomisation [5] on Commercial-off-the-shelf (COTS) platforms. However, software randomisation methods have been only demonstrated so far on CPUs. In this work, we demonstrate for the first time that software randomisation is also applicable on GPUs, and in particular its static variant which works at source code level, known as Toolchain Agnostic Software Randomisation (TASA) [6].

In addition to the Worst Case Execution Time, safety critical systems require the estimation of their power consumption. Such an estimation is useful in order to guarantee the adequate thermal design of the system, so that it stays within its operational limits, otherwise there is a risk of unavailability of the safety critical system. However, the complexity of the hardware complicates this estimation using measurement based techniques [7], since during the experiments performance during analysis time, there are power measurement peaks which might not be observed.

Usually manufacturers provide a conservative upperbound of the worst case power consumption, known as Thermal Design Power (TDP), however this can result in overprovisioning of the system resources. The use of Extreme Value Theory (EVT) [8], which is one of the foundations of MBPTA, has been shown possible in the past for the worst case power consumption of a simulated time-randomised multicore CPU based on the NGMP [7]. Similar to MBPTA, hardware or software randomisation is required to enable the use of this method. In this work, TASA can provide this property.

In particular, we demonstrate our method on a pedestrian detection use case deployed on the automotive platform NVIDIA Xavier, showing that it is possible to obtain competitive WCET estimations with respect to traditional measurement based methods, as well as to estimate the worst case power consumption, too.

## 2 System Architecture

This work has been developed within the H2020 project UP2DATE [9], which aims to provide safe and secure updates on complex, heterogeneous platforms featuring multiple CPUs and including accelerators such as GPUs. The benchmarking of the available embedded high performance platforms early in the project, led to the selection of the NVIDIA

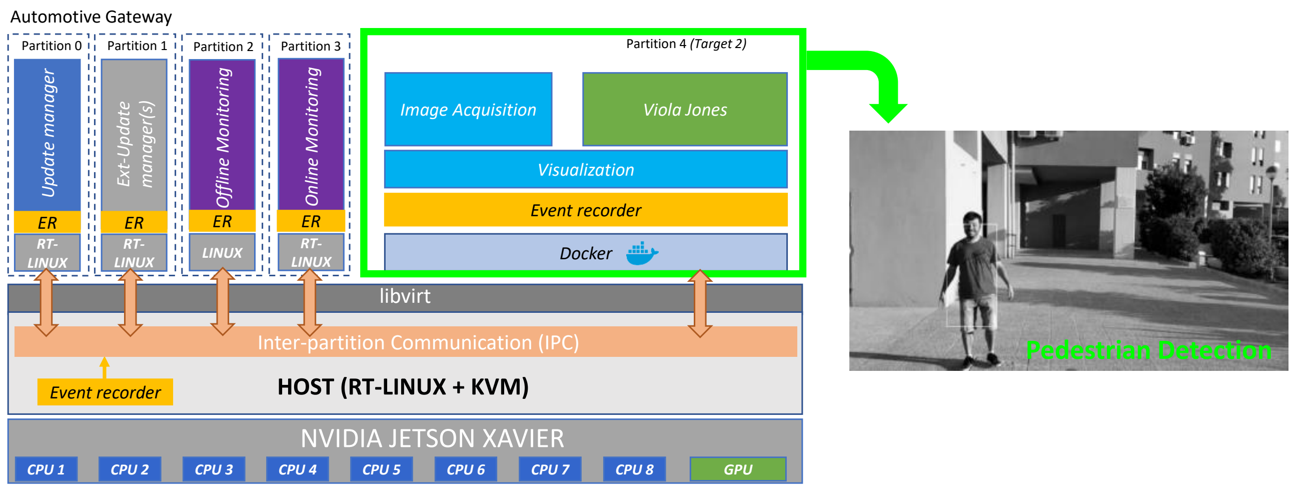


**Figure 1: UP2DATE System architecture based on the Xavier Automotive Gateway Platform, featuring multiple CPUs and a GPU.**

Xavier automotive platform [10]. This platform contains 8 ARM v8 compatible CPUs designed by NVIDIA, codenamed as Carmel. Each of the CPUs has private instruction and data caches, and shared L2 and L3 caches, connected to a 32GB LPDDR memory. Moreover, a GPU with 8 Streaming Multiprocessors (SMs) is connected to the LPDDR memory.

Since software updates are key for the project, it is important not only that to compute the WCET of safety-related software, but to remain also valid after an update. This led us to the following decisions regarding the system architecture. In order to facilitate software updates as well as time and space isolation required for the deployment of safety critical applications, the use of virtualisation has been decided [10] in order to create application partitions. For the deployment of CPU only partitions, the Linux KVM hypervisor has been selected, while for partitions requiring access to the GPU, the NVIDIA supported docker solution is used, since currently it is the only way that the GPU can be accessed from a virtualised environment on this platform. Figure 1 shows an overview of the system architecture considered in the UP2DATE project.

In order to simplify per-partition timing analysis, each partition is statically bound to a CPU and uses a single CPU. Similarly, in order to avoid the timing implications of GPU sharing, only a single partition is accessing the GPU. Overall, up to 8 partitions can be deployed on the platform, one for each CPU, and up to 1 of them is able to access the GPU. Given the large number of CPUs and the available memory of the platform (32GB), this approach is feasible. If the number of functionalities which need to be deployed are larger than the number of cores, they can be consolidated in one of the partitions.

## 3 Use Case Description

The UP2DATE project incorporates several industrial use cases from the railway and automotive domains [9]. However, none of them – which are based on legacy industrial applications deployed in existing single core safety critical platforms – had a high computational requirement needing a GPU. For this reason, we have developed a research use case which resembles a computationally intensive automotive task, such as pedestrian detection. Pedestrian detection is an important task needed for the implementation of Automatic Emergency Breaking functionality which since 2023 is required in all new cars sold in the European Union [11].

As a basis of the UP2DATE research use case we used a previously developed open source multi-CPU, multi-GPU pedestrian detection benchmark [12], based on Viola-Jones [13] [14]. While this classic machine vision algorithm has lower accuracy compared to modern implementations based on deep learning, it has the advantage of *explainability*, which is important for functional safety certification.

The main code changes are the code conversion to use only one CPU and GPU, and the instrumentation with the UP2DATE monitoring middleware, which allows to measure the execution time of the CPU and GPU tasks of the application, as well as the collection of performance counter measurements for both CPU and GPU. Moreover, the application has been enhanced with a new pedestrian data set, which has been used for project demos at dissemination events. The modified application is released also as open source at [15].

The application consists of 3 tasks. The first task performs the image acquisition. The second task processes the image in order to detect the pedestrians using the Viola-Jones method and returns the coordinates and the bounding boxes of the detections, which can be used as input by subsequent tasks e.g. for the implementation of the automatic emergency breaking functionality. Finally, the third task is responsible of displaying the detection, as well as saving the resulting image for logging purposes, e.g. for post accident examination for insurance or legal reasons.

## 4 GPU Software Randomisation

In order to enable the application of MBPTA on a conventional CPU and GPU platform, we apply software randomisation. Existing software randomisation methods [5] [6] are only available for CPUs. Moreover, dynamic software randomisation – the method with the highest Technology Readiness Level (TRL), demonstrated with several industrial use cases from the avionics [16] and the space [17] domain – is not available for ARM processors and more importantly, it is not possible to be implemented on GPUs. The reason is that dynamic software randomisation requires compiler modifications. However, GPU vendors and especially NVIDIA are notorious for using proprietary languages, such as CUDA, with closed source compiler and runtime systems, which prevent the implementation of such functionality.

Toolchain Agnostic Software Randomisation (TASA) [6], was developed to overcome this limitation, by performing software randomisation at source code level. TASA relies on the principle that compilers generate object code in the same order they appear in the source code. However, TASA was prototyped as a proof of concept, only for CPUs and demonstrated with small benchmarks. Moreover, it only supported ANSI C and had no support for the C preprocessor, which limited its capability to work with industrial grade code.

In this work we have reimplemented TASA from scratch within the CIL [18] source-to-source compiler framework. CIL is capable of parsing any C code, including gcc extensions, which makes it capable of working even with the Linux kernel, which is likely the most complex and largest C code base. This makes our TASA reimplementation, which we have open sourced at [19] to achieve the highest TRL on CPU. CIL provides features which are useful for the application of TASA such as inclusion of function prototypes as well as a linker at source code level, which allows to merge all application's source code files in a single source code file which can be randomised with TASA.

In order to enable GPU software randomisation, we extended CIL to support the CUDA programming language. CUDA is a C/C++ extension for GPU programming, therefore our implementation only supports its C subset, due to the capabilities of our compiler toolchain. Using the NVIDIA binary utilities such as `cuobjdump`, we verify that similar to the application of TASA in CPUs, the location of functions and variables in the program executable follows their definition order in the source file. Moreover, we verify with microbenchmarks that the randomisation of GPU code randomises GPU cache conflicts, which in turn provide random execution times.

## 5 Methodology

In order to compute the WCET of the pedestrian detection use case which is considered our *Unit of Analysis (UoA)*, first we measure the timing of the pedestrian detection partition in isolation. In conventional measurement based timing analysis (MBTA) for single core systems, the maximum execution time ($max(ET_{UoA_{isol}})$) is inflated by an engineering margin (EM) in order to compute the WCET:

$$WCET_{MBTA} = max(ET_{UoA_{isol}}) \times EM \tag{1}$$

In order to adapt the WCET to account for the contention created by other partitions, we use two different approaches, similar to [2].

In order to make the estimation completely independent of the software running on the other partitions, and therefore support arbitrary updates of any partition, we use a *Fully Time Composable (FTC)* approach. This approach considers the worst possible contention from the other partitions running in the cores 2 to 8 ($WC_i,\ i \in [2, 8]$), therefore at system operation the actual contention of other partitions is lower than the one considered at analysis:

$$WCET_{FTC} = max(ET_{UoA}) \text{ under } WC_i \text{ , where } i \in [2, 8] \\ \text{and } max(ET_{UoA}) \geq max(ET_{UoA_{isol}}) \tag{2}$$

In order to obtain the $WCET_{FTC}$, we perform the measurement of the UoA, which is executed in the CPU core 1, while the rest of the cores in the system (7 cores, from CPU 2 until CPU 8) are executing a program which generates the maximum contention. In order to achieve this effect, we are using a microbenchmark which always misses in the L2 and L3 shared caches, similar to [2]. Therefore, whenever the software of the pedestrian detection partition (UoA) attempts to access memory (L2, L3 or the LPDDR) it suffers multicore contention from 7 more cores.

Although the $WCET_{FTC}$ estimation obtained with the Full Time Composable scenario is completely independent of the actual software executed in the rest of the CPU cores or partitions (given the one-to-one mapping of partitions to CPU cores), this comes at the expense of excessive pessimism in the WCET.

However, the resulting measured pessimism of FTC is still lower than the theoretical upperbound of the multicore contention. Note that the theoretical upperbound of this method is to consider that in an 8 core processor, in the worst case the expected maximum slowdown that can be experienced due to memory contention is 8×. The reason for this is the following: assuming that the memory hierarchy serving the 8 cores of the system is completely fair, each core will get 1/8th of its bandwidth.

In terms of Power consumption, the same Fully Composable method can be applied. The power consumption is observed under the same worst case contenders in order to generate a higher power consumption than the one that is going to be experienced during the system deployment.

The manufacturer-provided Thermal Design Power (TDP) is also a safe upper bound that can be used, regardless of the workload executed in the other cores.

In order to alleviate the pessimism of FTC, we also consider a *Partial Time Composable scenario (PTC)*. In this case, the maximum contention ($max(C_i)$) generated by each partition is measured using the UP2DATE middleware in terms of cache misses per second. Then a higher contention threshold ($TH_i$) is decided for each partition ($CT_i$), which can account for future updates of that partition.

$$CT_i = max(C_i) \times TH_i \tag{3}$$

Based on this threshold, a configurable contender is created, which generates at least $CT_i$ memory contention. By replacing each partition with its corresponding configurable contender, the partition under analysis (the pedestrian detection partition in our case) is software randomised, and its

**Table 1: Pedestrian detection execution time measurements (ms) in isolation (left) and FTC (right)**

| | Time per processed image (ms) | |
|---|---|---|
| | **isolation** | **FTC** |
| **Minimum** | 17,519 | 18,605 |
| **Average** | 19,00468 | 22,24689 |
| **Maximum** | 26,521 | 111,923 |

probabilistic Worst Case Execution time is computed under that configuration ($WCET_{conf}$) using MBPTA.

The purpose of using software randomisation in the software under analysis is two fold: first, it allows to compute the WCET using MBPTA. The second reason is that the measured execution time under contention depends on the alignment in time of the memory requests generated by the other partitions. Unlike the FTC case in which every time that the pedestrian detection application tries to access memory suffers contention from the other 7 cores, in the PTC scenario this contention can be much lower. Therefore, there is a chance that the software under analysis accesses memory when the memory subsystem is not occupied serving requests by other CPUs/partitions. However, software randomisation changes both the number of miss requests generated from the software under analysis (i.e. due to the different conflicts among them in the caches) as well as the execution time of the task under analysis. Therefore, it allows to cover more cases during the analysis of the system.

In that case, the partial time composable WCET ($WCET_{PTC}$) is defined as:

$$WCET_{PTC_{conf}} = WCET_{conf} \text{ under } CT_i \text{ , where } i \in [2, 8] \quad (4)$$

It is worth noting that this PTC WCET is only valid for the particular configuration ($conf$ i.e. number of partitions and their specific thresholds) from which it was derived. For example, a given configuration may use less than 8 cores, e.g. if the number of partitions deployed in the system is lower. However, if a software update enables the use of an additional core, then the PTC WCET is no longer valid and the analysis needs to be repeated.

Moreover, this $WCET_{PTC_{conf}}$ continues to remain valid for as long as any new version of a partition after an update does not exceed the contention threshold $CT_i$ used in the determination of the WCET. At system operation, the UP2DATE monitoring middleware ensures that each partition stays within the limit defined in the particular configuration used during the analysis and takes corrective action. Examples of corrective actions are restarting the offending partition, stopping the partition if it is not a high criticality one, or rolling back to the previous software version. In our case, no violations have been observed, providing additional evidence that our estimations are valid.

For the energy consumption under the Partially Composable scenario we apply the same methodology. Again the software randomisation of the unit under analysis is used in order to generate power peaks that might not be observed during the measurements performed at analysis, e.g. due to the increased number of misses generated by a given memory layout, and predict its probabilistic worst case using Extreme Value Theory, similar to [7]. In addition, the configurable contenders allow to upperbound the power consumption of the activity in the other cores.

For the implementation of the worst case contention of the FTC we are using a microbenchmark that always misses in the various cache levels similar to [2], as well as the linux *stress* benchmark for memory traffic and we select the one that generates the highest slowdown to the pedestrian detection partition. For the configurable contenders we use the same approach, but the contention of each contender is tuned using the *cpulimit* linux utility, in order to match the selected $CT_i$ value.

## 6 Evaluation

In the left column of Table 1 the maximum observed execution time ($max(ET_{UoA_{isol}})$) from equation 1) per frame is 26,521 ms. In traditional measurement-based timing analysis methods (MBTA) applied on single core setups, this time is inflated by an engineering margin (EM) based on prior experience with the platform, in order to account for unknown latencies introduced by the architecture. In the PowerPC MPC 755 processor used in avionics, this EM is 20% ( therefore the $EM$ in equation 1 is 1,20 in this case) when the L2 cache of this single core processor is disabled [20]. Enabling the L2 cache would make the engineering margin much higher, and therefore cancel out much of the performance benefit offered by the use of the cache. The reason for this is that the L2 cache is unified and write-back, which means that the worst case for each cache entry is encountered when the replaced entry is dirty, and therefore its value needs to be written to memory before storing the new contents of the cache line. Consequently, level two or higher level caches are not used in industrial deployments of real-time systems or they are used under very specific configurations. To our knowledge, the only solution capable of analysing multi-level caches with a measurement based method, including the particularly challenging case of unified caches, is using time-randomisation [21].

For the NVIDIA Xavier which contains 3 cache levels and additional high-performance features compared to the MPC 755 such as dynamic branch predictor, prefetchers and multicore contention etc, there is no prior experience which can suggest an appropriate EM, although the previous example indicates that a higher EM is expected. However, for the sake of simplicity for the rest of this section we will assume optimistically that 20% is a sufficient EM also for NVIDIA Xavier as it has been done for comparison reasons in the MBPTA literature [20] [21]. Therefore, we assume that the WCET estimation per frame is $WCET_{MBTA} = 26,521 \times 1,20 = 31,8252$ms.

The average power consumption per frame with this setup is 9,99W while the maximum observed (high watermark) power consumption per frame is 10,973W. Note that the NVIDIA-supplied TDP (Thermal Design Power) of the platform we are using is 30W. In the absence of a reliable way to measure

**Table 2: Analysis of each UP2DATE partitions in terms of cache misses per second**

| | use case | online monitoring | update_manager | external_update_manager | offline_monitoring |
|---|---|---|---|---|---|
| **L1 Accesses** | 126953948,353333 | 1059333 | 511360,7 | 218952,5 | 1701044 |
| **L1 Misses** | 783343,2 | 48374,79 | 35733,05 | 19553,89 | 73019,85 |
| **L2 Accesses** | 81260172 | 6660086 | 2490600 | 2133653 | 3290321 |
| **L2 Misses** | 1926062 | 888311,5 | 399974,2 | 344016,6 | 560166,6 |

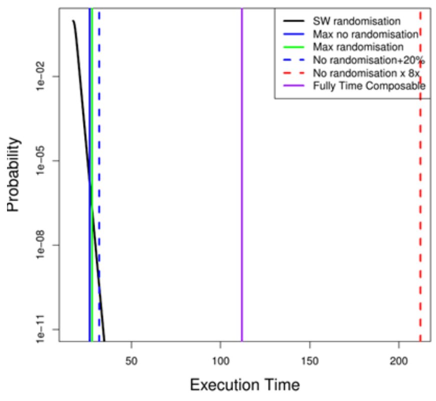


**Figure 2: PTC pWCET curve obtained using MBPTA and comparison with conventional timing analysis.**

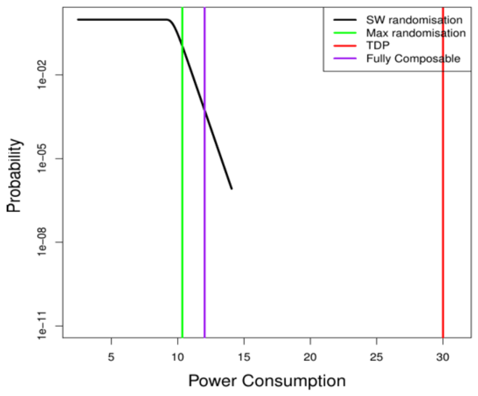


**Figure 3: Partially Composable Probabilistic Worst Case Power Consumption curve obtained using EVT and comparison with TDP.**

worst-case power consumption, this is the safe upper-bound that has to be used as a worst-case estimation.

Then, we profile the pedestrian detection partition under heavy stress to observe the worst-case execution time in terms of multicore contention for the FTC setup, in which the software executed in the other cores is replaced by worst-case contenders. Therefore the observed execution time is a safe upper-bound of the actual execution time when any other software is executed in the other cores. In the right column of Table 1 the maximum observed execution time ($WCET_{FTC}$) per frame is 111,923ms, which is 4,2$\times$ higher than the baseline execution time ($max(ET_{isol})$). Since the NVIDIA Xavier contains 8 CPUs, the theoretical maximum bus contention can be up to 8$\times$. Note that this is a conservative estimate, since prior work has shown that the combined effect of bus, a shared L2 cache and memory controller of a 4 core processor can be as high as 9$\times$ and under certain cases, more than 9$\times$ [22]. Therefore, under the assumption that 8$\times$ can be the contention upper-bound for NVIDIA Xavier, this means that the use of worst-case contenders in UP2DATE, can provide a WCET improvement in the full time composable scenario of 52%.

In terms of power consumption per frame, the average power consumption in the FTC scenario (i.e. under worst case contenders) is 11,35W, while the worst case observed (high watermark) power consumption is 12,05W, which is 10% higher than the baseline. Considering again that the safe upper bound of the power consumption is the 30W TDP provided by the manufacturer, the use of worst-case contenders in UP2DATE provides a 60% improvement.

Next, we analyse each UP2DATE partition in terms of cache misses in order to build a custom configurable contender which can replace each one. The obtained values are presented in Table 2. We select a tight threshold ($TH_i$) of 1%, which is one of the tightest PTC estimations we could provide.

Finally, we replace each UP2DATE partition with configurable contenders and apply software randomisation to the pedestrian detection, so that MBPTA can be used to estimate its probabilistic WCET (pWCET) and power consumption. We perform 500 randomisations and collect measurements over 2000 experiments, which we verify that satisfy the i.i.d property using the Box-Ljung test for independence and the Kolmogorov- Smirnov test for identical distribution.

Figure 2 shows the pWCET curve obtained using Measurement-Based Probabilistic Timing Analysis (MBPTA). The black line as well as the maximum observed execution time with software randomisation, are slightly longer than the corresponding one of the non-randomised version. This is expected since software randomisation allows to explore much more memory layouts and contention alignments in time, so that their timing effect is taken into account. We notice that for the cut-off probability $10^{-12}$ the estimated WCET ($WCET_{PTC}$) is slightly higher than the optimistic 20% engineering margin which was assumed to be enough for Xavier. This partial time composable pWCET is 4$\times$ smaller than the fully time composable WCET as well as 7,5$\times$ smaller compared to the theoretical maximum contention that can be experienced among the 8 cores of the NVIDIA Xavier.

The application of MBPTA for the power measurements shown in Figure 3 predicts that the power consumption can be up to 15W per frame. This estimate is larger than the highest observed power consumption of 12,05W in the fully composable case. This is valid because despite the fact that the worst-case contenders can increase power consumption, they are not able to increase the power of the platform to its maximum, since this depends also on the task under analysis. This estimate is 2× better than the conservative use of TDP.

## 7 Conclusion

In this paper we presented the Worst Case Execution Time and Power Estimation of a GPU application executing on the NVIDIA Xavier embedded multicore and GPU platform. We have achieved our goal using software randomisation which we enabled for the first time on GPU. Timing and power estimates per frame are 50-60% improved in the fully composable scenario compared to the theoretical contention (number of cores) and power consumption bounds (TDP). The partial composable scenarios provide a 4× improvement compared to the fully composable scenario in timing, but more conservative power results which are 50% better than the TDP.

## 8 Acknowledgments

This work was funded by the Ministerio de Ciencia e Innovacion - Agencia Estatal de Investigacion (PID2019-107255GB-C21/AEI/10.13039/501100011033 and IJC-2020-045931-I), the European Commission's Horizon 2020 programme under the UP2DATE project (grant agreement 871465), an ERC grant (No. 772773) and the HiPEAC Network of Excellence.

## References

[1] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-Based Probabilistic Timing Analysis for Multi-path Programs," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[2] E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernández, J. Abella, and F. J. Cazorla, "MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding," in *Reliable Software Technologies - Ada-Europe*, 2017.

[3] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, "A Cache Design for Probabilistically Analysable Real-time Systems," in *Design, Automation and Test in Europe (DATE)*, 2013.

[4] M. Fernández, D. Morales, L. Kosmidis, A. Bardizbanyan, I. Broster, C. Hernández, E. Quiñones, J. Abella, F. J. Cazorla, P. Machado, and L. Fossati, "Probabilistic Timing Analysis on Time-randomized Platforms for the Space Domain," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[5] L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. D. Berger, and F. J. Cazorla, "Probabilistic Timing Analysis on Conventional Cache Designs," in *Design, Automation and Test in Europe (DATE)*, 2013.

[6] L. Kosmidis, R. Vargas, D. Morales, E. Quiñones, J. Abella, and F. J. Cazorla, "TASA: Toolchain-agnostic Static Software Randomisation for Critical Real-time Systems," in *Proceedings of the 35th International Conference on Computer-Aided Design (ICCAD)*, 2016.

[7] D. Trilla, C. Hernández, J. Abella, and F. J. Cazorla, "An Approach for Detecting Power Peaks During Testing and Breaking Systematic Pathological Behavior," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019.

[8] S. Kotz and S. Nadarajah, *Extreme Value Distributions*. Imperial College Press, 2000.

[9] I. Agirre, P. Onaindia, T. Poggi, I. Yarza, F. J. Cazorla, L. Kosmidis, K. Grüttner, M. Abuteir, J. Loewe, J. M. Orbegozo, and S. Botta, "UP2DATE: Safe and Secure Over-the-air Software Updates on High-performance Mixed-Criticality Systems," in *23rd Euromicro Conference on Digital System Design (DSD)*, 2020.

[10] A. Jover-Alvarez, A. J. Calderón, I. Rodriguez, L. Kosmidis, K. Asifuzzaman, P. Uven, K. Grüttner, T. Poggi, and I. Agirre, "The UP2DATE Baseline Research Platforms," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[11] P. E. Ross, "Brakes that Slam Themselves: Automatic emergency braking will become standard in Europe," *IEEE Spectrum*, vol. 59, no. 1, 2022.

[12] M. M. Trompouki, L. Kosmidis, and N. Navarro, "An Open Benchmark Implementation for Multi-CPU Multi-GPU Pedestrian Detection in Automotive Systems," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017.

[13] P. A. Viola and M. J. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," in *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001.

[14] P. A. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, 2004.

[15] A. Jover-Alvarez and L. Kosmidis, "Open-Source UP2DATE Pedestrian Detection Use Case Repository." `https://gitlab.bsc.es/up2date/violajones-tasa`.

[16] L. Kosmidis, C. Maxim, V. Jégu, F. Vatrinet, and F. J. Cazorla, "Industrial Experiences with Resource Management Under Software Randomization in ARINC653 Avionics Environments," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2018.

[17] F. Cros, L. Kosmidis, F. Wartel, D. Morales, J. Abella, I. Broster, and F. J. Cazorla, "Dynamic software randomisation: Lessons learned from an aerospace case study," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.

[18] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer, "CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs," in *Conference on Compiler Construction (CC)*, 2002.

[19] L. Kosmidis, "Open-Source TASA repository." `https://gitlab.bsc.es/lkosmidi/tasa_cil`.

[20] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla, "Measurement-based Probabilistic Timing Analysis: Lessons from an Integrated-Modular Avionics Case Study," in *8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2013.

[21] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla, "Multi-level Unified Caches for Probabilistically Time Analysable Real-Time Systems," in *Proceedings of the IEEE 34th Real-Time Systems Symposium (RTSS)*, 2013.

[22] M. Fernández, R. Gioiosa, E. Quiñones, L. Fossati, M. Zulianello, and F. J. Cazorla, "Assessing the Suitability of the NGMP Multi-core Processor in the Space Domain," in *Proceedings of the 12th International Conference on Embedded Software (EMSOFT)*, 2012.

# 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
sales@adacore.com
www.adacore.com

506 Royal Road
La Caverne, Vacoas 73310
Republic of Mauritius
Contact: David Sauvage
david.sauvage@adalabs.com

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

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

Enterprise House
Baloo Avenue, Bangor
North Down BT19 7QT
Northern Ireland, UK
enquiries@sysada.co.uk
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

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch

Ada
europe