The journal for the international Ada community

Ada User Journal

Volume 43
Number 4
December 2022

Editorial 197
Quarterly News Digest 198
Conference Calendar 214
Forthcoming Events 218

AEiC 2022 BoF Session

J.P. Rosen. *Report on the ASIS BoF Session* 221

Procs. of the Workshop on Challenges and New Approaches for Dependable and CPS Engineering of AEiC 2022

W. John et. al. *ANIARA Project - Automation of Network Edge Infrastructure and Applications with Artificial Intelligence* 223

A Balador et. al. *DAIS Project - Distributed Artificial Intelligence Systems: Objectives and Challenges* 227

A. Bagnato et. al. *AI-Augmented Model-Based Capabilities in the AIDOaRt Project: Continuous Development of Cyber-physical Systems* 230

A. Bagnato, J. Krasnodebska. *MORPHEMIC - Optimization of the Deployment and Life-Cycle Management of Data-Intensive Applications in the Cloud Computing Continuum* 235

A. Imbruglia et. al. *5G Communication and Security in Connected Vehicles* 240

R. Sousa et. al. *Managing Non-functional Requirements in an ELASTIC Edge-Cloud Continuum* 245

Procs. of the HILT'22 - Supporting a Rigurous Approach to Software Development Workshop

C. Dross. *Containers for Specification in SPARK* 249

S. Tucker Taft. *Rigorous Pattern Matching as a Language Feature* 255

D. Larraz, C. Tinelli. *Finding Locally Smallest Cut Sets using Max-SMT* 261

L. Humphrey. *Basic Formal Verification of a Waypoint Manager for Unmanned Air Vehicles in SPARK* 269

Produced by Ada-Europe

Information on Subscriptions and Advertisements

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

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

ADA USER JOURNAL

Volume 43
Number 4
December 2022

Contents

	Page
Editorial Policy for *Ada User Journal*	196
Editorial	197
Quarterly News Digest	198
Conference Calendar	213
Forthcoming Events	217
AEiC 2022 BoF Session	
J. P. Rosen. *"Report on the ASIS BoF Session: The Future of ASIS and Vendor Independent Tools"*	221
Proceedings of the "Workshop on Challenges and New Approaches for Dependable and Cyber-physical Systems Engineering" of AEiC 2022	
W. John et al. *"ANIARA Project - Automation of Network Edge Infrastructure and Applications with Artificial Intelligence"*	223
A. Balador, S. Sinaei, M. Pettersson, I. Kaya. "*DAIS Project - Distributed Artificial Intelligence Systems: Objectives and Challenges"*	227
A. Bagnato, A. Cicchetti, L. Berardinelli, H. Bruneliere, R. Eramo. *"AI-Augmented Model-Based Capabilities in the AIDOaRt Project: Continuous Development of Cyber-physical Systems"*	230
A. Bagnato, J. Krasnodębska. *"MORPHEMIC - Optimization of the Deployment and Life-Cycle Management of Data-Intensive Applications in the Cloud Computing Continuum"*	235
A. Imbruglia, D. Cancila, M. Settembre. *"5G Communication and Security in Connected Vehicles"*	240
R. Sousa, E. Sabate, M. González-Hierro, A. Barros, C. Zubia, L. M. Pinho, E. Kartsakli. *"Managing Non-functional Requirements in an ELASTIC Edge-Cloud Continuum"*	245
Proceedings of the "HILT'22 - Supporting a Rigorous Approach to Software Development Workshop"	
C. Dross. *"Containers for Specification in SPARK"*	249
S. Tucker Taft. *"Rigorous Pattern Matching as a Language Feature"*	255
D. Larraz, C. Tinelli. *"Finding Locally Smallest Cut Sets using Max-SMT"*	261
L. Humphrey. *"Basic Formal Verification of a Waypoint Manager for Unmanned Air Vehicles in SPARK"*	269
Ada-Europe Associate Members (National Ada Organizations)	277
Ada-Europe Sponsors	Inside Back Cover

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 referring to the convergence between the Ada User Journal and the sister publication Ada Letters, promoted by the existing collaboration between Ada-Europe and ACM SIGAda. This makes it easier to establish and maintain a collaboration for sharing content between the two publications. The expected benefits are both for readers, who gain access to richer content, and contributors, who reach a wider audience. We plan to continue this collaboration while moving towards a possible unification of the two publications, such that all subscribers of the AUJ and the Ada Letters, typically members of Ada-Europe or ACM SIGAda, will be receiving the same and a single publication at some (hopefully) near future.

Concerning the contents of this last issue of 2022, we include the proceedings of DeCPS 2022, the "Workshop on Challenges and New Approaches for Dependable and Cyber-physical Systems Engineering" that took place with AEiC 2022, and the initial part of the proceedings of "HILT 2022 - Supporting a Rigorous Approach to Software Development Workshop", the seventh in the HILT series of conferences and workshops focused on the use of High Integrity Language Technology. Overall, we include 10 papers, covering the work being done in several projects in the CPS and IoT areas (ANIARA, DAIS, AIDOaRt, MORPHEMIC, ELASTIC), providing insights on cybersecurity challenges when using 5G, addressing language-related topics (how to use containers for specification in SPARK, and how to exploit pattern matching as a programming language feature), and proposing approaches related to the application of formal methods for system development and verification. We hope the reader will appreciate the quality of these technical contents.

I would like to add a note to highlight a short contribution by Jean-Pierre Rosen, reporting on the ASIS Birds-of-a-Feather Session that took place in AEiC 2022, and to highlight the new process for commenting on the Ada language standard, which is fully explained on page 219.

Finally, and as usual, we include the News Digest section prepared by its editor, Alejandro Mosteo, and the Calendar and Events section prepared by Dirk Craeynest, which closes with a two-page announcement of AEiC 2023 to be held mid-June in Lisbon.

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

Quarterly News Digest

Alejandro R. Mosteo

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

Contents

Preface by the News Editor	198
Ada-related Events	198
Ada-related Resources	200
Ada-related Tools	201
References To Publications	205
Ada and Other Languages	205
Ada Practice	206

[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,

Do not miss the upcoming Ada-Europe 2023 conference, for which an announcement can be found in this issue [1]. The conference will be celebrated, as always, in mid-June and will take place this year in lovely Lisbon.

I will also take this opportunity to congratulate Fabien Chouteau on receiving the 2022 ACM SIGAda Award for Outstanding Ada Community Contributions [2]. This award is entirely deserved, and I look forward to seeing his future initiatives to promote the Ada language in the open source community.

Finally, with this issue, we are mostly caught up and back on track with our regular news schedule. Remember that the last News Digest included only timely announcements, as the issue devoted most of its space to technical papers.

[1] "CfC 27th Ada-Europe Int. Conf. Reliable Software Technologies", in Ada-related Events.

[2] "Winners of 2022 ACM SIGAda Awards", in Ada-related Events.

Sincerely,
Alejandro R. Mosteo.

Ada-related Events

ACM SIGAda HILT'22 Workshop on Supporting Rigorous S/W Development

[Event in the past, for the record. —arm]

From: Tucker Taft <tucker.taft@gmail.com>
Subject: Re: ACM SIGAda HILT'22 Workshop on Supporting Rigorous S/W Development -- Oct 14, 2022
Date: Tue, 4 Oct 2022 13:59:44 -0700
Newsgroups: comp.lang.ada

There is now an online option for attending the ACM SIGAda HILT'22 workshop featuring Niko Matsakis and Rustan Leino.

Anyone registered for the workshop will receive a link allowing use of Zoom and/or the "Whova" app to attend the workshop remotely. The organizers of the associated conference (ASE'22) have indicated that remote attendees may register at the lowest attendee price ("Student Member"). So if you or a colleague might be interested in participating in the workshop remotely, please register soon for the October 14th workshop, at: https://conf.researchr.org/attending/ase-2022/registration and indicate "Student Member" as your category of attendee.

[Original announcement omitted. —arm]

For more information see: https://conf.researchr.org/track/ase-2022/ase-2022-workshop-hilt-22

#formalmethods #softwareengineering #ada #rust #spark #dafny #ACM #ASE

FOSDEM 2023: Call for Devroom

From: Mockturtle <framefritti@gmail.com>
Subject: FOSDEM 2023: call for devroom. Deadline: 18/10
Date: Sun, 9 Oct 2022 10:11:14 -0700
Newsgroups: comp.lang.ada

Dear all,
I just discovered that on 29/9 FOSDEM 2023 published the Call for DevRoom.

The deadline for the proposal is 18/10.

https://fosdem.org/2023/news/2022-09-29-call_for_devrooms/

From: Dirk Craeynest <dirk.craeynest@gmail.com>
Date: Sun, 9 Oct 2022 23:17:28 -0700

> I just discovered that on 29/9 FOSDEM 2023 published the Call for DevRoom.

We've been working behind the scenes on this already. Stay tuned for an announcement with more details on the AdaFOSDEM mailing list in the very near future!

Fer and Dirk

Winners of 2022 ACM SIGAda Awards

From: Tucker Taft <tucker.taft@gmail.com>
Subject: ANN: Winners of 2022 ACM SIGAda Awards
Date: Fri, 21 Oct 2022 11:54:27 -0700
Newsgroups: comp.lang.ada

ACM Special Interest Group on Ada (SIGAda) is pleased to announce the following SIGAda awards for 2022.

==========

Winner of the 2022 Robert Dewar Award for Outstanding Ada Community Contributions, for broad, lasting contributions to Ada technology and usage:

Fabien Chouteau

Fabien Chouteau has been the lead of AdaCore's Ada Community outreach activities for many years. He has been the energy behind the "Make With Ada" and "Crate of the Year" contests, and has invigorated the Ada hobbyist market by encouraging support of amateur Ada champions, fostering the development of the excellent Alire package manager for Ada, and working to move all AdaCore libraries from GPL to a more permissive ("Apache 2.0") license.

==========

Winner of the 2022 ACM SIGAda Distinguished Service Award, for exceptional contributions to SIGAda activities and products:

Luis Miguel Pinho

Luis Miguel Pinho (PhD SMIEEE SMACM) is a Professor and Researcher in the Computer Engineering Department of the Polytechnic of Porto - School of

Engineering (ISEP), in Portugal. Miguel is the current editor of Ada Letters and serves as the SIGAda Secretary-Treasurer. He was a member of the Research Center in Real-Time and Embedded Computing Systems, and Executive Director of the Porto Research, Technology & Innovation Center. He served as General Chair and Program Co-Chair of Ada-Europe 2006 and General Co-Chair of ARCS 2015, was a Keynote Speaker at RTCSA 2010 and Program Co-Chair of Ada-Europe 2012, Ada-Europe 2016 and RTNS 2016. He was Editor-in-Chief of the Ada User Journal, and is a member of the HiPEAC network of excellence.

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Wed, 26 Oct 2022 02:39:03 -0700

Thanks a lot Tuck,
I am honored to receive the ACM SigAda award for my contribution to the Ada community.

It's been a blast working towards the broader adoption of Ada/SPARK, improving the ecosystem, and seeing the community evolve along the years. And this is a good opportunity for me to thank everyone who contributed to this effort, and/or who believed in our vision for the future of the Ada/SPARK community.

There is still a lot to do obviously, and I think the biggest challenge is to show those who have already seen Ada in the past how the language and its ecosystem have evolved. But we entered an exciting time for Ada/SPARK, as more and more people are questioning their choice of programming languages.

In my opinion, the most important topics for the future of Ada/SPARK are:

First, foster collaboration and welcome newcomers. This is why Alire and its ecosystem are game changing.

Second, spread awareness on the amazing power of SPARK, and have it recognized as the truly bleeding edge technology it is.

Third, use the technology to show what it can do. Since my first "Make With Ada" blog post in 2015, I have always been convinced that the best way to advocate for a technology is to use it. This is Make With Ada means to me.

Happy hacking!

No Ada DevRoom in FOSDEM 2023

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: No Ada DevRoom in FOSDEM 2023, alternative DevRooms and Ada-Europe support
Date: Tue, 8 Nov 2022 12:41:17 +0100
Newsgroups: comp.lang.ada

Dear Ada community,

Our proposal for an Ada Developer Room for FOSDEM 2023 has been declined. I asked whether we could have a virtual DevRoom just like in FOSDEM 2022, but it seems unlikely. This means Ada will (most likely) not take part in the new edition of FOSDEM. We are saddened by this decision, but the amount of proposals was indeed very large: 88 DevRoom proposals were submitted!

Nonetheless, we would like to encourage Ada developers to submit presentations to other DevRooms that may fit your interests You can find the accepted DevRooms in [1]. I think the rooms that could be of interests to the Ada community are "Confidential Computing", "Embedded, Mobile and Automotive", "FOSS Educational Programming Languages", "Microkernel and Component-based OS", "Open Source Firmware, BMC and Bootloader" and "Security". However, take a look at all the proposals! Maybe you are writing some RISC-V or networking software in Ada, and there is a DevRoom just for it Please keep the AdaFOSDEM mailing list [2] informed about submissions and definitely about accepted proposals: we'll build a consolidated list of Ada-related talks at FOSDEM 2023, as we did before [3]. If you have any questions or issues, we will gladly help you where we can.

We are also happy to announce that Ada-Europe [4], after learning that there would be no Ada DevRoom in FOSDEM, has opened the possibility of adding a new "DevRoom like" track in their 2023 conference [5]. The Ada-Europe conference will take place in Lisbon between the 13 and 16 of June, 2023. If you are interested in this possibility, please, contact Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> to let him know.

Best regards,
The Ada FOSDEM team

[1] https://fosdem.org/2023/news/2022-11-07-accepted-developer-rooms/

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

[3] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/21/210206-fosdem.html

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

[5] http://www.ada-europe.org/conference2023/cfp.html

Advent of Code 2022

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Advent of Code 2022
Date: Sun, 4 Dec 2022 03:50:09 -0800
Newsgroups: comp.lang.ada

In case you've missed it:
https://adventofcode.com/

There is even a chat room for Adaists about it @ https://forum.ada-lang.io/

Enjoy!

Happy Birthday, Ada!

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Subject: Happy Birthday, Ada!
Date: Sat, 10 Dec 2022 11:35:20 +0100
Newsgroups: comp.lang.ada

Born this date in 1815/1980.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Sat, 10 Dec 2022 03:31:23 -0800

Congratulation on Your Birthday,
Lady Ada

https://www.ada-deutschland.de/sites/default/files/AdaTourCD/AdaTourCD2004/Ada Magica/20.html

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 17 Dec 2022 05:47:10 -0600

Also the 10th Anniversary of Ada 2012.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Sun, 18 Dec 2022 04:35:38 -0800

> Also the 10th Anniversary of Ada 2012. - Randy.

I would have bet that this date would be the release of ISO 2022. So it's going to be 2023?

CfC 27th Ada-Europe Int. Conf. Reliable Software Technologies

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC 27th Ada-Europe Int. Conf. Reliable Software Technologies
Date: Tue, 20 Dec 2022 16:49:01 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

[CfC is included in the Forthcoming Events Section. —arm]

Post-Ada Workshop at Ada-Europe 2023

From: Marius Amado-Alves <amado.alves@gmail.com>
Subject: Post-Ada at Ada-Europe 2023 anyone?
Date: Wed, 21 Dec 2022 09:25:22 -0800
Newsgroups: comp.lang.ada

Would anyone be interested in co-organizing or attending a Post-Ada workshop at Ada-Europe 2023 (Lisbon, 13-16 June)?

Any thoughts appreciated.

The "Post-Ada" concept has been debated here in CLA. It encompasses lessons learnt from the three decade long Ada experiment, ideas for betterment of the language, and creation of languages anew, like Parasail and King.

One way to approach the 'problem' would be to classify features of Ada as "keep, kill, or to be improved," for example:

- loop statements: keep

- function expressions: keep

- cursors: kill

- attributes vs. operations (tick vs. dot): kill

- inheritance: to be improved

- Unicode characters and strings: to be improved

A general issue could be to compare or harmonize this approach with the future (?) revision of Ada via Ada Issues. Personally I feel Ada (202X) is already too big to grow anymore. I suspect compiler maintainers would agree, and hope they could participate (sponsor?)

Maybe a full-day workshop with the structure:

1. plenary: presentations, debate coffee break

2. creation of a list of topics, of some kind of organization lunch

3. parallel sessions by subgroups of participants, by topic

coffee break

4. plenary: subgroup reports, debate, integration, conclusion, maybe plans for the future

Please relay at will.

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 21 Dec 2022 18:08:18 +0000

> Would anyone be interested in co-organizing or attending a Post-Ada workshop at Ada-Europe 2023 (Lisbon, 13-16 June)?

> Any thoughts appreciated.

I probably won't be able to attend; my life is pretty much being destroyed right now.

> The "Post-Ada" concept has been debated here in CLA. It encompasses lessons learnt from the three decade long Ada experiment, ideas for betterment of the language, and creation of languages anew, like Parasail and King.

Really? No love for my "mad" :) ramblings?

https://github.com/Lucretia/orenda

> One way to approach the 'problem' would be to classify features of Ada as "keep, kill, or to be improved," for example:

> - loop statements: keep

> - function expressions: keep

> - cursors: kill

> - attributes vs. operations (tick vs. dot): kill

Wrong. Attributes are a really interesting and useful part of Ada and the solution in Orenda to getting addresses of objects, aspects would enable setting them on creation.

> - inheritance: to be improved

> - Unicode characters and strings: to be improved

Should be the basis of all text.

> A general issue could be to compare or harmonize this approach with the future (?) revision of Ada via Ada Issues. Personally I feel Ada (202X) is already too big to grow anymore. I suspect compiler maintainers would agree, and hope they could participate (sponsor?)

Won't happen, I've mentioned it before and was told it was not going to happen.

Ada-related Resources

[Delta counts are from November 13th to February 12th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 12 Feb 2023 12:44 CET
To: Ada User Journal readership

Ada groups on various social media:

- Reddit: 8_291 (+91) members [1]

- LinkedIn: 3_418 (+19) members [2]

- Stack Overflow: 2_309 (+36) questions [3]

- Telegram: 159 (+6) users [4]

- Gitter: 151 (+11) people [5]

- Ada-lang.io: 101 (+51) users [6]

- Libera.Chat: 82 (+5) concurrent users [7]

- Twitter: 32 (-5) tweeters [8]
  49 (-36) unique tweets [8]

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

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

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

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

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

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

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

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

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: 12 Feb 2023 12:44 CET
To: Ada User Journal readership

Rosetta Code: 920 (+1) examples [1]
39 (=) developers [2]

GitHub: 763* (=) developers [3]

Alire: 324 (+15) crates [4]

Sourceforge: 240 (+2) projects [5]

Open Hub: 214 (=) projects [6]

Codelabs: 54 (+1) repositories [7]

Bitbucket: 31 (=) repositories [8]

AdaForge: 0** (-8) repositories [9]

*This number is unreliable due to GitHub search limitations.

**This site is currently unreachable.

[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

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

Language Popularity Rankings

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: 12 Feb 2023 12:44 CET
To: Ada User Journal readership

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

- TIOBE Index: 23 (+4) 0.60% (+0.12%) [1]

- PYPL Index: 17 (=) 0.94% (+0.13%) [2]

- IEEE Spectrum* (general): 35 (=) Score: 1.16 [3]

- IEEE Spectrum (jobs): 33 (=) Score: 0.79 [3]

- IEEE Spectrum (trending): 32 (=)
  Score: 3.95 [3]

*The Spectrum ranking has been revamped, no longer using the same categories and rating methodology. Thus, historic trends are omitted for this issue except for the default category.

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

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

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

XMPP Public Ada MUCs

From: Alastair Hogge <agh@riseup.net>
Subject: Re: XMPP public Ada MUCs
Date: Wed, 7 Dec 2022 09:02:20 -0000
Newsgroups: comp.lang.ada

Someone has created an Ada MUC [multi-user chat] at xmpp:ada@conference.magicbroccoli.de. It is low traffic at the moment.

Interested participates can sign up for free XMPP accounts at:

https://404.city/

https://magicbroccoli.de/register/

Some information on getting started with XMPP:

https://xmpp.org/getting-started/

New Process for Submitting Comments about the Ada Language

From: Tucker Taft <tucker.taft@gmail.com>
Subject: New process for submitting comments about the Ada language
Date: Sun, 18 Dec 2022 16:54:55 -0800
Newsgroups: comp.lang.ada

[The announcement with the new commenting process for the Ada language standard appears in page 220 of this same issue —arm]

Ada-related Tools

AdaStudio-2022 Release 01/10/2022 Free Edition

From: Leonid Dulman <leonid.dulman@gmail.com>
Subject: Announce: AdaStudio-2022 release 01/10/2022 free edition
Date: Sat, 1 Oct 2022 00:19:44 -0700
Newsgroups: comp.lang.ada

I'm pleased to announce AdaStudio-2022.

It’s based on Qt-6.4.0-everywhere opensource (expanded with modules from Qt-5.15: qtgraphicaleffects qtgamepad qtx11extras qtwinextras), VTK-9.2.0, FFMPEG-5.1.1, OpenCV-4.6.0, SDL2-2.24.0, MDK-SDK(wang-bin)

Qt6ada version 6.4.0 open source and qt6base.dll ,qt6ext.dll (win64), libqt6base.so, libqt6txt.so(x86-64) built with Microsoft Visual Studio 2022 x64 Windows, GCC amd64 in Linux.

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

AdaStudio-2022 includes the following modules: qt6ada, vtkada, qt6mdkada, qt6cvada (face recognition, 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

[Removed detailed file contents. —arm]

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"

HAC v.0.21

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: HAC v.0.21
Date: Sat, 1 Oct 2022 02:37:27 -0700
Newsgroups: comp.lang.ada

HAC (HAC Ada Compiler) is a quick, small, open-source Ada compiler, covering a subset of the Ada language.

HAC is itself fully programmed in Ada.

Web site: http://hacadacompiler.sf.net/

From there, links to sources, and an executable for Windows.

Source repositories:

#1 svn: https://sf.net/p/hacadacompiler/code/HEAD/tree/trunk/

#2 git: https://github.com/zertovitch/hac

HAC is also available through Alire: https://alire.ada.dev/

* Main improvements since v.0.2:

  - Added Virtual Machine Variables, another means for exchanging data between the HAC program and the program hosting the VM.

  - SmallAda's tasking is working again in its HAC reincarnation -- at least, for some simple tasks.

  - HAL becomes HAT (HAC Ada Toolbox), to avoid name collision with HAL = "Hardware Abstraction Layer".

Enjoy!

LEA v.0.82

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Subject: Ann: LEA v.0.82
Date: Sat, 1 Oct 2022 02:46:10 -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

Improvements:

- more ready-to-use Ada code samples
- improved Dark Side look
- indentation lines
- improvements in navigation (find/replace, compilation errors)
- embeds HAC v.0.21; details: see other post...

Features:

- multi-document
- multiple undo's & redo's
- multi-line 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.

Enjoy!

VIM Bundle for Ada

From: Martin Krischik <martin.krischik@gmail.com>
Subject: VIM bundle for Ada
Date: Tue, 11 Oct 2022 10:19:57 -0700
Newsgroups: comp.lang.ada

I have updated the VIM bundle for Ada. If you are using VIM you should consider updating:

https://github.com/krischik/vim-ada

GCC 12.1.0 macOS Cross-compiler to arm-eabi

From: Simon Wright <simon@pushface.org>
Subject: Ann: GCC 12.1.0 macOS cross-compiler to arm-eabi
Date: Sat, 15 Oct 2022 20:11:45 +0100
Newsgroups: comp.lang.ada

Find the above at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-arm-eabi.

Built on Intel, also runs on Apple silicon under Rosetta.

Scroll down to the bottom of the page to find the installation package.

VisualAda for Visual Studio 2022 Release 1.0.0

From: Alex Gamper <alby.gamper@gmail.com>
Subject: ANN: VisualAda (Ada Integration for Visual Studio 2022) release 1.0.0
Date: Sat, 15 Oct 2022 15:06:20 -0700
Newsgroups: comp.lang.ada

Dear Ada Community,

VisualAda version 1.0.0 for Visual Studio 2022 has been released.

This is the initial release for Visual Studio 2022 and is a port of the existing VisualAda version 1.3 for Visual Studio 2017/2019.

Please feel free to download the free plugin from the following URL: https://marketplace.visualstudio.com/items?itemName=AlexGamper.VisualAda-2022

VIM Plugin Update

From: Martin Krischik <martin.krischik@gmail.com>
Subject: Another update to the VIM plugin.
Date: Tue, 25 Oct 2022 09:42:06 -0700
Newsgroups: comp.lang.ada

Since GPS support was dropped for macOS having proper Vim plugins for Ada has become kind of important again. I added Alire compiler support so a press of <F7> will compile again.

It's actually two updated:

https://github.com/krischik/vim-ada/releases/tag/v_5.1.0

https://github.com/krischik/vim-ada/releases/tag/v_5.2.0

Have fun.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Wed, 26 Oct 2022 00:21:19 -0700

Thanks Martin,

I also recommend using neovim instead of vim, because of the builtin LSP (language-server protocol) support. We can then independently install the Ada language server from AdaCore (https://github.com/AdaCore/ada_language_server), and with a small configuration step we now have full cross-references in Ada...

The main difficulty is loading the proper project file. I will likely write a small blog post on the subject, though I could simply post the config I have here if there's interest.

From: Martin Krischik <martin.krischik@gmail.com>
Date: Wed, 26 Oct 2022 07:45:36 -0700

Thanks for the heads up.

[...]

Nice, there is a macOS version. But I notice no dependencies to any GUI framework and when I did try it out there was indeed no GUI support. I'm actually using GVim — the Vim with the graphical user interface and I'm not going back to a Terminal based editor. Still good to know the option exists.

Gnu Emacs Ada Mode 7.3.1

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Gnu Emacs Ada mode 7.3.1 released
Date: Wed, 26 Oct 2022 06:29:58 -0700
Newsgroups: comp.lang.ada

Gnu Emacs Ada mode 7.3.1 is now available in GNU ELPA; the beta version has been promoted to release.

ada-mode and wisi are now compatible with recent GNAT versions. The grammar is updated to the proposed Ada 2022 version.

Incremental parse is provided. It still has some bugs, so it is not enabled by default. To try it:
(setq-default wisi-incremental-parse-enable t).

Incremental parse often gets confused; to recover, use M-x wisi-reset-parser. That does a full parse of the entire buffer, which can be noticeably slow in large buffers.

See the NEWS files in ~/.emacs.d/elpa/ada-mode-7.3.1 and wisi-4.0.0, or at http://www.nongnu.org/ada-mode/, for more details.

The required Ada code requires a manual compile step, after the normal list-packages installation ('install.sh' is new in this release):

```
cd ~/.emacs.d/elpa/ada-mode-7.3.1
./build.sh
./install.sh
```

This requires AdaCore gnatcoll packages which you may not have installed; see ada-mode.info Installation for help in installing them.

Gnu Emacs Ada Mode 8.0 Beta

From: Stephen Leake <stephen_leake@stephe-leake.org>
Subject: Gnu Emacs Ada mode 8.0 beta released.
Date: Mon, 07 Nov 2022 16:12:29 -0800
Newsgroups: comp.lang.ada

Gnu Emacs Ada mode 8.0 beta is now available in GNU ELPA devel for beta testing.

All Ada mode executables can now be built with Alire (https://alire.ada.dev/); this greatly simplifies that process.

gpr-query and gpr-mode are split out into separate GNU ELPA packages. You must install them separately (Emacs install-package doesn't support "recommended packages" like Debian does).

Ada mode can now be used with Eglot; this is controlled by new variables:

ada-face-backend - one of wisi, eglot, none
ada-xref-backend - one of GNAT, gpr_query, eglot, none
ada-indent-backend - one of wisi, eglot, none

The indent and face backends default to wisi if the wisi parser is found in PATH, to eglot if the Ada LSP server is found, and none otherwise. The xref backend also looks for the gpr_query executable in PATH.

The current AdaCore language server (23) support face but not indent. The current version of eglot (19) does not support face. So for now, eglot + ada_language_server only provides xref.

The AdaCore language server ada_language_server is installed with GNATStudio (which ada-mode will find by default), or can be built with Alire. If you build it with Alire, either put it in PATH, or set gnat-lsp-server-exec.

I have not tested ada-mode with lsp-mode. You can set ada-*-backend to 'other to experiment with that, or tree-sitter, or some other backend.

To access the beta version via Gnu ELPA, add the devel archive to package-archives:
(add-to-list 'package-archives (cons "gnu-devel" "https://elpa.gnu.org/devel/"))

Then M-x list-packages; the beta release shows as ada-mode version 8.0.3.0.20221106.55317, wisi version similarly.

Please report success and issues to the Emacs ada-mode mailing list https://lists.nongnu.org/mailman/listinfo/ada-mode-users.

The required Ada code requires a manual compile step, after the normal list-packages installation:

```
cd ~/.emacs.d/elpa/ada-mode-7.3beta*
./build.sh
./install.sh
```

If you have Alire installed, these scripts use it. Otherwise, this requires AdaCore gnatcoll packages which you may not have installed; see ada-mode.info Installation for help in installing them.

Artificial Intelligence Libraries

From: Marius Amado-Alves <amado.alves@gmail.com>
Subject: Re: Artificial Intelligence libraries in ADA
Date: Thu, 10 Nov 2022 09:58:27 -0800
Newsgroups: comp.lang.ada

Resurrecting this 3-year old thread, see what happens:-)

I too need AI and Machine Learning libraries, and I am literally disgusted at the perspective of having to use Python or Go or C++ for this. Has anything come up in the last 3 years? Maybe a binding to TensorFlow?

I plan to use Carter's REM NN, and maybe Kasakov's fuzzy_ml, for some experiments, but at some point, I'll want, like Bjorn Ludin, *recurrent* architectures, probably LSTM (Long Short* Term Memory), as I want to segment and classify text.

(Jeff: can we somehow reengineer REM NN towards recurrency? Maybe by inserting recurrent layers?)

*Not a typo. The ML geniuses really say "long short"...

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 10 Nov 2022 20:10:00 +0100

> (Jeff: can we somehow reenginer REM NN towards recurrency? Maybe by inserting recurrent layers?)

Probably best to discuss this off line. You can contact me by e-mail.

From: Rod Kay <rodakay5@gmail.com>
Date: Fri, 11 Nov 2022 21:20:04 +1100

> Also, Rod Kay (charlie on irc) did something re TF iirc.

I generated a thin binding to the TensorFlow C API via swig4ada around mid June. The binding has not been tested apart from a 'hello_TF' demo which simply calls the 'TF_Version' function and prints it.

I've been distracted by other projects since but as chance would have it, I've recently resumed work on swig4ada and TF will definitely be one of the top priorities re testing swig4ada.

I'll try to take another look at it this weekend and to get the TF binding onto github, if possible.

GCC 12.2.0 for macOS (x86_64 and aarch64)

From: Simon Wright <simon@pushface.org>
Subject: ANN: GCC 12.2.0 for macOS (x86_64 and aarch64)
Date: Sun, 20 Nov 2022 19:02:46 +0000
Newsgroups: comp.lang.ada

Find GCC 12.2.0 & tools for Intel silicon (will run on Apple silicon under Rosetta) at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.2.0-x86_64

Built on High Sierra with Python 3.9 (because Apple has withdrawn 2.7 in Monterey).

Also, the same for Apple silicon, built on Ventura but I've done my best to make sure it'll run on Monterey, at https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.2.0-aarch64

I've marked both as pre-release, but I'm especially interested (if anyone has some time on their hands) in a check of the aarch64 version on Monterey.

XNAdaLib and GNATStudio 2022 Binaries for macOS Monterey

From: Blady <p.p11@orange.fr>
Subject: [ANN] XNAdaLib and GNATStudio 2022 binaries for macOS Monterey.
Date: Sat, 26 Nov 2022 09:07:51 +0100
Newsgroups: comp.lang.ada

This is XNAdaLib 2022 built on macOS 12.6 Monterey for Native Quartz with GNAT FSF 12.1 (github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-x86_64) including:

- GTKAda 22.2 (www.adacore.com/gtkada) with GTK+ 3.24.33 (www.gtk.org) complete,
- Glade 3.40.0 (glade.gnome.org),
- Florist mid-2022a (github.com/Blady-Com/florist),
- AdaCurses 6.3 (patch 20221105) (invisible-island.net/ncurses/ncurses-Ada95.html),
- Gate3 0.5d (sourceforge.net/projects/lorenz),
- Components 4.64 (www.dmitry-kazakov.de/ada/components.htm),
- AICWL 3.25 (www.dmitry-kazakov.de/ada/aicwl.htm),
- Zanyblue 1.4.0 (zanyblue.sourceforge.net),
- PragmARC mid-2022 (pragmada.x10hosting.com/pragmarc.htm),
- UXStrings 0.4.0 (github.com/Blady-Com/UXStrings) - NEW
- GNOGA 2.2 mid-2022 (www.gnoga.com),
- SparForte 2.5 (sparforte.com),
- HAC 0.21 (https://hacadacompiler.sourceforge.io)

Here is also GNATStudio 23.0wb as a standalone app for macOS 12.

See readme for important details. There could be some limitations that you might meet. Feel free to report them on MacAda list (http://hermes.gwu.edu/archives/gnat-osx.html). Any help will be really appreciated.

Both packages have been posted on Source Forge: https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2022-monterey

Simple Components v4.65

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components v4.65
Date: Sat, 26 Nov 2022 23:08:41 +0100
Newsgroups: comp.lang.ada

The library 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 the previous version:

- Bug fix in HTTP server causing memory leaks in accumulated bodies when browser keeps connection on;
- Python bindings, backward compatibility to lower versions of Python 3, e.g. 3.8;

- Julia and Python bindings for OSX corrected.

Units of Measurement for Ada v3.12

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Units of measurement for Ada v3.12 (New SI prefixes)
Date: Sat, 26 Nov 2022 23:11:57 +0100
Newsgroups: comp.lang.ada

The library provides measurement unit support for Ada.

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

Changes to the previous version:

- Added four new SI prefixes adopted by General Conference on Weights and Measures (CGPM) in November 2022.

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 30 Nov 2022 07:44:13 -0800

This is a bit confusing. From

https://www.lne.fr/en/news/general-conference-weights-and-measures-2022:

to express quantities of digital information using orders of magnitude in excess of 1024, has been adopted.

Thus, four new prefixes have been introduced:

ronna pour 1027

ronto pour 10-27

quetta pour 1030

quecto pour 10-30

End quote.

Abbreviations?

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 30 Nov 2022 18:03:36 +0100

> ronna pour 1027
> ronto pour 10-27
> quetta pour 1030
> quecto pour 10-30
> End quote.
>
> Abbreviations?

q, r, R, Q

From: Adamagica <christ-usch.grein@t-online.de>
Date: Wed, 30 Nov 2022 09:49:48 -0800

> I'm wondering when these prefixes will turn up in https://physics.nist.gov/cuu/Units/prefixes.html.

You can find them here:

https://www.bipm.org/en/measurement-units/si-prefixes

The PragmAda Reusable Components

From: Pragmada Software Engineering <pragmada@pragmada.x10hosting.com>
Subject: [Reminder] The PragmAda Reusable Components
Date: Thu, 1 Dec 2022 11:57:28 +0100
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.

GNAT 12 on FreeBSD

From: Alastair Hogge <agh@riseup.net>
Subject: GNAT-12 on FreeBSD
Date: Mon, 12 Dec 2022 05:17:53 -0000
Newsgroups: comp.lang.ada

Description: This is an Ada compiler, from GCC-12.

Since Ada support must be built by an Ada-capable compiler, only platforms for which a bootstrap compiler is available can build it.

It is based on release versions of the Free Software Foundation's GNU Compiler Collection. It uses the GCC Runtime Library Exception, so the resulting binaries have no licensing requirements. Binaries produced by the AUX compiler should be legally handled the same as binaries produced by any FSF compiler.

It offers continuous improvements to the Ada 2022 standard since GCC 11.

https://www.freshports.org/lang/gnat12/
https://cgit.freebsd.org/ports/tree/lang/gnat12

laceOS: an Operating System Tailored for Ada Development

From: Rod Kay <rodakay5@gmail.com>
Subject: Ann: 'laceOS' ~ An operating system tailored for Ada development.
Date: Sat, 17 Dec 2022 17:23:24 +1100
Newsgroups: comp.lang.ada

After spending many years installing various operating systems and setting them up for Ada development, I thought I'd try to make a simple OS installer which contains all the configuration and packages I usually use.

I thought this might be useful to others, perhaps lecturers/students, hobbyists, newcomers to Ada or anyone wanting to experiment with the latest Ada features.

The installer is very simple, asking a few questions (several with defaults) and takes about 10 minutes to do the installation.

Here is the Github link for anyone interested ...

https://github.com/charlie5/laceOS

Feedback/critique/suggestions most welcome.

Regards.

P.S. The installer is written in Ada. :)

Adare_Net v0.0.128

From: Daniel Norte De Moraes <danielcheagle@tutanota.com>
Subject: ANN: Adare_Net v0.0.128
Date: Mon, 19 Dec 2022 20:08:53 -0000
Newsgroups: comp.lang.ada

Adare_Net new version v0.0.128:

Better code,

Added a adare_net.pdf manual,

Full client and server examples in udp and tcp.

Adare_net from version v0.0.128 approaches its v.0.1.0 version!

Adare_Net is a small, portable and easy to use Ada network lib. It supports ipv4 ipv6 udp and tcp, Socket Synchronous I/O Multiplexing and can 'listen' with ipv6, too.

https://github.com/danieagle/adare-net

SDLAda 2.5.5

From: Luke A. Guest <laguest@archeia.com>
Subject: [COTY] SDLAda-2.5.5 submitted
Date: Sat, 31 Dec 2022 14:27:23 +0000
Newsgroups: comp.lang.ada

Just to inform people that SDLAda isn't dead, yet, it's just dormant. I finally got around my issues with Alire and submitted the 2.5.5 crate.

https://github.com/AdaCore/Ada-SPARK-Crate-Of-The-Year/issues/22

References to Publications

NSA Guidance on Software Memory Safety

From: Jerry <list_email@icloud.com>
Subject: NSA Releases Guidance on How to Protect Against Software Memory Safety Issues
Date: Thu, 10 Nov 2022 15:48:00 -0800
Newsgroups: comp.lang.ada

"Examples of memory safe languages include C#, Go, Java®, Ruby™, Rust®, and Swift®."

https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/3215760/nsa-releases-guidance-on-how-to-protect-against-software-memory-safety-issues/

https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

Didn't the U.S. government once sponsor the development of a memory-safe language? (eye-roll)

Ada and Other Languages

MS Going to Rust (and Linux Too)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: MS going to rust (and Linux too)
Date: Sat, 24 Sep 2022 09:52:34 +0200
Newsgroups: comp.lang.ada

Apparently, Microsoft does not want to use C/C++ anymore:
https://www.zdnet.com/article/programming-languages-its-time-to-stop-using-c-and-c-for-new-projects-says-microsoft-azure-cto
and going to Rust. No word about glorious VBA and illustrious C#, though. The best ever inventions of the computing era deserve no mention... (:-))

Ah, GC does not sit well with them, who might think? (:-))

BTW, it seems that the Linux kernel will rust as well...

From: Luke A. Guest <laguest@archeia.com>
Date: Sat, 24 Sep 2022 09:50:33 +0100

> Apparently Microsoft does not want to use C/C++ anymore:

Yeah, they're 20 years behind, I came to that conclusion then.

Well, people and companies will follow like sheep.

>No word about glorious VBA and illustrious C#

They'll stay as they are but likely will move to being implemented in rust.

[...]

> BTW, it seems that the Linux kernel will rust as well...

There was conversation about using zig as well a while ago.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 24 Sep 2022 11:13:07 +0200

> They'll stay as they are but likely will move to being implemented in rust.

I bet MS-Rust gets written in QBasic... (:-))

> There was conversation about using zig as well a while ago.

This one is from Linus himself.

Anyway, as expected, since computing resources begin actively stagnating, damn, even a used rusted (no pun intended (:-)) 3 years old HDD is twice more expensive now, the SW industry slowly turns away from well established practices of not caring about performance, efficiency, quality etc. I wonder, who will first dare proclaim that Agile was trash... (:-))

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Sat, 24 Sep 2022 04:09:48 -0700

Sounds like "U.S. Department of Defense going to Ada" :-) ...

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Sat, 24 Sep 2022 13:41:24 +0200

> I wonder, who will first dare proclaim that [xyz] was trash... (:-))

Won't it be the presenter to use [xyz] in an economically informed speech about a new trendy replacement that is already a thing.

Trash in systems obeys a universal law, familiar to every consultant. That it piles up, and while leading to stagnation, trash also creates opportunities

- for oblivion,

- for cleaning out and

- for rebuilding.

A fresh start.

As a starting point, Rust has the fine mechanisms that will facilitate turning the language into a generator of consumable goods, including itself. It is, therefore, economically viable. By design, Rust meets many a business demand, since it doesn't stop at just technical ideas, of which it inherits many.

Write a really good driver for Linux using Ada 2012 and do not use capital letters in the source text, at least where Linux doesn't. Be silent about the language. Can an Adaist do that, to save the language?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 24 Sep 2022 14:31:38 +0200

> Write a really good driver for Linux using Ada 2012 [...]

The song remains the same. No, Python need not to have Linux drivers in order to be hugely popular, like the Herpes virus need not to be...

And it is not about Ada. It is about a potentially turning point as the SW developing process hits certain limits one ignored before. Selling hot air is a very respectable and profitable activity, but in this case the reality begins showing its ugly bigotry face. Though Ada could provide some answers, it is not in the game anyway. Nevertheless, things become interesting...

From: Nasser M. Abbasi <nma@12000.org>
Date: Sat, 24 Sep 2022 07:46:46 -0500

> BTW, it seems that Linux kernel will rust as well...

This is a link that talks about using rust in Linux kernel

"Linux embracing Rust will boost robotics community"

"Linus Torvalds mentioned that the Rust programming language would be used in the upcoming Linux 6.1 kernel"

<https://www.therobotreport.com/linux-embracing-rust-will-boost-robotics-community/>

What I do not understand is, why not Ada instead of Rust? I thought Ada was designed for embedded low level software.

Maybe it is just more verbose than rust, and do not use {}.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 24 Sep 2022 15:36:07 +0200

> What I do not understand is, why not Ada instead of Rust?

Look at it this way. If Linus was not aware 30 years ago that there were better OSes than UNIX and better languages than C, why should he suddenly do now?

> Maybe it is just more verbose than Rust, and do not use {}.

It is never technical. You can try to rationalize your preference afterwards, but in reality, it is free will at play, even in the case of choosing Ada.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sat, 24 Sep 2022 10:29:54 -0700

There is also a lot more emphasis on performance in the Rust world than in the Ada world. Part of this is due to resources, but a lot has to do with how the language itself is defined unfortunately. People working on the Linux kernel are definitely interested in performance (and remember they are using any programming language in a significantly different fashion than other programmers). People coming from C++ likely initially chose that language because it was advertised as the most performant.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Sat, 24 Sep 2022 19:56:02 +0200

> why should he suddenly do now?

Why not? He is actually talking about Rust, given C.

> It is never technical.

It needs to be technical to some extent. Suggesting to write a kernel in Python would encounter some technical opposition.

> You can try to rationalize your preference afterwards, but in reality, it is free will at play, even in the case of choosing Ada.

The point is that it's not free will. It seems about choice and about what drives choice. Some very old job descriptions very sincerely include "manipulating public opinion".

Think "Ada mandate"... Or better, don't, just don't.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 24 Sep 2022 21:07:01 +0200

> Why not? He is actually talking about Rust, given C.

He is just growing old... (:-))

> It needs to be technical to some extent.

To some very infinitesimal extent. Actually my point was that the extent has an obvious tendency to grow now. Which is why we observe knee-jerk reactions from some weaklings... (:-))

> Suggesting to write a kernel in Python would encounter some technical opposition.

Honestly? The next generation will fully embrace Python as soon the last of the old farts retire. Linux held way too long, IMO... (:-))

> It seems about choice and about what drives choice.

Huh, in the not so distant future I expect drivers using HTTP to communicate inside the kernel encoding data in JSON and XML and written in JavaScript... I am almost serious. This garbage triumphally marches across embedded world right now, so no smiley.

Ada Practice

Reexposing Generics Formal Parameters

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Subject: Calling inherited primitive operations in Ada
Date: Wed, 31 Aug 2022 01:15:38 -0700
Newsgroups: comp.lang.ada

[Although the original post is about reusing inherited subprograms, the conversation quickly veered into a technical issue with generic formals, which is raised in the first answer. —arm]

A small blog post that you might find interesting:

https://deepbluecap.com/calling-inherited-primitive-operations-in-ada/

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 31 Aug 2022 21:13:14 +0200

This same technique is used in generics to work around another language design "feature":

```
generic
   type Foo is ...;
package
   subtype Actual_Foo is Foo;
```

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Wed, 31 Aug 2022 23:56:26 -0700

To me, this is an orthogonal issue though (which would be worth its own blog post in fact). I can never remember (or perhaps not even understand) the reason for this limitation in Ada, which is a major pain when dealing with generics indeed…

I like the "Actual_" prefix, which I assume is some sort of convention in your code.

From: amo...@unizar.es <amosteo@unizar.es>
Date: Thu, 1 Sep 2022 00:57:33 -0700

Is this about how according to some mystifying rules generic formals are[n't] visible from outside the generic?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Sep 2022 12:02:43 +0200

Right. I do not remember the rules, just the fact that they are quite logical. Unfortunately the logic of [it] is not very helpful. (:-))

As for primitive operations the problems are on many levels, from lacking introspection to missing inheritance of implementation by composition (AKA hooking).

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 1 Sep 2022 13:59:29 +0200:

> Is this about how according to some mystifying rules generic formals are[n't] visible from outside the generic?

This seems like a non-issue to me. Any code that has visibility to a generic instance knows the actuals used for that instance. Can anyone provide real examples where this is a problem?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Sep 2022 14:37:36 +0200

> This seems like a non-issue to me. Any code that has visibility to a generic instance knows the actuals used for that instance.

That would make the code fragile. Should be avoided as much as possible as a form of aliasing. [...]

> Can anyone provide real examples where this is a problem?

Defaulted formal package actual part:

```
generic
   package Foo is new Bar (<>);
package Baz is ...
   -- What were these actuals in Foo?
```

This is one of most useful features used to reduce lists of formal parameters and simplify instantiations.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Thu, 1 Sep 2022 07:10:03 -0700

I have seen quite a number of cases of needing the subtype like Dmitry was showing. In a small number of cases, those were actual bugs in GNAT, but most of the time the compiler was correct. Mostly, you start to see the issue when you have generic packages that have formal generic packages.

Here is a quick example and the corresponding compiler error message. In Main, there is no way to see T. Of course, I can use Integer_Signature directly, but this is an issue.

If I rename Integer_Signature then I have to change a lot of places in my code (The aliasing that Dmitry was talking about)

```
with Signature;
generic
   with package Sign is new Signature (<>);
package Algo is
   procedure Compute (V : Sign.T) is null;
end Algo;

with Algo;
with Signature;
```

```
package Lib is
  package Integer_Signature is new
     Signature (Integer);
  package Integer_Algo is new Algo
    (Integer_Signature);
end Lib;

with Lib;
procedure Main is
  V : Lib.Integer_Algo.Sign.T;
  --   main.adb:3:24: "Sign" is not a visible
  -- entity of "Integer_Algo"
begin
  null;
end Main;

generic
  type T is private;
package Signature is
end Signature;
```

There are more interesting examples, somehow this one doesn't seem that bad. So here is another one:

```
   generic
     type T is private;
   package Gen is
   end Gen;

   with Gen;
   generic
     type T is private;
     with package Must_Match is new
       Gen (T);
     with package Need_Not_Match is new
        Gen (<>);
   package Gen2 is
     V1 : Must_Match.T;    --  "T" is not a
                -- visible entity of "Must_Match"
     V2 : Need_Not_Match.T;  -- instance of -
     -- same package,but this time T is visible
   end Gen2;

   with Gen, Gen2;
   procedure P2 is
     package G is new Gen (Integer);
     package G2 is new Gen2
       (Integer, G, G);
   begin
     null;
   end P2;
```

I dug out the explanation that Tucker Taft once sent to the Ada-Comment mailing list (2019-11-14):

<<<

10/2

{AI95-00317-01} The visible part of a formal package includes the first list of basic_declarative_items of the package_specification. In addition, for each actual parameter that is not required to match, a copy of the declaration of the corresponding formal parameter of the template is included in the visible part of the formal package. If the copied declaration is for a formal type, copies of the implicit declarations of the primitive subprograms of the formal type are also included in the visible part of the formal package.

10.a/2

Ramification: {AI95-00317-01} If the formal_package_actual_part is (<>), then the declarations that occur immediately within the generic_formal_part of the template for the formal package are visible outside the formal package, and can be denoted by expanded names outside the formal package.If only some of the actual parameters are given by <>, then the declaration corresponding to those parameters (but not the others) are made visible.

10.b/3

Reason: {AI05-0005-1} We always want either the actuals or the formals of an instance to be nameable from outside, but never both. If both were nameable, one would get some funny anomalies since they denote the same entity, but, in the case of types at least, they might have different and inconsistent sets of primitive operators due to predefined operator "reemergence." Formal derived types exacerbate the difference. We want the implicit declarations of the generic_formal_part as well as the explicit declarations, so we get operations on the formal types.

>>>

From: amo...@unizar.es
<amosteo@unizar.es>
Date: Thu, 1 Sep 2022 08:50:21 -0700

> I have seen quite a number of cases of needing the subtype like Dmitry was showing. In a small number of cases, those were actual bugs in GNAT, but most of the time the compiler was correct.

> Mostly, you start to see the issue when you have generic packages that have formal generic packages.

This matches exactly my experience. I don't have enough grasp of the details to come up with a realistic short example, but I did hit this issue pretty often in two libs where I used signature packages quite extensively:

https://github.com/mosteo/rxada

https://github.com/mosteo/iterators

Initially I was always under the impression I was hitting GNAT bugs but then it turned out there were rules about it. A couple example places (you can see the renamings at the top. I was adding them "on demand" so to say):

https://github.com/mosteo/iterators/blob/master/src/iterators-traits-containers.ads

https://github.com/mosteo/rxada/blob/master/src/priv/rx-impl-transformers.ads

Thanks Emmanuel for the examples and digging out Tucker's explanation.

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 1 Sep 2022 18:03:19 +0200

> Mostly, you start to see the issue when you have generic packages that have formal generic packages.

None of these deal with the example I responded to

```
generic
   type T is ...
package P is
   subtype Actual_T is T;
```

> Reason: {AI05-0005-1} We always want either the actuals or the formals of an instance to be nameable from outside, but never both.

This is true in all these examples. I have used Ada since 1984, and this has never been a problem for me (as initially presented, this would have existed in Ada 83). Of course, I generally avoid generic formal packages. They seem to me to be a work around for poor design, and I prefer to correct the design.

From: Emmanuel Briot
<briot.emmanuel@gmail.com>
Date: Thu, 1 Sep 2022 11:54:00 -0700

I think I have a more interesting example. This one is extracted from my attempted traits containers, for which I had published a blog post at AdaCore. My enhanced fork of the library is at

https://github.com/briot/ada-traits-containers

if someone wants to experiment with non-trivial generics code (and containers are of course a case where generics fully make sense).

Here is the example:

```
generic
  type Element_Type is private;
  type Stored_Type is private;
package Elements is
end Elements;

with Elements;
generic
  type Element_Type is private;
package Definite_Elements is
  package Traits is new Elements
    (Element_Type, Stored_Type =>
     Element_Type);
end Definite_Elements;

with Definite_Elements;
generic
  type Key_Type is private;
package Maps is
  package Keys is new Definite_Elements
    (Key_Type);
  function "=" (L, R : Keys.
    Traits.Stored_Type) return Boolean
    --   "Stored_Type" is not a visible entity of
    -- "Traits"
```

```
    is (False);
end Maps;
```

This is not case where the actual is visible unless I happen to know how Definite_Element is implemented and that it will use Element_Type for Stored_Type (and this is not a knowledge I wish client packages to have, the whole point of Element and Definite_Element is to basically hide how elements can be stored in a container, and whether we need memory allocation for instance).

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 1 Sep 2022 23:33:44 +0200

> [traits example]

As presented, this seems very over complicated. Elements and Definite_Elements add no value, and this is just a complex way of writing

```
generic
   type Key is private;
package Maps is
   function "=" (L : in Key; R : in Key) return
     Boolean is (False);
end Maps;
```

One hopes that the actual library makes better use of these packages than this small example. [...]

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Thu, 1 Sep 2022 23:11:45 -0700

> Elements and Definite_Elements add no value, and this is just a complex way of writing

I did point you to the full repository if you prefer an extensive, real-life code sample. This was just an extract showing the gist of the issue.

> One hopes that the actual library makes better use of these packages than this small example.

[...] These packages are mostly implementation details. They are used to build high-level packages similar to the Ada containers, except with much better code reuse, more efficient, and SPARK-provable.

> Ada has excellent features for hiding, but these are not among them. Assuming

And that's exactly our point in this discussion. Ada on the whole is very good (we would not be using it otherwise), but it does have a number of limitations which are sometimes a pain, this being one of them. Not acknowledging the limitations when they exist is naive, all languages have them.

[...]

From: amo...@unizar.es <amosteo@unizar.es>
Date: Fri, 2 Sep 2022 01:35:11 -0700

> this seems very over complicated. Elements and Definite_Elements add no value, and this is just a complex way of writing

Going in a tangent, and I guess you know perfectly well, but this is caused by the painful duplication of code that Ada pushes you to by not having a native way to abstract storage of definite vs indefinite types. So the provider of a generic library very soon faces this conundrum about duplicating most interfaces, if not implementations, or resort to non-trivial generics, or accept an unnecessary penalty for definite types, or push to the client the definite storage matter. There's simply no satisfying solution here (that I know of). The duplication of every standard container to have both (in)definite variants is a strong indictment.

I can understand the desire to have full control of allocation and object sizes, but that there's not a language way to work around this duplication, with appropriate restrictions to go with it, is... bothersome. Not making a dig at the ARG, which I understand is overstretched as it is.

There was a proposal circulating some time ago that seemed promising, that I can't quickly find. Something like

```
type Blah is record
   Dont_care_if_in_heap: new
     Whatever_Definiteness;
  -- Would apply to indefinite types or formals
end record;
```

I don't think it made into https://github.com/AdaCore/ada-spark-rfcs/ or https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues or I can't find it.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Sep 2022 10:48:53 +0200

> I can understand the desire to have full control of allocation and object sizes, but that there's not a language way to work around this duplication, with appropriate restrictions to go with it, is... bothersome. Not making a dig at the ARG, which I understand is overstretched as it is.

Containers should be implementable without generics. Just saying.

> There was a proposal circulating some time ago [...]

I would prefer constraint propagation/management support + tuples instead:

```
   type Blah (Parameters :
     Whatever_Definiteness'Constraints) is
     Sill_Care : Whatever_Definiteness
       (Parameters);
   end record;
```

From: amo...@unizar.es <amosteo@unizar.es>
Date: Fri, 2 Sep 2022 02:20:44 -0700

> Containers should be implementable without generics. Just saying.

Are you now referring to current Ada or to hypothetical features?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Sep 2022 11:55:11 +0200

> Are you now referring to current Ada or to hypothetical features?

Hypothetical like all types having classes and supertypes. E.g. when you instantiate generic with a type you semantically place the type in the implicit class of formal types of the generic. You cannot do that now without generics. Furthermore, there is no way to describe relationships between types like array index and array, like range and discrete type of its elements etc.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 2 Sep 2022 12:41:42 +0200

> [...]not having a native way to abstract storage of definite vs indefinite types [...]

The only indefinite data structure that is needed seems to be holders. Any other indefinite data structure can be implemented as the equivalent definite data structure of holders, so there need be no duplication of implementations. That one cannot use a single pkg for both does result in duplication of the spec, but that seems like less of an issue to me.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Sep 2022 13:04:34 +0200

> The only indefinite data structure that is needed seems to be holders.

The language should support and encourage design that does not rely on memory pools.

In my view one of the major advantages of Ada is that indefinite objects can be handled without resorting to hidden or explicit pointers to pools.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Fri, 2 Sep 2022 04:20:38 -0700

> I was not willing to spend more than about 15 minutes trying to understand this, so I may be missing something.

Fair enough. The library is really a set of experiments, mostly successful, and I think it might have been used for the implementation of the current SPARK containers in GNAT, though I am not positive there. I did look at the pragmARC components, and there you indeed chose to have a large number of similar-looking packages and code duplication. I guess we'll have just to

agree to disagree on the design approach there. But of course, users having choices is what makes an ecosystem interesting.

What I was really going after are graphs and their algorithms. In particular, I want those algorithms to work on any graph data structure provided it has a number of primitive operations. In fact, the algorithm could also work when the graph is kind of implicit in the code, even if we do not have an actual Graph object. And for this, you need generics.

A similar approach is what Rust uses all over the place with its traits, or what C++ Boost library uses for its graphs, too.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Sep 2022 19:01:28 -0500

>[...] painful duplication of code that Ada pushes you to by not having a native way to abstract storage of definite vs indefinite types.

This is premature optimization at its worst.

> [...]

There is no penalty in a code sharing implementation like Janus/Ada: the implementation of definite types is essentially the same as you would write by hand for an indefinite type. In most cases, all one needs is an indefinite generic.

(The plan for Janus/Ada was always to use post-compilation optimization to reduce the overhead of generics, but admittedly, that part never got built. If I had infinite time...)

Assuming otherwise is certainly premature optimization.

> There's simply no satisfying solution here [...]

The original expectation for the containers was that there would be many variants of each container, because the needs for memory management, task management, and persistence differ between applications: there is no one-size fits all solution.

But I agree on one point: the "basic" container is unnecessary; one should either use the indefinite or bounded container (depending on your memory management needs, either fully fixed or fully heap-based)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Sep 2022 19:07:27 -0500

>These packages are mostly implementation details [...]

(Wading in where I should probably not tread... :-)

But they violate the #1 principle of the Ada.Containers: ease of use. One principle that we insisted on was that a single instantiation was the maximum we would use, because we did not want people moving from arrays to containers to have to replace one declaration with a half page of magic incantations. (This is the reason that there is no container interface, for one consequence, and certainly no signature packages.)

In general, people either understand and like signature packages, or really do not understand them and just use them when insisted on. The standard containers in Ada needed to be usable by the maximum number of users, and insisting on bells and whistles that many don't understand does not help.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 2 Sep 2022 19:12:25 -0500

> In my view one of the major advantages of Ada is that indefinite objects can be handled without resorting to hidden or explicit pointers to pools.

But they're implemented with some sort of hidden allocation. (GNAT uses a "secondary stack", whatever that is, but that is just a restricted form of pool). Janus/Ada uses built-in pools with cleanup for all such things to simplify the interface (the code for allocations and stand-alone objects is mostly shared, both within the compiler and at runtime).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 3 Sep 2022 10:23:01 +0200

> But they're implemented with some sort of hidden allocation. [...]

For a programmer that does not matter. The problem with pools is locking, non-determinism, issues with protected actions. If [the] secondary or primary stack is the program stack, nobody really cares.

BTW, merely doing pool tracing/bookkeeping becomes a sheer nightmare if you cannot return a string from a function.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sat, 3 Sep 2022 10:59:16 +0200

> One principle that we insisted on was that a single instantiation was the maximum we would use

Except for queues

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 6 Sep 2022 19:42:57 -0500

> Except for queues

Right, and one consequence of that is that the queues aren't used much. (Not sure if they would be used much in any case, they're definitely a specialized need compared to a map.)

From: Simon Wright <simon@pushface.org>
Date: Sat, 03 Sep 2022 20:00:00 +0100

> One principle that we insisted on was that a single instantiation was the maximum

And this was one reason that I didn't put up any arguments at Ada Europe 2002 for the Ada 95 Booch Components to form a basis for Ada.Containers - you'd need 3 instantiations, one after the other.

```
-- A company's Fleet holds a number of
-- Cars.
  with BC.Containers.Collections.Bounded;
  with Cars;
  package My_Fleet is

     use type Cars.Car;

     package Abstract_Car_Containers
     is new BC.Containers (Cars.Car);

     package Abstract_Car_Collections
     is new
       Abstract_Car_Containers.Collections;

     package Fleets
     is new
       Abstract_Car_Collections.Bounded
       (Maximum_Size => 30);

     The_Fleet : Fleets.Collection;

  end My_Fleet;
```

The other was a lack of consistency in the implementation (Length? Size?).

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sun, 4 Sep 2022 23:56:43 -0700

> for the Ada 95 Booch Components [...] you'd need 3 instantiations

I definitely see the same issue. The way my library is trying to workaround that is as follows: Those instantiations are only needed for people who want/need to control every aspect of their containers, for instance how elements are stored, how/when memory is allocated, what is the growth strategy for vectors, and so on.

Most users should not have to care about that in practice. So we use code generation at compile time to generate high-level packages similar to the Ada containers, with a limited set of formal parameters (in src/generated, to be more specific). We can generate bounded/unbounded versions, definite/indefinite versions, and any combination of those.

One of the intentions of the library, initially, had been the implementation of the Ada containers and SPARK containers in GNAT, as a way to share as much code as possible between the two.

Randy Brukardt:

> Assuming otherwise is certainly premature optimization.

I am quoting a bit out of context, though I believe it is close enough. Designers of containers must care about performance from the get-go. Otherwise, people might just as well use a list for everything and just traverse the list all the time. We all know this would be way too inefficient, of course, which is why there are various sorts of containers. Anyone who has actually written performance-sensitive code knows that memory allocations are definitely something to watch out for, and the library design should definitely take that into account.

Jeff Carter:

> The only indefinite data structure that is needed seems to be holders

Although it is certainly true that using holders works, it is not applicable when designing a containers library that intends to be mostly compatible with Ada containers. The latter have chosen, long ago and before Holder was a thing, to have definite and indefinite versions. The main benefit to this approach is that users still retrieve directly the type they are interested in (e.g. String) rather than a holder-to-string. I must admit I have very limited experience with Holders, which I have never used in production code (nor, apparently, have my colleagues and ex-colleagues).

Randy Brukardt:

> Ada *DOES* support default values for formal parameters of generics

Hey, I just discovered that, thanks Randy ! For people who also did not know that:

```
generic
   type Item_Count is range <>  or use
     Natural;
package Gen is
```

It is supported by GNAT's newer versions (I don't know when it was implemented though)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 5 Sep 2022 09:34:37 +0200

> Although it is certainly true that using holders works, it is not applicable when designing a containers library that intends to be mostly compatible with Ada containers.

Right. Holder requires finalization and finalization means language prescribed finalization lists which is highly undesirable in many cases.

> The main benefit to this approach is that users still retrieve directly the type they are interested in (e.g. String) rather than a holder-to-string.

And that the container designer has control over the pool where the items get actually allocated.

> I must admit I have very limited experience with Holders, which I have never used in production code (nor, apparently, have my colleagues and ex-colleagues).

I have been using the idea for a long time, since Ada 95 before the standard library had them. In my experience holders multiply the number of container variants:

1. Definite elements

2. Indefinite elements

   +

3. Holder elements in the interface (and maybe implementation)

The third gets a holder package as a formal parameter or, alternatively, is a child of a holder package (for performance reasons). The container interface has direct operations in terms of the Element_Type as well as in terms of the holder type.

Sometimes the holder variant is actually the indefinite one that promotes holders only in its interface.

P.S. In my opinion helper types/package is an evil of far greater scale than any premature optimization!

The programmers doing the latter at least try to understand the code they write.

From: amo...@unizar.es <amosteo@unizar.es>
Date: Mon, 5 Sep 2022 01:53:19 -0700

> This is premature optimization at its worst.

Just because the language doesn't offer a way to do it. Otherwise I wouldn't need to care.

> There is no penalty in a code sharing implementation like Janus/Ada

Well, that sounds neat for Janus/Ada, but is a different issue to clients having to wrap their indefinite types prior to instantiation, and suffer the unwrapping throughout the code.

> Assuming otherwise is certainly premature optimization.

I'm of the opinion that it goes beyond just premature optimization, in the terrain of readability/maintainability by causing boilerplate, and when generics specializations do become necessary, by causing code duplication.

[...]

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Mon, 5 Sep 2022 11:30:56 +0200

> Those instantiations are only needed for people who want/need to control every aspects of their containers [...] So we use code generation at compile time to generate high-level packages similar to the Ada containers

This seems backwards. The user should encounter the forms most likely to be used first; if part of the packages are in a subdirectory, those should be the ones less likely for the typical user to use.

> The main benefit to this approach [definite+indefinite containers] is that users still retrieve directly the type they are interested in (e.g. String) rather than a holder-to-string.

Before Ada.Containers existed, I commonly used definite data structures from the PragmARCs with Unbounded_String, which is partly a specialized holder for String (and partly a specialized Vector for Positive/Character). Generalizing from this led to implementing a Holder pkg.

When I said a holder is the only indefinite pkg that is needed, I meant to imply that other indefinite structures would be implemented using the definite form + holder.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 6 Sep 2022 19:51:44 -0500

> I am quoting a bit out of context

Definitely out of context. If you have code which is truly performance sensitive, then it cannot also be portable Ada code. That's because of the wide variety of implementation techniques, especially for generics. (For Janus/Ada, if you have critical performance needs, you have to avoid the use of generics in those critical paths -- sharing overhead is non-zero.)

I agree that the design of the containers matters (which is why we made the sets of operations for the various containers as close as possible, so switching containers is relatively easy). But the indefinite/definite thing is premature optimization - it makes little difference for a Janus/Ada generic, at least in the absence of the full-program optimizer (that we never built). If your code isn't intended to be portable Ada code, then *maybe* it makes sense to worry about such things. But the expectation was always that containers would be useful in cases where the performance is *not* critical - one probably should use a custom data structure for performance critical things. (But most things aren't really performance critical in reality.)

GNAT Speed Comparison on Older Intel versus Apple Silicon M1

From: Jerry <list_email@icloud.com>
Subject: GNAT Speed Comparison on Older Intel versus Apple Silicon M1
Date: Tue, 8 Nov 2022 20:07:32 -0800
Newsgroups: comp.lang.ada

I use GNAT on a late 2008 MacBook Pro with a 2.4 GHz Intel Core 2 Duo for

heavy numerical computing. It is not uncommon for my programs to run several minutes to several hours. Does anyone have a feel for how much speed increase I would see using GNAT on an Apple Silicon M1 PowerBook Pro? My main curiosity is single-core runs since GNAT does not parallelize; I am aware that I can run multiple programs simultaneously on multiple cores.

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Date: Wed, 9 Nov 2022 08:38:48 +0100

Hi Jerry,

taking the results from Geekbench: [1] for your current MacBook and [2] for the M1 MacBook from 2021; the results show that single core performance of the M1 MacBook Pro is about 6.4 times faster.

However, notice that it is running on Aarch64 natively for the M1. Nonetheless, you can run x86 programs with little performance hit thanks to Apple Rosetta.

Also, GNAT afaik, allows for parallel computations using tasks. The multicore performance gain between the two models is about 24x.

These results are however just an average. Maybe your program does not see such improvements as it may bottleneck earlier or it may see greater gains.

Regards,

[1] https://browser.geekbench.com/macs/macbook-pro-early-2008

[2] https://browser.geekbench.com/v5/cpu/18518008

From: Jerry <list_email@icloud.com>
Date: Wed, 9 Nov 2022 22:26:18 -0800

> Hi Jerry,

> taking the results from Geekbench:

That's a great site. Thanks. Clicking around a bit I was able to find separate comparisons for single-core floating point and the speed-up is 5.2.

> However, notice that it is running on Aarch64 natively for the M1.

GNAT compiles to Aarch64 now, right?

> Nonetheless, you can run x86 programs with little performance hit thanks to Apple Rosetta.

"little performance hit" compared to Intel code running on Rosetta versus Intel silicon or compared to native ARM? And I wonder how long until Apple takes away Rosetta this time? Last time it was two OS updates and then, poof, gone.

From: Simon Wright <simon@pushface.org>
Date: Sun, 13 Nov 2022 16:29:54 +0000

> GNAT compiles to Aarch64 now, right?

You can download an aarch64-apple-darwin21 compiler for C, C++, Ada at [1]. However, it won't compiler C (or, I guess, C++) on Ventura - I'm working on a GCC 12.2 version.

[1] https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-12.1.0-aarch64-1

Variable Value If Exception Is Raised

From: Nytpu <alex@nytpu.com>
Subject: Variable value if exception is raised
Date: Sun, 20 Nov 2022 18:03:04 -0000
Newsgroups: comp.lang.ada

Hello everyone,

If an exception is *explicitly* raised during a variable assignment, what happens to the variable contents Are they in an undefined ("abnormal") state, or are the previous contents preserved?

For example:

```
with Ada.Text_IO;
procedure Test is
   function Always_Raises return Integer is
   begin
      raise Program_Error;
      return 1;
   end Always_Raises;

   I : Integer := 0;
begin
   -- insert a nested handler, because the
   -- ARM § 11.4 ¶ 3 *does* say that the
   -- currently executing body is "abnormally
   -- completed" (including finalizing
   -- everything) before
   -- entering the exception handler
   begin
      I := Always_Raises;
   exception
      when others => null;
   end;
   Ada.Text_IO.Put_Line(Integer'Image(I));
end;
```

What, if anything, will be printed (Disclaimer: I know the preexisting variable value will be preserved in GNAT specifically, but I'm asking if the standard guarantees that's the case)

I read through the ARM 2012 § 11 and § 5.2, as well as skimming through everything related to “assignment” and “exceptions” in the ARM index; and didn't see much relating to this. All I saw is this:

> When an exception occurrence is raised by the execution of a given construct, the rest of the execution of that construct is abandoned

— ARM 2012 § 11.4 ¶ 3

Which I guess implicitly protects variable values since assigning to a variable is performed after evaluating the right hand side, but still not necessarily a clear answer.

I did see in § 13.9.1 that language-defined validity checks (e.g. bounds checks) failing or calling `abort` in a task during an assignment will cause the variable to enter an "abnormal" (i.e. invalid) state, but that doesn't cover user-raised exceptions.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 20 Nov 2022 20:00:32 +0100

If the exception occurs during evaluation of the RHS, as in your example, then the language guarantees that the value of the LHS is unchanged. The execution of the assignment statement is abandoned before the value of the LHS is changed.

If an exception is raised while adjusting a controlled LHS, then the value of the LHS has already been changed before the exception is raised.

> -- insert a nested handler, because
> -- the ARM § 11.4 ¶ 3 *does*
> -- say that the currently executing
> -- body is "abnormally completed"
> -- (including finalizing everything)
> -- before entering the exception
> -- handler

This comment is false. Finalization does not occur until the exception handler finishes. Exception handlers would be pretty useless otherwise.

String View of File

From: Jesper Quorning <jesper.quorning@gmail.com>
Subject: String view of file
Date: Mon, 21 Nov 2022 00:30:00 -0800
Newsgroups: comp.lang.ada

Is it possible to write something like this with Ada

```Ada
package my_rw_file is new file
  (name => "whatever",
   mode => read_write,
   implementation => standard
   -- or portable or fast
  );
package as_string is new xxx
   (from => my_rw_file);
-- parse (as_string);
package data is new parse (as_string,
format => markdown); -- or whatever
```

Sorry, I’m new to Ada

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Mon, 21 Nov 2022 14:01:01 +0100

Do you mean, gobble up a file into a string and then parse that? Yes, that's possible in a number of ways.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Mon, 21 Nov 2022 14:48:47 +0100

[...]

If you want to read the arbitrary contents of a file into a String, that's easily done:

```
with Ada.Directories;
package String_A_File is
   use type Ada.Directories.File_Size;

   function File_As_String (Name : in String)
   return String with
     Pre  => Ada.Directories.Exists (Name)
   and then
       Ada.Directories.Size (Name) <=
       Ada.Directories.File_Size (Integer'Last),
       Post => File_As_String'Result'First = 1
   and
        File_As_String'Result'Last =
        Integer (Ada.Directories.Size (Name) );
end String_A_File;

with Ada.Sequential_IO;

package body String_A_File is
   function File_As_String (Name : in String)
return String is
      subtype FAS is String (1 .. Integer
      (Ada.Directories.Size (Name) ) );

      package FAS_IO is new
Ada.Sequential_IO (Element_Type => FAS);
      File   : FAS_IO.File_Type;
      Result : FAS;
   begin -- File_As_String
      FAS_IO.Open (File => File, Mode =>
        FAS_IO.In_File, Name => Name);
      FAS_IO.Read (File => File, Item =>
        Result;
      FAS_IO.Close (File => File);
      return Result;
   end File_As_String;
end String_A_File;
```

This presumes that Result will fit on the stack. If that's likely to be a problem, then you will need to use Unbounded_String and read the file Character by Character.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 21 Nov 2022 17:52:14 +0200

For the OP's benefit (Jeffrey of course knows this): an alternative to Unbounded_String is to allocate the Result string on the heap, and return an access to the heap string. With that method, you can still read the entire string with one call of FAS_IO.Read instead of Character by Character.

From: Marius Amado-Alves <amado.alves@gmail.com>
Date: Mon, 21 Nov 2022 08:11:05 -0800

Use Ada.Sequential_IO (Character), load to an Unbounded_String, save from a String or Unbounded_String.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Mon, 21 Nov 2022 17:42:56 +0100

> For the OP's benefit (Jeffrey of course knows this)

I know it, and I deliberately reject it. Having access types in a pkg spec is poor design. Delegating the associated memory management and all its opportunities for error to the pkg client is very poor design.

If access types are used, they should be hidden and encapsulated with their memory management. This makes it easier to get the memory management correct. Since this is what using Unbounded_String does for you, I think it's better to use it than to expend extra effort doing something similar.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 21 Nov 2022 19:29:12 +0200

> I know it, and I deliberately reject it.

I agree in general, but there are design trade-offs that depend on issues not made clear in the original question, such as the size of the file and the performance requirements. So I thought that the OP should know of the heap alternative, even if it has some poorer properties too.

From: Qunying <zhu.qunying@gmail.com>
Date: Mon, 21 Nov 2022 09:29:49 -0800

If you are using GNAT with gnatcoll, then you may try its mmap facility, https://docs.adacore.com/gnatcoll-docs/mmap.html

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Mon, 21 Nov 2022 13:43:39 -0800

You may be interested by this:

https://github.com/zertovitch/zip-ada/blob/master/zip_lib/zip_streams.ads#L148

It can be actually used out of the context of Zip archives.

Ada 2022 in GNAT

From: Simon Belmont <sbelmont700@gmail.com>
Subject: Ada 2022 in GNAT
Date: Thu, 8 Dec 2022 15:37:15 -0800
Newsgroups: comp.lang.ada

Has anyone seen (or willing to type up...) any broad-strokes information about how GNAT (et al) actually plans to implement the parallelization features of Ada 2022? Take advantage of GPUs or just stick to CPU cores, or some kind of binding to OpenMP, etc, on Linux vs Windows vs Vworks, etc? I'm mostly just curious and haven't seen any of that in-the-weeds type information floating around, or at least anything that isn't a few years old.

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Fri, 9 Dec 2022 09:07:50 -0800

> In the past year or so, we have been working hard assessing and implementing most of these Ada 202x changes (called AIs: Ada Issues in ARG terms). The implementation work and feedback from first users allowed us to identify that a few of these features would need additional time and attention. This led us to make a difficult decision - in order to allow for more investigation and to avoid users to start to rely on constructs that may need to change or be replaced, we decided to put on hold the implementation of some of the changes in language. Of course, we're currently engaged with the ARG to discuss these.

> The main set of features that AdaCore and GNAT are putting on hold are related to the support for parallel constructs. While the overall vision is an exciting and promising one, we realized when looking at the state of the art and gathering user requirements that there were a lot more aspects to consider on top of those currently addressed by the AIs. Some of these are related to GPGPU (General Purpose GPU) support as well as their future CPU counterparts, and include topics such as control of memory transfer, precise allocation of tasks and memory on the hardware layout, target-aware fine tuning options as well as various other parametrization needs. These capabilities happen to be fundamental to obtain actual performance benefits from parallel programming, and providing them may require profound changes in the language interface. Consequently, we're putting all parallel AIs on hold, including support for the Global and Nonblocking aspects beyond the current support in SPARK.

See https://blog.adacore.com/ada-202x-support-in-gnat

From: Simon Belmont <sbelmont700@gmail.com>
Date: Sat, 10 Dec 2022 12:03:08 -0800

> See https://blog.adacore.com/ada-202x-support-in-gnat

That post is over two years old, surely that can't still be the state of things? I'm not sure what it says when the big, marquee item of the newest standard isn't even actively being worked on in the big, marquee compiler.

From: Fabien Chouteau <fabien.chouteau@gmail.com>
Date: Tue, 13 Dec 2022 04:10:43 -0800

> That post is over two years old, surely that can't still be the state of things?

There has been progress on the Ada2022 support, but no change on that part.

Conference Calendar

Dirk Craeynest

Department of Computer Science, 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

January 16-18

18th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2023), Toulouse, France. Topics include: software development for high performance parallel systems; tools for compilation, evaluation, optimization of high performance parallel systems (compiler support, tracing, and debugging for parallel architectures, ...); embedded real-time systems, mixed criticality system support, dependable systems, ...; software support for embedded architectures (tracing and real-time analysis of embedded applications, runtime software); etc.

☺ February 04-05

Free and Open source Software Developers' European Meeting (FOSDEM'2023), Brussels, Belgium. FOSDEM 2023 is an international two-day event (Sat-Sun 4-5 Feb), held in Brussels, Belgium. After our 11th Ada DevRoom in 2022 there won't be a new Ada DevRoom in 2023, but there will be a stand during both days of the event with as theme "It's time to learn Ada!"

February 25-26

32nd ACM SIGPLAN **International Conference on Compiler Construction** (CC'2023), Montréal, Québec, Canada. Co-located with CGO, PPoPP, and HPCA. Topics include: processing programs in the most general sense (analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques (program representation, analysis, and transformation; code generation, optimization, and synthesis; the verification thereof); run-time techniques (memory management, virtual machines, and dynamic and just-in-time compilation); programming tools (refactoring editors, checkers, verifiers, compilers, debuggers, and profilers); techniques, ranging from programming languages to micro-architectural support, for specific domains such as secure, parallel, distributed, embedded or mobile environments; design and implementation of novel language constructs, programming models, and domain-specific languages; etc.

Feb 25 – Mar 01

IEEE/ACM **International Symposium on Code Generation and Optimization** (CGO'2023), Montreal, Canada. Topics include: code generation, translation, transformation, and optimization for performance, energy, virtualization, portability, security, or reliability concerns, and architectural support; static, dynamic, and hybrid analysis for performance, energy, memory locality, throughput or latency, security, reliability, or functional debugging; efficient profiling and instrumentation techniques; novel and efficient tools; compiler design, practice, and experience; compiler abstraction and intermediate representations; vertical integration of language features, representations, optimizations, and runtime support for parallelism; deployed dynamic/static compiler and runtime systems for general-purpose, embedded system and Cloud/HPC platforms; compiler-support for vectorization, thread extraction, task scheduling, speculation, transaction, memory management, data distribution, and synchronization; etc.

March 06-10
(h)

25th **International Symposium on Formal Methods** (FM'2023), Lübeck, Germany. Topics include: development and application of formal methods in a wide range of domains including trustworthy AI, software, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, healthcare and biology; techniques, tools and experiences in interdisciplinary settings; experiences of applying formal methods in industrial settings; design and validation of formal method tools; formal methods in practice

(industrial applications of formal methods, experience with formal methods in industry, tool usage reports, experiments with challenge problems); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, and experimental validation of tools); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, and method integration); special FM 2023 session on "Formal methods meets AI" (focused on formal and rigorous modelling and analysis techniques to ensuring safety, robustness etc. (trustworthiness) of AI-based systems); etc.

March 13-17 — 20th IEEE **International Conference on Software Architecture** (ICSA'2023), L'Aquila, Italy. Topics include: architecture evaluation and quality aspects of software architectures; model-driven engineering for architecture; component-based software engineering; automatic extraction and generation of software architecture descriptions; refactoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; linking architecture to requirements and/or implementation; architecture & continuous integration/delivery, and DevOps; training, soft skills, coaching, mentoring, education, and certification of software architects; architecture for legacy systems and systems integration; architecting families of products; roles and responsibilities for software architects; etc.

☺ March 13-17 — **International Conference on the Art, Science, and Engineering of Programming** (Programming'2023), Tokyo, Japan.

March 21-24 — 30th IEEE **International Conference on Software Analysis, Evolution, and Reengineering** (SANER'2023), Macao SAR, China. Topics include: theory and practice of recovering information from existing software and systems; software tools for software evolution and maintenance; software analysis, parsing, and fact extraction; software reverse engineering and reengineering; program comprehension; software evolution analysis; software architecture recovery and reverse architecting; program transformation and refactoring; mining software repositories and software analytics; software reconstruction and migration; software maintenance and evolution; program repair; software release engineering, continuous integration, and delivery; education related to all of the above topics; etc.

Mar 27 – Apr 02 — 38th ACM/SIGAPP **Symposium on Applied Computing** (SAC'2023), Tallinn, Estonia.

- Mar 27 – Apr 02 — **Embedded Systems Track** (EMBS'2023). Topics include: the application of both novel and well-known techniques to the embedded systems development.
- Mar 27 – Apr 02 (v) — 17th **Track on Dependable, Adaptive, and Secure Distributed Systems** (DADS'2023). Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; applications of DADS; etc.

April 05 — **Eelco Visser Commemorative Symposium,** Delft, the Netherlands. Topics include: language engineering, program transformation, language workbenches, declarative language specification, name binding and scope graphs, type soundness and intrinsically-typed interpreters, language specification testing, language implementation generation, domain-specific programming languages, DSLs for software deployment, DSLs for web application development, tool-supported programming education.

April 15-19 — 14th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2023), Coimbra, Portugal. Deadline for submissions: January 7, 2023 (artifact track submission), January 15, 2023 (data challenge, emerging research track, posters, demos), January 22, 2023 (tutorials).

April 16-20 — 16th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2023), Dublin, Ireland. Topics include: manual testing practices and techniques, security testing, model-based testing, test automation, static analysis and symbolic execution, formal verification and model checking, software reliability, testability and design, testing and development processes, testing in specific domains (such as embedded, concurrent, distributed, ..., and real-time systems), testing for cyber-physical systems, testing/debugging tools, empirical studies, experience reports, etc.

April 17-20 — 28th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2023), Barcelona, Catalunya, Spain. Theme: "Human Values in RE".

April 22-27 — 26th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2023), Paris, France. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to

Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems). Deadline for submissions: January 5, 2023 (ESOP, FASE, FoSSaCS artefact submissions), January 16, 2023 (Doctoral Dissertation Award nominations).

April 26-27 29th **International Symposium on Model Checking of Software** (SPIN'2023). Topics include: automated tool-based techniques to analyze and model software for the purpose of verification and validation. Deadline for submissions: January 9, 2023 (abstracts), January 16, 2023 (papers).

April 24-28 26th **Ibero-American Conference on Software Engineering** (CIbSE'2023), Montevideo, Uruguay. Topics include: formal methods applied to software engineering (SE), mining software repositories and software analytics, model-driven SE, software architecture, software dependability, software ecosystems and systems-of-systems, SE education and training, SE for emerging application domains (e.g., cyber-physical systems, IoT, ...), SE in the industry, software maintenance and evolution, software processes, software product lines, software quality and quality models, software reuse, software testing, technical debt management, etc. Deadline for submissions: January 8, 2023 (abstracts), January 16, 2023 (papers), February 6, 2023 (doctoral symposium, journal first).

May 09-12 16th **Cyber-Physical Systems and Internet of Things Week** (CPS-IoT Week'2023), San Antonio, Texas, USA. Event includes: 5 top conferences, HSCC, ICCPS, IoTDI, IPSN, and RTAS, multiple workshops, tutorials, and competitions.

☺ May 09-12 29th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2023). Topics include: systems research related to embedded systems and time-sensitive systems; original systems, applications, case studies, methodologies, and algorithms that contribute to the state of practice in design, implementation, verification, and validation of embedded systems or time-sensitive systems. Deadline for submissions: February 10, 2023 (brief presentations).

May 14-20 45th **International Conference on Software Engineering** (ICSE'2023), Melbourne, Victoria, Australia. Topics include: the full spectrum of Software Engineering. Deadline for submissions: February 1, 2023 (TCSE award nominations).

May 14-15 11th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2023). Topics include: approaches, methods and tools for verification and validation; formal approaches to safety and security related issues; scalability of formal method applications; integration of formal methods within the software development lifecycle; model-based engineering approaches; correctness-by-construction approaches for software and systems engineering; application of formal methods to specific domains, e.g., autonomous, cyber-physical, intelligent, and IoT systems; formal methods in a certification context; case studies developed/analyzed with formal approaches; experience reports on the application of formal methods to real-world problems; guidelines to use formal methods in practice; usability of formal methods; etc. Deadline for submissions: January 9, 2023 (abstracts), January 16, 2023 (papers).

May 16-18 15th **NASA Formal Methods Symposium** (NFM'2023), Houston, Texas, USA. Topics include: challenges and solutions for achieving assurance for critical systems, such as formal verification, including theorem proving, model checking, and static analysis, advances in automated theorem proving including SAT and SMT solving, use of formal methods in software and system testing, techniques and algorithms for scaling formal methods (abstraction and symbolic methods, compositional techniques, parallel and/or distributed techniques, ...), etc.

May 23-25 15th **Software Quality Days** (SWQD'2023), Munich, Germany. Topics include: all topics about software and systems quality, such as improvement of software development methods and processes, testing and quality assurance of software and software-intensive systems, project and risk management, domain specific quality issues such as embedded, medical, automotive systems, novel trends in software quality, etc.

☺ June 07-08 31st **International Conference on Real-Time Networks and Systems** (RTNS'2023), Dortmund, Germany. Topics include: real-time applications design and evaluation (automotive, avionics, space, railways, telecommunications, process control, ...), real-time aspects of emerging smart systems (cyber-physical systems and emerging applications, ...), real-time system design and analysis (real-time tasks

modeling, task/message scheduling, mixed-criticality systems, Worst-Case Execution Time (WCET) analysis, security, ...), software technologies for real-time systems (model-driven engineering, programming languages, compilers, WCET-aware compilation and parallelization strategies, middleware, Real-time Operating Systems (RTOS), ...), formal specification and verification, real-time distributed systems, etc. Deadline for submissions: January 23, 2023 (abstracts 2nd round), January 25, 2023 (papers 2nd round).

♦ June 13-16 27th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC 2023), Lisbon, Portugal. Sponsored by Ada-Europe. In cooperation with ACM SIGAda (pending), and the Ada Resource Association (ARA). Deadline for submissions: January 16, 2023 (journal-track papers), February 27, 2023 (industrial track and work-in-progress papers, tutorials and workshop proposals).

July 17-21 **Software Technologies: Applications and Foundations** (STAF'2023), Leicester, UK. Topics include: practical and foundational advances in software technology. Deadline for submissions: May 21, 2023 (workshop papers).

☺ July 17-21 37th **European Conference on Object-Oriented Programming** (ECOOP'2023), Seattle, USA.

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: March 16, 2023 (Journal track abstracts); March 20, 2023 (workshops, tutorials, education classes, and special sessions proposals); March 23, 2023 (Journal track full papers); May 22, 2023 (Work-in-Progress track papers).

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: February 5, 2023 (workshops), February 6, 2023 (abstracts), February 13, 2023 (full papers).

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. Deadline for submissions: May 15, 2023.

☺ October 23-27 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.

☺ Oct 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. Deadline for submissions: April 14, 2023 (round 2).

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

Conference Chair

António Casimiro
casim@ciencias.ulisboa.pt
University of Lisbon, Portugal

Journal-track Chair

Elena Troubitsyna
elenatro@kth.se
KTH Royal Inst. of Technology, Sweden

Industrial-track Chairs

Alexandre Skrzyniarz
alexandre.skrzyniarz@fr.thalesgroup.com
Thales, France

Sara Royuela
sara.royuela@bsc.es
Barcelona Supercomputing Center, Spain

Work-In-Progress-track Chairs

Bjorn Andersson
baandersson@sei.cmu.edu
Carnegie Mellon University, USA

José Cecílio
jmcecilio@fc.ul.pt
University of Lisbon, Portugal

Tutorial and Education Chair

Luis Miguel Pinho
lmp@isep.ipp.pt
ISEP, Portugal

Workshop Chair

Frank Singhoff
singhoff@univ-brest.fr
University of Brest, France

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

General Information

The **27th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2023)** will take place in Lisbon, Portugal. 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 submissions present research advances supported by solid theoretical foundation and thorough evaluation.
- Industrial-track submissions highlight the practitioners' side of a challenging case study or industrial project.
- Work-in-progress-track submissions illustrate a novel research idea that is still at an initial stage, between conception and first prototype.
- Tutorial submissions guide attenders through a hands-on familiarization with innovative developments or with useful features related to reliable software.

Schedule

16 January 2023	Submission deadline for journal-track papers
27 February 2023	Submission deadline for industrial-track papers, work-in-progress papers, tutorial and workshop proposals
20 March 2023	First round notification for journal-track papers, and notification of acceptance for all other types of submissions
13-16 June 2023	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
- Real-Time Systems
- High-Integrity Systems and Reliability
- Ada Language
- Applications in a variety of domain

More specific topics are described on the conference web page.

Call for Journal-track Submissions

Following a journal-first model, this edition of the conference again includes a journal track, which seeks original and high-quality papers that describe mature research work on the conference topics. Accepted journal-track papers will be published in the "Reliable Software Technologies (AEiC2023)" Special Issue of JSA -- the *Journal of Systems Architecture* (Scimago Q1 ranked, impact factor 5.936).

General information for submitting to the JSA can be found at the *Journal of Systems Architecture* website. The submission link will be available on the conference web page. Contributions must be submitted by **16 January 2023**. JSA has adopted the Virtual Special Issue model to speed up the publication process, where Special Issue papers are published in regular issues, but marked as SI papers. Acceptance decisions are made on a rolling basis. Therefore, authors are encouraged to submit papers early, and need not wait until the submission deadline. Authors who have successfully passed the first round of review will be invited to present their work at the conference. Please note that the AEiC 2023 organization committee will waive the Open Access fees for the first four accepted papers, which do not already enjoy OA from personalized bilateral agreements with the Publisher. Subsequent papers will follow JSA regular publishing track. Prospective authors may direct all enquiries regarding this track to the corresponding chair, Elena Troubitsyna (elenatro@kth.se).

Call for Industrial-track Submissions

The conference seeks industrial practitioner presentations that deliver insight on the challenges of developing reliable software. Especially welcome kinds of submissions are listed on the conference web site. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer review process. Interested authors shall submit a one-to-two pages abstract, by **27 February 2023**, via EasyChair at https://easychair.org/my/conference?conf=aeic2023, selecting the "Industrial Track". The format for submission is strictly in PDF, following the *Ada User Journal* style. Templates are available at http://www.ada-europe.org/auj/guide.

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference and will also be invited to expand their contributions into full-fledged articles for publication in the *Ada User Journal*, which will form the proceedings of the industrial track of the Conference. Prospective authors may direct all enquiries regarding this track to its chairs Alexandre Skrzyniarz (alexandre.skrzyniarz@fr.thalesgroup.com) and Sara Royuela (sara.royuela@bsc.es).

Call for Work-in-Progress-track Submissions

The work-in-progress track seeks two kinds of submissions: (a) ongoing research and (b) early-stage ideas. Ongoing research submissions are 4-page papers describing research results that are not mature enough to be submitted to the journal track. Early-stage ideas are 1-page papers that pitch new research directions that fall within the scope of the conference. Both kinds of submissions must be original and shall undergo anonymous peer review. Submissions by recent MSc graduates and PhD students are especially sought. Authors shall submit their work by **27 February 2023**, via EasyChair at https://easychair.org/my/conference?conf=aeic2023, selecting the "Work in Progress Track". The format for submission is strictly in PDF, following the *Ada User Journal* style. Templates are available at http://www.ada-europe.org/auj/guide.

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference and will also be offered the opportunity to expand their contributions into 4-page articles for publication in the *Ada User Journal*, which will form the proceedings of the WiP track of the Conference. Prospective authors may direct all enquiries regarding this track to the corresponding chairs Bjorn Andersson (baandersson@sei.cmu.ed) and José Cecílio (jmcecilio@fc.ul.pt).

Call for Tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic or practitioner slant, including hands-on or practical elements. Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half-day or full-day), the intended level of the contents (introductory, intermediate, or advanced), and a statement motivating attendance. Tutorial proposals shall be submitted by e-mail to Tutorial and Education Chair, Luís Miguel Pinho (lmp@isep.ipp.pt), with subject line: "[AEiC 2023: tutorial proposal]". Tutorial proposals shall be submitted by **27 February 2023**. The authors of accepted full-day tutorials will receive a complimentary conference registration, halved for half-day tutorials. The *Ada User Journal* will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

The conference welcomes satellite workshops centred on themes that fall within the conference scope. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the AEiC conference. Workshop organizers shall also commit to producing the proceedings of the event, for publication in the *Ada User Journal*. Workshop proposals shall be submitted by e-mail to the Workshop Chair, Frank Singhoff (singhoff@univ-brest.fr), with subject line: [AEiC 2023: workshop proposal]. Workshop proposals shall be submitted at any time but no later than the **27 February 2023**. Once submitted, each workshop proposal will be evaluated by the conference organizers as soon as possible.

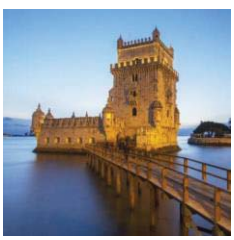
Call for Exhibitors

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

Venue

The conference will take place at the Hotel Fénix Lisboa, near downtown Lisbon, Portugal. June is full of events in Lisbon, including the festivities in honour of St. António (June 13 is the town holiday), with music, grilled sardines, and popular parties in Alfama and Bairro Alto neighbourhoods. There's plenty to see and visit in Lisbon, so plan in advance!



AEiC 2023
Lisboa



New process for commenting on the Ada Language Standard

The Ada Rapporteur Group (ARG) of Working Group 9 (ISO WG9), within ISO/IEC JTC1 SC22, is responsible for maintaining and advancing the International Ada Programming Language Standard. In the past we have used an "ada-comment" mailing list as the official place to file comments or suggestions about the Standard. Over the past six months we have been moving to an online approach using a website and a GitHub issue repository. The new process seems to be working, so we ask all those with comments about the Ada Standard or the Ada Reference Manual to visit the new ARG website:

https://arg.adaic.org/

and select the "Community Input" page. There you will find forms for filing comments, or for requesting the formation of a "language study group" to focus on particular thorny topics associated with the language (an example might be "distributed computing" or "tree pattern matching").

Rather than filling out a form, you can head over to the ARG GitHub repository:

https://github.com/Ada-Rapporteur-Group/User-Community-Input

and select the GitHub "issue" tab to post your comments, or join a discussion on issues already there.

We would also welcome "meta" comments if you have thoughts on how to improve the ARG process itself.

Ada Rapporteur Group

Ada User Journal

Call for Contributions

Topics: **Ada, Programming Languages**, **Software Engineering Issues** and **Reliable Software Technologies** in general.

Contributions: **Refereed Original Articles**, **Invited Papers**, **Proceedings** of workshops and panels and **News and Information** on Ada and reliable software technologies.

More information available on the
Journal web page at

http://www.ada-europe.org/**auj**

Online archive of past issues at http://www.ada-europe.org/auj/archive/

Report on the ASIS Birds-of-a-Feather Session: The Future of ASIS and Vendor Independent Tools

J-P. Rosen

Adalog, 2 rue du Docteur Lombard, 92130 Issy-Les-Moulineaux, FRANCE; email: rosen@adalog.fr

Abstract

This paper reports on the informal BoF-session that took place on the last day of the 26th International Conference on Reliable Software Technologies (AEiC 2022) on June 17th in Ghent, Belgium.

This session gathered people interested in the Ada Semantic Interface Specification (ASIS) to discuss the current situation of the standard and its future.

Keywords: Ada, static analysis, ASIS.

1 Introduction

Ada is a complex language to compile and analyze; ASIS is an international standard that alleviates the burden of writing third-party program analysis tools by providing access to the syntactic tree built by the compiler with an API for traversing the tree, syntactic and semantic queries, etc. Several tools and utilities have been designed that build on ASIS.

The ASIS standard has not been updated beyond the Ada 95 version of the language. Some vendors support strictly the standard, hence only Ada 95 technology. Others vendors have implemented extensions for later versions of Ada. On the whole, the long-term support of ASIS is uncertain.

This situation raises concerns for third-party tool developers and users. The session aimed at gathering tool users, tool writers, compiler vendors, and other interested parties into an informal meeting to discuss the following topics:

- user expectations on existing and future program analysis tools;
- opportunities for updating the ASIS standard;
- support for ASIS (by vendors or volunteers);
- viability of abandoning ASIS to migrate to different libraries.

2 Presentation of ASIS

J-P. Rosen introduced the session by giving an overview presentation of ASIS.

ASIS (*Ada Semantic Interface Specification*) [1] is an international standard, first developed for Ada 83, then updated to Ada 95. It was developed by an international committee (the ASIS working group). The standard was not evolved for subsequent updates of Ada, however implementations, especially the AdaCore one, continued to add support up to Ada 2012. AdaCore announced however that their implementation would not be upgraded to support Ada 2022.

ASIS is an API to explore and get information from the decorated syntactic tree, as produced by the associated compiler. This guarantees that a tool based on ASIS will see the code exactly as the compiler sees it, including implementation dependent elements allowed by the standard, and elements defined in the System and Standard packages. However, this implies that an ASIS implementation is linked to a certain compiler. A tool based on ASIS must provide specific executable versions for each supported compiler.

Since ASIS operates on a tree resulting from a successful compilation, it cannot handle incorrect or incomplete code. For the same reason, it was a deliberate design decision to *not* provide any operation that would modify, or even add information, to the syntactic tree. It is purely oriented towards analysing a program, with no way to modify it.

More information about ASIS can be found in [2]

3 Vendor Presentations

Vendors of ASIS solutions were invited before the conference to present their offer.

AdaCore declined, having no representative available at that time, and PTC did not attend the conference. However, a participant stated that at PTC's latest webinar on Ada tools, PTC said they had a full ASIS implementation of the standard (i.e. ASIS-95), and that they would consider implementing ASIS for later versions of Ada if a new ASIS standard were to become available.

4 User presentations

Adalog was the only participant who developed a full tool with ASIS, AdaControl, probably the most demanding ASIS application. Information about AdaControl can be found at [3].

5 Discussion

There was some discussion about the official and practical state of ASIS.

One issue was the confusion in earlier discussions in the comp.lang.ada newsgroup, caused by a misunderstanding about what it means that ISO holds the copyright on the ASIS standard document. Some people there claimed that

due to the ISO copyright, no one can legally implement nor even use ASIS without paying money to ISO [sic].

It was clarified that this obviously is not correct, which is supported by the publicly available portions of the ASIS standard document [4]. In fact, the standard explicitly allows implementations and even extensions: see sections '1.1.3 Conformity with this International Standard' and '1.1.4 Implementation permissions'.

Another issue was that although the current ASIS standard is for Ada 95, AdaCore's implementation has evolved and now fully supports Ada 2012. As most of the Ada 2022 updates are aspects, there is little new syntax. Therefore, to update the ASIS standard for Ada 2022 would be relatively simple, and include the following changes:

- build from experience, add obviously missing semantic queries from the current ASIS standard;
- straighten semantics in some insufficiently or incorrectly defined cases and remove some implementation permissions to improve portability of applications;
- update to Ada 2012, building upon existing AdaCore extensions;
- update to Ada 2022.

Lobbying might be required to convince vendors to support a new standard, but see above for PTC. AdaCore stated that they were not interested to support ASIS any further, therefore volunteers are needed to support ASIS in FSF GNAT. Another possibility is to provide a vendor independent implementation, like the former Gela ASIS [5], or by making an ASIS layer on top of Libadalang [6], or via the database approach presented at the conference by the poster of Quentin Dauprat.

6 Actions

Since the discussion showed that there was clearly an interest in continuing ASIS, each participant took some action:

- a participant said that they would speak to those in their work environment about using AdaControl and this could produce a demand for ASIS;
- a participant said they would also discuss with their work colleagues with a view to using AdaControl; they would also like to implement ASIS in the HAC compiler [7] and will contact PTC to gauge their interest in progressing ASIS;
- a participant suggested that ASIS could be supplied via Alire by a community effort;
- a participant stated that they would raise this at work;
- a participant said they are willing to coordinate the effort with volunteers in an ASIS working group and lead the possible development of a new ASIS standard;
- a participant said they would implement some static analysis in Libadalang and ASIS/AdaControl as a comparison;
- a participant said they would contact Ada-Switzerland members to gauge which ones were making use of ASIS;
- A participant will undertake some further technical development

7 Conclusion

The session showed that there was a clear interest in pursuing ASIS, both from a user and from an industrial point of view. However, this would make sense only with an updated ASIS standard.

Volunteers have stepped forward to start such an effort. After the meeting, a mailing list has been established to coordinate among participants. All interested parties are invited to send a mail to rosen@adalog.fr to be added to that list.

References

[1] ISO/IEC 15291: Information technology — Programming languages — Ada Semantic Interface Specification (ASIS)

[1] J-P. Rosen, "ASIS vs. LibAdalang: A Comparative Assesment", Ada User Journal, Volume 42 numbers 3-4, September - December 2021.

[2] https://adacontrol.fr/

[3] https://webstore.iec.ch/preview/info_isoiec15291%7Bed1.0%7Den.pdf

[4] https://github.com/faelys/gela-asis

[5] https://github.com/AdaCore/langkit-query-language/blob/master/user_manual/source/language_reference.rst

[6] https://hacadacompiler.sourceforge.io/

ANIARA Project - Automation of Network Edge Infrastructure and Applications with Artificial Intelligence

Wolfgang John, Ali Balador, Jalil Taghia, Andreas Johnsson, Johan Sjöberg
Ericsson, Stockholm, Sweden; email: {wolfgang.john, ali.balador, jalil.taghia, andreas.a.johnsson, johan.sjoberg}@ericsson.com

Ian Marsh, Jonas Gustafsson
RISE Research Institute of Sweden, Stockholm, Sweden; email: {ian.marsh, jonas.gustafsson}@ri.se

Federico Tonini, Paolo Monti
Chalmers University, Sweden; email: {tonini, mpaolo}@chalmers.se

Pontus Sköldström
Qamcom AB, Sweden; email: {pontus.skoldstrom}@qamcom.com

Jim Dowling
Hopsworks AB, Sweden; email: jim@hopsworks.ai

Abstract

Emerging use-cases like smart manufacturing and smart cities pose challenges in terms of latency, which cannot be satisfied by traditional centralized infrastructure. Edge networks, which bring computational capacity closer to the users/clients, are a promising solution for supporting these critical low latency services. Different from traditional centralized networks, the edge is distributed by nature and is usually equipped with limited compute capacity. This creates a complex network to handle, subject to failures of different natures, that requires novel solutions to work in practice. To reduce complexity, edge application technology enablers, advanced infrastructure and application orchestration techniques need to be in place where AI and ML are key players.

1 Introduction

ANIARA project is based on two use case families addressing smart manufacturing and smart cities. Figure 1 illustrates the functional technologies for evolving 5G edge systems. The application scenarios are drawn upon the synergies between 5G, cloud computing and edge computing. Our studies include elaborating detailed use case descriptions for verticals using 5G and the edge cloud. Concerning manufacturing, ANIARA focuses on two main aspects [1]. These are: environmental monitoring and control of the factory floor concerning properties such as air quality; temperature and power management and operations monitoring and control such as robot cell control, logistics and safety. These use cases drive the high-level system architecture requirements and serve as the basis for technology-specific use cases such as private 5G, edge micro datacenters and AI at the edge.

The remainder of this paper is organized to provide additional details of the components in Figure 1. Section 2 provides details about edge platform infrastructure and services that have been developing in ANIARA, including lightweight and portable execution environments and fast, dependable feature stores. Apart from edge infrastructure, developing edge AI enablers is also another important objective of the ANIARA. Section 3 describes AI/ML methods designed and developed within the project so far, including ML models for life-cycle management, intelligent feature selection and distributed learning in the edge considering privacy. ANIARA also addresses management and orchestration including both edge and cloud scenarios, to satisfy the specific needs of the use cases, detailed in Section 4. Section 5 presents research on smart power for building a large-scale 5G edge system, shown in the lower portion of Figure 1. Section 6 provides a comprehensive related work. Finally, Section 7 gives an overview of works planned to be done in the future.

2 Edge platform components

2.1 Programmable, light containerisation

Lightweight and portable execution environments have been identified as a crucial enabler for higher flexibility and dynamism of application deployment in a distributed network [2, 3]. In ANIARA, we experiment with WebAssembly technologies to fill this gap. WebAssembly [4] is an *open* binary instruction format for a stack-based virtual machine, designed to support existing programming languages in a web browser environment. As an example, Edgedancer, offers infrastructure support for portable, provider-independent, and secure

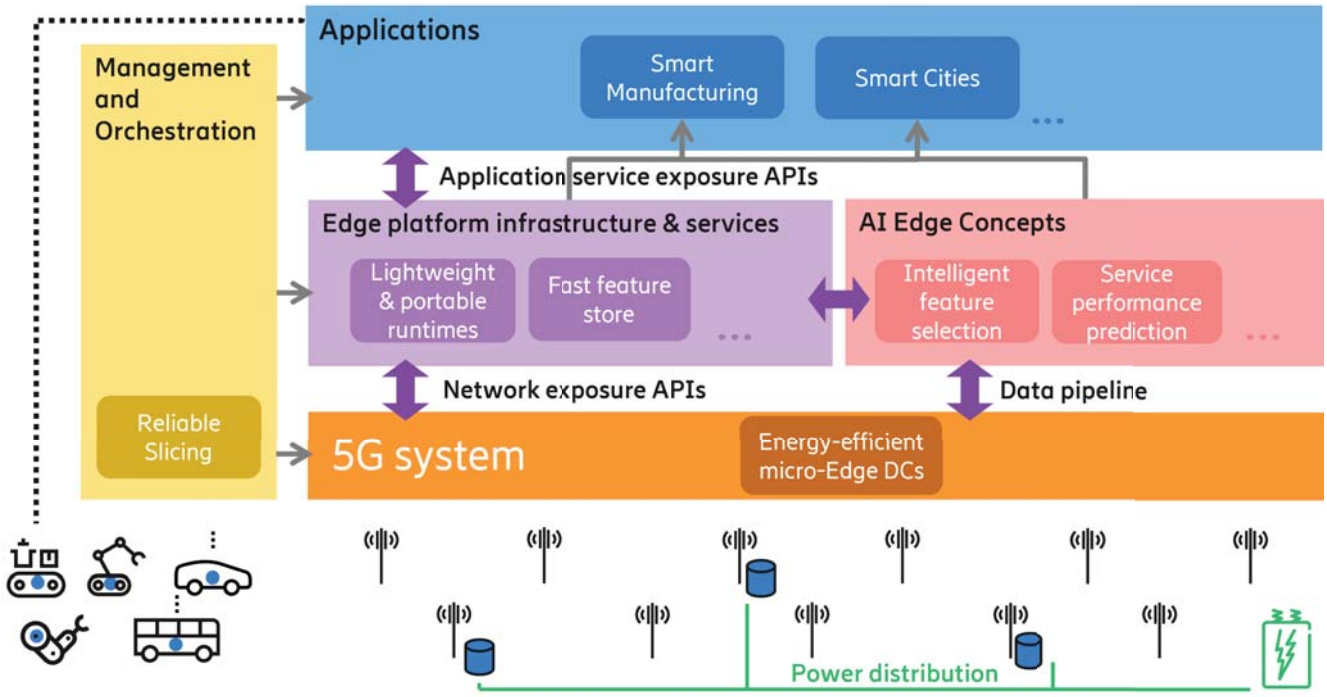


Figure 1: Functional structure of the ANIARA project.

migration of edge services, it is a lightweight and generic execution environment by utilising WebAssembly [5]. In the browser, WebAssembly is generally faster than JavaScript due to its more compact format and manual memory management. The virtual machine is, however, not bound to browsers, but can be used standalone on various platforms. A deployment flow is where a high-level language such as C, C++, Rust, Python and so on can be compiled to WebAssembly (using clang) and executed on various hardware or software platforms. Webassembly allows edge applications to execute within a wide range of devices and operating systems. In ANIARA, we wrote an application in Rust and executed it on a WebAssembly runtime using a $30 microcontroller, to prove the versatility and low footprint. The same application was also written in Python that runs (on WebAssembly again) in an ASCII terminal.

2.2 Fast, dependable, feature store

A production-grade AI solution for edge computing, known as Feature stores, is a part of ANIARA. They act as a halfway house between data scientists and data engineers, enabling the same feature computation code to be used for model training and inference. Feature stores also act as a centralized repository of AI models and have been adapted to perform in distributed scenarios for real-time data synchronization between geographically distributed Edge locations.

The underlying database is called RonDB, evaluated using **LATS**, meaning low **L**atency, high **A**vailability, high **T**hroughput, scalable **S**torage principles to list RonDB's performance:

- RonDB x3 times lower latency
- RonDB 1.1M reads/sec, REDIS 800k reads/sec
- RonDBs processed 200M reads in a 30 node cluster

3 AI for Edge Management

We divide AI for edge management into three main research tracks: (i) ML model life-cycle management, (ii) Intelligent feature selection, and (iii) Distributed learning in the edge. The three tracks have been studied within a use case for service performance prediction from edge statistics available to an operator with the objective to automate parts of the edge management process.

ML model life-cycle management: Ensuring high-performing ML models that are continuously updated and correctly deployed is critical for successful integration of ML in the edge. Transfer learning [6, 7], which is one approach, allows for structured incorporation of previously acquired knowledge enabling timely and robust model adaptation, especially when data are scarce for reliable training of ML models. In ANIARA, we studied concepts and methods for adapting and selecting ML models for mitigating ML-model performance degradation due to expected vertical and horizontal scaling in the edge infrastructure and 5G system, with specific focus on performance models for networked services [8,9,10].

Intelligent feature selection: A challenge for AI at the edge is related to network overhead with respect to measurements and monitoring, and feature selection for improved model performance [11]. A key enabler for ML models is timely access to reliable data, in terms of features, which require pervasive measurement points throughout the network. However, excessive monitoring is associated with network overhead. Using domain knowledge provides hints to find a balance between overhead reduction and identifying future ML requirements. A review on techniques for unsupervised feature selection is provided in [7] and authors in [12] show a comprehensive review of online feature selection techniques. In ANIARA, We implemented an unsupervised feature selection method

that uses a structured approach in incorporation of the domain knowledge acquired from domain experts or previous learning experiences [13].

Distributed learning in the edge: Multi-domain service metric prediction is a key component in the ANIARA framework. It should enable privacy-preserving sharing of knowledge between operators, and low-overhead training of models within an operator in terms of data sharing. The approach requires in-network processing capabilities to enable federated learning. The works in [14, 15] review federated learning methods in mobile edge networks and edge computing, respectively. Our work in ANIARA extends the scope of these works specifically towards edge clouds for telecommunication industry. We are working on a multi-domain service metric prediction framework using federated learning, corresponding to a scenario where several services are managed by a number of operators in geographically distributed locations.

4 Edge-cloud orchestration

Slicing allows provisioning of multiple services over the same infrastructure, where virtual or physical resources are interconnected to form end-to-end logical networks, also known as slices [16]. Orchestrators run resource allocation algorithms to select the most suitable set of resources to satisfy the specific needs of the clients. In the edge cloud, compute resources are located close to the users, allowing provisioning of low latency services and enabling 5G Ultra-Reliable Low-Latency Communication slices. Allocating backup resources requires protecting the slices against link or node failures. Backup resources can be provided by means of a dedicated protection scheme, where resources are dedicated for each slice. Since backup resources are accessed only in case of failures, shared protection schemes can be developed, where backup resources are shared among different slices to decrease the overall amount of required resources.

Resource allocation strategies for 5G networks and reliable services have been investigated recently. In particular, different techniques for backup protection of optical network resources, relying on both DP and SP schemes, have been presented in [17, 18]. Works propose efficient DP and SP algorithms for cloud and baseband resources in 5G access and metro networks, see [19, 20]. Considering connectivity and compute resources separately may lead to impractical solutions, especially when resources are scarce. Our work focuses on the dynamic slice provisioning where both type of resources are jointly allocated. A whitepaper presents an overview of the market and implementation trends, see [21].

In ANIARA, we developed a heuristic-based shared protection to encourage sharing of backup connectivity and cloud resources. We also evaluated this against a dedicated protection scheme using a Python simulator, published in [22]. Results show that the shared approach reduces the blocking probability by order of magnitude, and is especially beneficial when in-node processing resources are scarce.

5 Edge-cloud power research

Installing thousands of edge data centers, primarily in cities will require significant amounts of power, however many power grids are already utilized close to 100%. To maintain high availability, the edge-data centers need to be complemented with alternative power sources and pro-active power management systems. This requires tailored hardware solutions integrated with the power grid and on-site power generation. Supporting active load balancing by going off-grid for shorter periods of time. We are working on the design and implementation of a series of micro-edge-data center demonstrators for deployment at industrial sites. The first generation consists of a double rack configuration including, cooling, UPS-system with batteries, multiple power source inputs and IT-hardware.

To build out a large-scale 5G edge system, smart power utilization is required. One approach is to utilize the on-site UPS installation to go off-grid during peak power periods. The battery storage needs to be dimensioned to address this active usage. Incentivizing the active participation from the edge data centers in the load-balance activity is necessary. Moving away from fixed to dynamic prices will permits battery charging during off-peaks and discharging during higher-priced periods. Discharging implies less grid power. The value for a power grid operator will be larger than that reflected by the customer energy price, if the data center is placed in a particularly energy-hungry section of the grid. Therefore, we have initiated a dialog with a major power grid operator.

6 Future work

Going forward in the AI for edge field, we will study distributed learning under various sources of data and system heterogeneity. The objective is to tackle concerns with data privacy, resource heterogeneity among AI actors, challenges in re-usability of previously learned ML models, and difficulties in effective incorporation of domain knowledge. Experiments with WebAssembly based runtimes to implement *code-once, execute anywhere* approaches across the device-edge-cloud continuum are ongoing. The idea is to support offloading applications from the user equipment device to the edge node. Future work for the Feature store is a Kubernetes operator for RonDB and using it to store and serve our WASM containers. Improving RonDB and implementing an *evaluation store*, feature drift detection is planned. Before a widespread edge data center can be used, we will need to work on power integration aspects that means deployment/installation of the physical hardware. Installation at hard-to-reach places, requiring easy assembly on-site and an autonomous operation with minimal on-site maintenance is ongoing (a demo was shown at the mid-term review, April 2022). We will also investigate the potentials and limitations of resource sharing in bare metal deployments of containers, and enhanced scaling strategies to improve utilization.

Acknowledgements

This work was supported by EU Celtic Plus (ID C2019/3-2), Vinnova (under the project ID 2020-00763), Bundesministerium für Bildung und Forschung (under the name "AI-NET ANIARA 16KIF1274K") and InnovateUK (under the project ID 106197: ukANIARA) via the ANIARA project.

References

[1] A. Y. Ding *et al.*, "Roadmap for edge AI: A dagstuhl perspective," *CoRR*, vol. abs/2112.00616, 2021.

[2] G. Wikström *et al.*, "6g – connecting a cyber-physical world: A research outlook towards 2030," *Ericsson, White paper*, Feb. 2022.

[3] A. Sefidcon, W. John, M. Opsenica, and B. Skubic, "The network compute fabric – advancing digital transformation with ever-present service continuity," *Ericsson Technology Review*, June 2021.

[4] A. Haas *et al.*, "Bringing the web up to speed with webassembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, (New York, NY, USA), p. 185–200, Association for Computing Machinery, 2017.

[5] M. Nieke, L. Almstedt, and R. Kapitza, "Edgedancer: Secure mobile webassembly services on the edge," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '21, (New York, NY, USA), p. 13–18, Association for Computing Machinery, 2021.

[6] K. R. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, pp. 1–40, 2016.

[7] S. Solorio-Fernández, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "A review of unsupervised feature selection methods," *Artificial Intelligence Review*, vol. 53, pp. 907–948, 2019.

[8] M. E. F. G. Sanz and A. Johnsson, "Exploring approaches for heterogeneous transfer learning in edge clouds," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

[9] H. Larsson, J. Taghia, F. Moradi, and A. Johnsson, "Source selection in transfer learning for improved service performance predictions," in *2021 IFIP Networking Conference and Workshops*, 2021.

[10] F. Moradi, R. Stadler, and A. Johnsson, "Performance prediction in dynamic clouds using transfer learning," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[11] X. Wang, F. S. Samani, A. Johnsson, and R. Stadler, "Online feature selection for low-overhead learning in networked systems," in *2021 17th International Conference on Network and Service Management (CNSM)*, pp. 527–529, IEEE, 2021.

[12] X. Hu, P. Zhou, P. Li, J. Wang, and X. Wu, "A survey on online feature selection with streaming features," *Frontiers of Computer Science*, vol. 12, pp. 479–493, 2016.

[13] J. Taghia, F. Moradi, H. Larsson, X. Lan, M. Ebrahimi, and A. Johnsson, "Policy-induced unsupervised feature selection: A networking case study," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, 2022.

[14] W. Y. B. L. et. al, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, pp. 2031–2063, 2020.

[15] Q. Xia, W. Ye, Z. Tao, J. Wu, and Q. Li, "A survey of federated learning for edge computing: Research problems and solutions," *High-Confidence Computing*, 2021.

[16] A. A. Barakabitze, A. Ahmad, R. Mijumbi, and A. Hines, "5G network slicing using SDN and NFV: A survey of taxonomy, architectures and future challenges," *Computer Networks*, vol. 167, p. 106984, 2020.

[17] N. Shahriar, S. Taeb, S. R. Chowdhury, M. Zulfiqar, M. Tornatore, R. Boutaba, J. Mitra, and M. Hemmati, "Reliable slicing of 5G transport networks with bandwidth squeezing and multi-path provisioning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, 2020.

[18] A. Marotta, D. Cassioli, M. Tornatore, Y. Hirota, Y. Awaji, and B. Mukherjee, "Reliable slicing with isolation in optical metro-aggregation networks," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3, 2020.

[19] B. M. Khorsandi, F. Tonini, and C. Raffaelli, "Centralized vs. distributed algorithms for resilient 5G access networks," *Photonic Network Communications*, vol. 37, pp. 376–387, Jun 2019.

[20] H. D. Chantre and N. L. Saldanha da Fonseca, "The location problem for the provisioning of protected slices in NFV-based MEC infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 7, pp. 1505–1514, 2020.

[21] T. L. F. Project, "State of the edge 2021: A market and ecosystem report for edge computing," whitepaper, 2021.

[22] E. Amato, F. Tonini, C. Raffaelli, and P. Monti, "A resource sharing method for reliable slice as a service provisioning in 5G metro networks," in *2021 International Conference on Optical Network Design and Modeling (ONDM)*, pp. 1–3, 2021.

DAIS Project - Distributed Artificial Intelligence Systems: Objectives and Challenges

Ali Balador, Sima Sinaei

RISE Research Institute of Sweden, Vasteras, Sweden; email: {ali.balador, sima.sinaei}@ri.se

Mats Pettersson

Mobilvägen 10, 223 62 Lund, Sweden; email: Mats.Pettersson@sensative.com

İlhan Kaya

Organize Sanayi Bolgesi, 45030 Manisa, Turkey; email: ilhan.kaya@vestel.com.tr

Abstract

DAIS is a step forward in the area of artificial intelligence and edge computing. DAIS intends to create a complete framework for self-organizing, energy efficient and private-by-design distributed AI. DAIS is a European project with a consortium of 47 partners from 11 countries coordinated by RISE Research Institute of Sweden.

Keywords: Edge Computing, Distributed AI, Federated Learning, DAIS, KDT JU, EU project.

1 Introduction

In recent years, technological developments in consumer electronics and industrial applications have been advancing rapidly. More and smaller, networked devices are able to collect and process data anywhere. This Internet of Things (IoT) is a revolutionary change for many sectors like building, automotive, digital industry, energy, healthcare, etc. As a result, the amount of data being generated at the Edge level has and will increase dramatically, resulting in higher network bandwidth requirements. In the meantime, with the emergence of novel applications, such as automated driving, lower latency of the network is required.

The new paradigm of edge computing (EC) provides new solutions by bringing resources closer to the user, keeps sensitive & private data on device, and provides low latency, energy efficiency, and scalability compared to cloud services while reducing the network bandwidth. This in addition brings cost savings. EC guarantees the quality of service when dealing with a massive amount of data for cloud computing [1, 2].

At the same time, Artificial Intelligence (AI) applications based on machine learning (especially deep learning algorithms) are being fuelled by advances in models, processing power, and big data. In Cisco annual report (2018-2023) [3], they asked 83 organizations when all reported that they have edge computing use cases where artificial intelligence, internet of things and 5G had higher portions. The huge increase of devices at the network edge drives the need for enterprises to manage and analyze data from IoT endpoints. Shifting traffic from the network core to the edge affects computing and communications architectures. To have a successful edge computing strategy, it is important to make sure the overall infrastructure is efficient, manageable.

The developments of AI applications mostly require processing of data in centralized cloud locations and hence cannot be used for applications where milliseconds matter or for safety-critical applications. For example, as the sensors and cameras mounted on an autonomous vehicle generate about a gigabyte of data per second, it is difficult, if not impossible to upload this data and get instructions from the cloud in real-time. Same situation is valid for face recognition applications, where they have high temporal requirements for processing either online or offline. Moreover, edge computing offers security benefits due to wider data distribution at the edge level. Reducing the distance data has to travel for processing means decreasing the opportunities for trackers and hackers to intercept it during transmission and preserves its privacy. With more data remaining at the edges of the network, central servers are also less likely to become targets for cyber attacks. This has led to a growing interest in Federated Learning (FL), as a promising distributed learning paradigm that allows multiple parties to jointly train a global ML model on their combined data without any participants having to reveal their data to a centralized server.

The concept of Federated Learning was developed by Google researchers in 2016, as a promising solution for addressing the issues of communication costs, data privacy, and legalization [4, 5, 6]. This ensures the privacy of data during the training process. An edge server or cloud server periodically gathers the trained parameters to create and update a better and more accurate model, which is sent back to the edge devices for local training. Generally, there are three main steps in the FL training process. 1) Central server shares an initial model. 2) Participants train their local data with the initial model and share the local model with the central server. 3) Central server aggregates the local models and shares the global model with participants.

Federated learning methods have been deployed in practice by major companies [7, 8] and play a critical role in sup-

porting privacy-sensitive applications where training data are distributed at the edge [9]. FL has several distinct advantages over traditional centralized ML training:

- Training time is reduced. Multiple devices are used to calculate gradients in parallel, which offers significant speedups.
- Inference time is reduced. At the end of the day, each device has its own local copy of the model, so it can make predictions extremely quickly without relying on slow queries to the cloud.
- Privacy is preserved. Uploading sensitive information to the cloud presents a significant privacy risk for applications like healthcare devices. Privacy breaches in these settings may be a matter of life and death. As such, keeping data local helps preserve the privacy of end-users.
- Collaborative learning using FL is easy and consumes less power as the models are trained on edge devices. The term implies that edge computing is a suitable environment for using FL. It is a technology that enables the training of ML models on mobile edge networks. Therefore, the communication costs, privacy, security, and legalization issues could be alleviated by leveraging FL in the EC paradigm.

2 DAIS Project Objectives

DAIS is a huge step forward in the area of artificial intelligence and edge computing. DAIS aims at providing edge computing architecture, including both hardware and software for industrial applications. DAIS is a pan-European effort spanning three years with a total budget of €33 million and a consortium of 47 partners from 11 countries, shown in Figure 1. Coordinated by Ali Balador from RISE, research institute of Sweden, and with the support of Europe's industry, Europe's leading Research Organizations, the European Union via the KDT Joint Undertaking and the participating national funding agencies, it is possible to bring together European and International key players to the benefit of Europe's economy and society.

Figure 1: DAIS project consortium.

DAIS intends to create a complete framework for self-organizing, energy efficient and private-by-design distributed AI. The framework covers an end-to-end system consisting of a variety of heterogeneous nodes ranging from simple IoT nodes to high performance servers and cluster nodes. The framework consists of a hardware framework and a software framework enabling both traditional end-to-end services as well as horizontal systems with multiple service providers using common data, IoT and AI resources, shown in Figure 2. Multiple connected nodes using the DAIS framework creates a complete DAIS System supporting both vertically and horizontally distributed AI. The DAIS framework will also support open standards to interact with other devices, platforms, and services. In DAIS, we leverage on existing hardware devices, refine them as well as develop new devices.

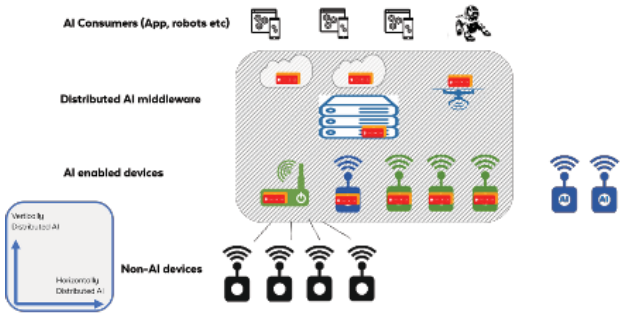


Figure 2: Complete DAIS ecosystem.

The DAIS SW framework includes a DAIS AI framework, a DAIS security framework, and a DAIS communication framework including SW components, tools, policies, interfaces, and data-models enabling compatibility, privacy, and secure data communication between products and services creating a fully distributed AI system. The DAIS framework will allow for standardised data pipelines, sharing compute load between nodes, model distribution as well as model learning hierarchies such as federated learning. Furthermore, providing a standardised way of sharing, keeping track of, and distributing model specific configurations, data and model architectures in a secure manner. The framework is producer-, generator-, vendor-and service-agnostic, hence allowing interaction between the different nodes in vertical and horizontal way.

Within the project, we address many fundamental distributed AI challenges in several value chains and across the technology stack. Here, we will only present some examples. For example, we will explore novel hardware architectures for neuromorphic computing. The software framework will allow the deployed AI algorithms to adapt to changes in the system's environment,e.g., by replacing machine learning models. We will explore ways of developing models in nodes and close to the sensor. The framework will allow data direction agnostic pipeline construction hence enabling development of novel data mining and -analysis solutions, upon which the AI can act. We will also investigate anomaly detection techniques.

An orthogonal aspect is efficiency. In DAIS, we both address energy and communication efficiency. We address these challenges both on a hardware level and software level. For example, we develop technologies for analysing data close to the sensor so that raw data does not have to be communicated

to other nodes. Thus, reducing energy consumption as less data is communicated and communication is expensive in terms of energy. As a side effect, privacy is improved as extracted, and more anonymous, features can be communicated between nodes.

3 DAIS Application Domains

DAIS project covers three domains including digital industry (e.g., condition monitoring for smart grids, AI-driven Automated Guided Vehicle (AGV), optimal regulation of turbines in hydropower plants and frequency converter as a node for edge computing), digital life (e.g., privacy preserving distributed personal TV recommendation system, smart lighting solutions and edge AI in office environment) and transport and smart mobility (e.g., fire monitoring and firefighting system for drones, self-provisioning of Drone fleet for the transportation of goods).

These application domains cover a vast space of applications, hence they aim to tackle a lot of challenges. These challenges can be solved with a smart utilization of different software and hardware components combining the power of artificial intelligence (AI) with internet of things (IoT). One challenge is that application of digital twins into different industrial applications requires obtaining vast amount of real time data from IoT devices usually within a time critical way and examine the data to optimize the system parameters and generate smart actions to improve the energy efficiency, utilization and maintenance of those systems.

For digital life applications, fragmentation and power requirements of the devices and user's comfort and privacy when it comes to processing of their data presents several confrontations to the service providers.In a more traditional way where the data is processed on the cloud systems could breach the security and privacy and considered to be risky. So the solution in these contexts would be to deploy privacy by design principles and move the components for artificial intelligence models instead of the data, and process it where it is produced, e.g. user's devices.

Final application area in DAIS project is to leverage smart edge AI principles to offer smarter ways for transport i.e drones and fleets. One of the major challenges here is to obtain a lot of data and to process it without any delay for self provisioning, detecting and avoiding obstacles while transporting goods, monitoring the accidents and fires. Freshness of the data for these applications are vital as the dynamic nature of the environment will impose several restrictions on feasibility and usefulness of these edge AI methods.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] A. Y. Ding, E. Peltonen, T. Meuser, A. Aral, C. Becker, S. Dustdar, T. Hiessl, D. Kranzlmüller, M. Liyanage, S. Magshudi, N. Mohan, J. Ott, J. S. Rellermeyer, S. Schulte, H. Schulzrinne, G. Solmaz, S. Tarkoma, B. Varghese, and L. C. Wolf, "Roadmap for edge AI: A dagstuhl perspective," *CoRR*, vol. abs/2112.00616, 2021.

[3] "Cisco Annual Internet Report (2018–2023) White Paper." `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`, 2020. [Online; accessed 19-February-2022].

[4] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[5] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[6] P. M. Mammen, "Federated learning: Opportunities and challenges," *arXiv preprint arXiv:2101.05428*, 2021.

[7] H. G. Abreha, C. J. Bernardos, A. D. L. Oliva, L. Cominardi, and A. Azcorra, "Monitoring in fog computing: state-of-the-art and research challenges," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 36, no. 2, pp. 114–130, 2021.

[8] X. Zhou, W. Liang, J. She, Z. Yan, I. Kevin, and K. Wang, "Two-layer federated learning with heterogeneous model aggregation for 6g supported internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 5308–5317, 2021.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

AI-Augmented Model-Based Capabilities in the AIDOaRt Project: Continuous Development of Cyber-physical Systems

Alessandra Bagnato

Softeam, Docaposte Groupe, France, Alessandra.bagnato@softeam.fr

Antonio Cicchetti

Mälardalen University, IDT, Sweden, antonio.cicchetti@mdu.se

Luca Berardinelli

Johannes Kepler University, Linz, Austria, luca.berardinelli@jku.at

Hugo Bruneliere

IMT Atlantique, LS2N (UMR CNRS 6004), hugo.bruneliere@imt-atlantique.fr

Romina Eramo

University of L'Aquila, Italy, romina.eramo@univaq.it

Abstract

The paper presents the AIDOaRt project, a 3 years long H2020-ECSEL European project involving 32 organizations, grouped in clusters from 7 different countries, focusing on AI-augmented automation supporting modeling, coding, testing, monitoring, and continuous development in Cyber-Physical Systems (CPS). To this end, the project proposes to combine Model Driven Engineering principles and techniques with AI-enhanced methods and tools for engineering more trustable and reliable CPSs. This paper introduces the AIDOaRt project, its overall objectives, and used requirement engineering methodology. Based on that, it also focuses on describing the current plan regarding a set of tools intended to cover the model-based capabilities requirements from the project.

Keywords: Model-Based Engineering, Cyber-Physical Systems, Development Operations, Artificial Intelligence.

1 Introduction

The AIDOaRt [1] project aims at supporting systems engineering and continuous delivery activities, namely requirements engineering, modeling, coding, testing, deployment, and monitoring, with AI-augmented, automated Model-Based Engineering (MBE) [2] and Development Operations (DevOps). To achieve this goal, AIDOaRt proposes a model-based framework architecture (cf. Figure 1) that specifies proper methods and tools to enable design and run-time data collection, ingestion, and analysis to provide tailored and efficient AI/ML solutions that will be integrated and evaluated on concrete industrial case studies involving various Cyber-Physical Systems (CPS). The objective of the project, as shown in Figure 1, is to provide a model-based framework to support the CPS development process by introducing AI-augmented automation. Enhancing the DevOps toolchain by employing AI and Machine Learning (ML) techniques in multiple aspects of the system development process (such as modeling, coding, testing, and monitoring), supporting the monitoring of runtime data (such as logs, events, and metrics), software data, and traceability (Observe), analyzing both historical and real-time data (Analyze) and the automation of functionality (Automate).

2 The AIDOaRt Project

The project started on April 1, 2021, and involves 32 industrial and academic partners from different European countries [1]. The AIDOaRt project consortium consists of three different organization categories: Research Partners, Industrial Use Case Partners, and Technology Provider Partners. The industrial use case partners provide industrial case studies. Each case study comes with a set of requirements and a set of existing technologies and tools offered to the other two types of partners as baseline technologies. Both the technology providers and research partners develop new technologies and tools to satisfy the requirements specified by the industrial use case partners. They use the baseline technologies offered by the industrial use case partners and perform various research and

[1] https://www.aidoart.eu

[2] In this paper, we interchangeably use both Model-Based Engineering (MBE) and Model-Driven Engineering (MDE), although we are aware that MDE recalls a higher automation degree (cf. https://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/)

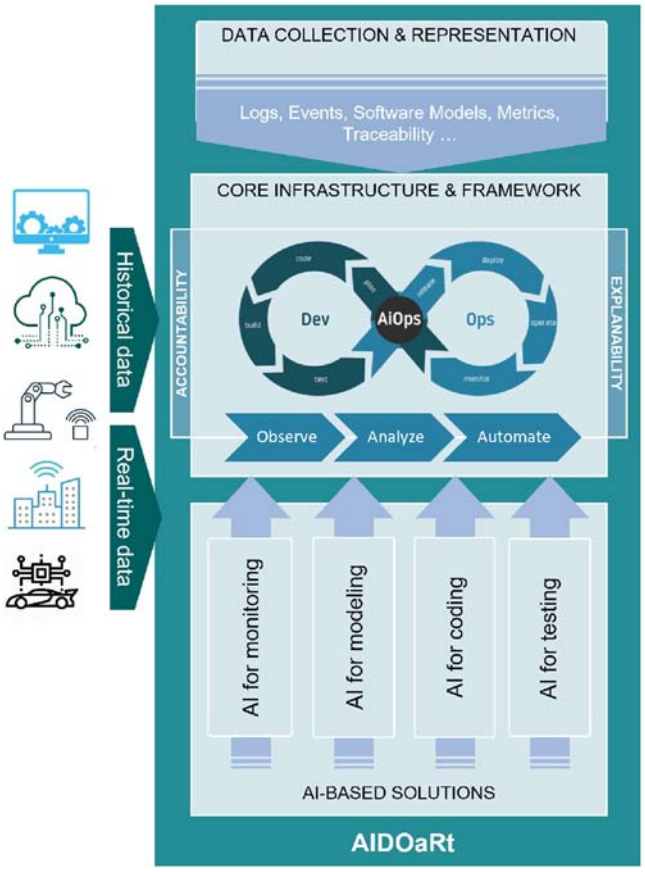


Figure 1 Overview of the AIDOaRt approach

development tasks to develop new capabilities and technologies, as potential exploitation assets. Finally, the new capabilities/technologies developed by the technology providers and research partners are evaluated and adopted by the industrial use case providers. In this way, the industrial use case providers benefit from the newly developed AIDOaRt technologies and tools. Moreover, the technology providers and research partners benefit by working on important, industry-relevant research and development problems and use case requirements. Industrial Use case partners provide 15 industrial case studies capturing CPS of varying complexity levels and in different application domains, ranging from automotive to software and communication systems. AIDOaRt Research and industrial partners propose 50+ candidate solutions potentially relevant to directly tackle the case study requirements, or generally fit within the framework of AIDOaRt, to fulfil typical data-centric, AI-augmented, or model-based engineering requirements. These solutions also vary in their offered services, the consumed or produced input/output data, and the technical constraints (data requirements, supported platforms and deployment procedures).

3 Project Requirements

The nature of our AIDOaRt project, in terms of number of partners and variety in requirements and proposed solutions, requires the adoption of a complete methodology to gradually define the architecture and to incrementally collect, refine and clarify the requirements, and map them to candidate solutions [2]. Industrial use case partners have identified in these case studies a quite varied list of 128 functional and data requirements that are satisfied by the project available tools. The requirements largely differ in terms of abstraction levels, broadness, and coverage and they cover the five key SE/DevOps activities (Requirements Engineering, Modeling, Coding, Testing and Monitoring) as well as different desired capabilities. As a concrete example from the project, Figure 2 shows the AIDOaRt Core Tool Set component and use case requirements related to the Model-based capabilities [3]. This component supports the loading, navigation, querying, transformation, tracing/federation and then the saving of the required models and metamodels for example, Volvo requires the "development of standard data classification, reusable definition, representation, usage (VCE_R07)" as well as to "customize standards-based modeling frameworks (e.g. UAF, SysML, UML) and metamodels to develop system, software, data architecture models (VCE_R05)". The adopted agile methodology [4] allows us to integrate newly reported updates on requirements and solutions descriptions from all partners in a flexible, simple, and traceable fashion.

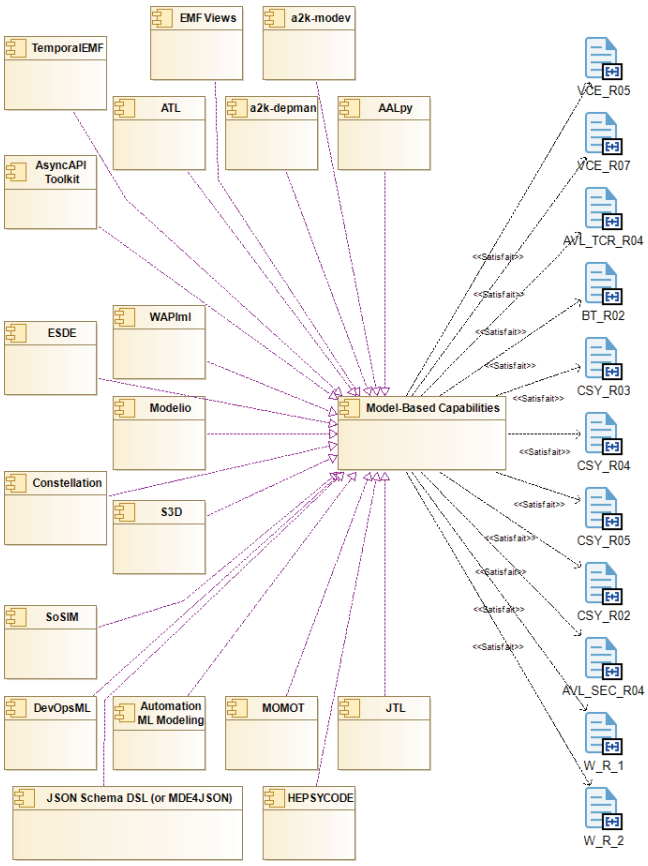


Figure 2 Model-Based Capabilities Traceability Diagram

4 Some Project Tools

This section provides the description of tools developed by IMT Atlantique (IMTA), SOFTEAM, Johannes Kepler University Linz (JKU) , University of L'Aquila (UNIVAQ), Mälardalen University (MDU), and that will be extended and used in the context of the overall AIDOaRt approach to respond to the model-based capabilities requirements.

4.1 EMF Views and ATL

EMF Views[3] is an Eclipse-based solution that brings the concept of database views to the modeling world. It allows creating model views that focus on only one part of a model, or views that combine several models together (and that potentially conform to different metamodels). Model views can be navigated and queried as regular models, and they can be used as inputs to model transformations (notably transformations specified in ATL).

ATL (ATL Transformation Language)[4] is an Eclipse-based model-to-model transformation language and toolkit. In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models. Complementary to EMF Views and ATL, *NeoEMF*[5] is a model persistence solution designed to store models in several kinds of NoSQL datastores. It is fully compatible with Eclipse/EMF, thus making it easy to integrate into (EMF-based) modeling applications.

In the context of AIDOaRt, the plan is to reuse and extend EMF Views to support the development of various news features of the tool as well as their application within different AIDOaRt use cases or scenarios. Thus, we already plan new contributions to EMF Views itself (e.g., in terms of extensions to the tool, realized as Eclipse plugins). Concerning ATL, the plan is to reuse it as is to support the development of various features for other tools / components / use cases. Thus, we do not plan any new contribution to ATL itself. No new contributions are also currently planned concerning NeoEMF. In general, all the refinements and extensions developed in the context of AIDOaRt will be made completely available as open-source Eclipse plugins under the terms of the Eclipse Public License v2.0 (EPL 2.0). They may also be made available under the following open-source Secondary Licenses when the conditions for such availability set forth in the Eclipse Public License, v. 2.0 are satisfied: GNU General Public License, version 3.

4.2 Modelio and Modelio Constellation

Modelio [6] is an open-source modeling environment supporting industry standards like UML and BPMN. Modelio proposes extension modules and can be used as a platform for building new Model-Driven Engineering (MDE) features such as model analysis through querying, code generation and reverse engineering of Java and C++.

Modelio Constellation, commercialized as "Modelio SaaS"[7], by Softeam, Docaposte, is a cloud-based requirements, systems modeling and project governance tool that offers simplified, concurrent, and collaborative access for large teams to the models in the Modelio environment. Modelio SaaS models the case studies requirements, solutions descriptions and implementation roadmaps, and the architecture specifications. A shared and distributed repository holds and synchronizes the various model fragments and artifacts. This repository represents a single source of truth for all the partners and a "live" model continuously updated by each partner throughout the whole project lifetime. Fragments and particular views of these model artifacts are automatically exported into human-readable documents and serve as the basis of the various project deliverables.

In AIDOaRt, Modelio will be enriched with extensions to facilitate System Architects efforts while defining requirements and system modeling, adding non-functional properties modelling, and advanced analysis techniques based on AI for inference on standards (e.g. UML, SysML).

4.3 AutomationML MOMoT, MDE4JSON and DevOpsML

AutomationML[8] (AML) is a modeling language based on CAEX the Computer Aided Engineering Exchange (CAEX) IEC standard[9]. CAEX is a neutral data format that allows storage of hierarchical object information. AutomationML was originally devised to support the engineering of cyber physical production systems (CPPS) by combining physical topology with 3D geometrical kinematics, and PLC software logic. CAEX and AML standards are developed in the XSD/XML technical space. JKU, as a member of the AML consortium, together with 54 other industrial and academic partners and, during the years, is developing MDE framework around AML and CAEX, to make them fully integrable with other standards (e.g., SysML) and more in general, to bridge it with the Eclipse Modeling Framework (EMF) tand related technologies. In AIDOaRt, we plan to adopt an *AML modeling workbench*[10], to satisfy modeling requirements (e.g., VCE05 and VCE07 stated in the previous section), promoting the adoption of AML/CAEX standards in AIDOaRt model-driven engineering processes.

MOMoT is a search-based model transformation tool that allows the optimization of in-place model transformation orchestration to solve engineering problems. Based on Eclipse EMF, in MOMOT the problem and solution domains are defined by the same Ecore-based metamodel. Consequently, both concrete problem and solution(s) are conforming *problem and solution instance models*. MOMOT relies on Henshin, a graph-based model transformation framework to generate solution instance models that optimize given *objectives* and while satisfying given *constraints*, specified in Java or OCL. Currently, MOMOT adopts MOEA[11] as a base meta-heuristic search framework. Currently, we are working on extending MOMOT with reinforcement learning techniques.

MDE4JSON (a.k.a. *JsonSchemaDSL*) is one of the first results of the AIDOaRt project [4]. The approach allows

[3] https://www.atlanmod.org/emfviews/
[4] https://www.eclipse.org/atl/
[5] https://neoemf.atlanmod.org/
[6] www.modelio.org
[7] https://www.modeliosoft.com/fr/solutions/modelio-saas.html
[8] www.automationml.org
[9] https://en.wikipedia.org/wiki/CAEX
[10] https://github.com/amlModeling/caex-workbench/
[11] http://www.moeaframework.org/

the integration of arbitrary JSON-based artifacts in a fully-fledged MDE process. reusing the native JSON concrete syntax/textual notation and generating Xtext-based editors. In AIDOaRt, it can be used both by UC and solution providers that manipulate JSON documents as shown for Keptn in continuous delivery scenarios [5].

DevOpsML is an EMF-based *conceptual framework* in its inception phase, developed in the context of the Lowcomote project[12] for modeling and integrating models representing *DevOps process(es)* and *platform(s)*. In AIDOaRt, DevOpsML can be used to create a simplified architectural model for AIDOaRt solutions or its overall framework.

4.4 Flexible + Blended modeling

MDU has collaborated with several companies with the common objective of introducing more formal ways of modeling, to move from descriptive to prescriptive uses of models (e.g., to enable continuous MBD). A tool has been developed to enable the bidirectional mapping from descriptive to prescriptive architecture modeling support by considering Draw.IO diagrams and a domain-specific language (DSL) defined through an xText-like grammar. In turn, the DSL enables analysis about the correctness/completeness of the architectural specifications.

The plan for AIDOaRt is to use the realized mapping and the lessons learned from that experience to introduce prescriptive modeling in other use cases. Indeed, we consider the existence of these forms of modeling as a precondition for enabling the AI-augmented interplay of DevOps and MDE.

4.5 JTL and TWIMO

JTL[13] (Janus Transformation Language) is an EMF/Eclipse based model-driven framework specifically tailored to support model synchronization and traceability. JTL allows to specify and execute model transformations in both forward and backward direction. It is specifically tailored to support non-bijective transformation (non-determinism). Also, JTL allows model synchronization and change propagation by means of bidirectional model transformation. Finally, JTL can automatically generate traceability relations between different models (e.g., runtime and design models) by exploiting bidirectional model transformations and automated reasoning techniques (answer set programming). Such traceability models can be used as inputs to any other EMF-based modeling and analysis tool (e.g., e.g., for model transformations, code generation, model analysis, etc.).

In AIDOaRt, the plan is to reuse and extend JTL to support the development of news features of the tool to contribute to the model-based core set capabilities as well as their application within different AIDOaRt use cases or scenarios.

TWIMO (digital TWin for MOdeling and analysis) is a framework that exploits MDE principles to drive AI/ML augmentation. It offers advanced modeling and AI/ML analysis and validation capabilities, in particular: (i) extending standard domain system language to offer advanced modeling and validation capabilities in the automotive domain, (ii) defining a domain-specific language for the specification of human driver behavior in the automotive domain, to offer advanced modeling and ML analysis capabilities. (iii) providing ML-based analysis and prediction capability on the human driver behavior in the automotive domain.

In AIDOaRt, the plan is to adapt and improve its methodologies and technologies with the scope to integrate them in the DevOps practices. UNIVAQ is interested to acquire AI/ML techniques to specifically improve pattern detection and correlation to support the designer more widely in the continuous improvement of software through the exploration of the effects of automated refactoring actions. In this respect, gathering feedback from the application of such technologies to real life complex CPSs may represent a compelling source of improvement. UNIVAQ aims to improve its data correlation techniques by exploiting AI/ML for the automated association of design and runtime information.

Conclusions

The paper describes the AIDOaRt approach to maximize the benefit of the project results by applying the developed technologies and tools on the project industrial case studies and by discovering more opportunities in the industry where the project results can be applied. A selection of the tools that will be extended and used in the context of the overall AIDOaRt approach to respond to the model-base capabilities requirements is also presented including some first analysis of the expected improvements after the first year of the project. As the work in the project progresses, discussions and in-depth negotiations are taking place between case study providers and solution providers. This is leading to precise and concrete use case and data requirements, and better choices and selections of candidate solutions.

Acknowledgements

Special thanks to all AIDOaRt consortium members. The AIDOaRt project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 101007350. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Austria, Czech Republic, Finland, France, Italy, and Spain.

References

[1] H. Bruneliere et al., "AIDOaRt AI-augmented Automation for DevOps, a Model-based Framework for Continuous Development in Cyber-Physical Systems", *Microprocessors and Microsystems: Embedded Hardware Design*, Elsevier, 2022, 94, pp.104672. doi: 10.1016/j.micpro.2022.104672.

[2] A. Sadovykh et al, "Applying Model-based Requirements Engineering in Three Large European Collaborative Projects: An Experience Report", 2*9th IEEE International Requirements Engineering*

[12] www.lowcomote.eu

[13] http://jtl.univaq.it/

Conference, Sep 2021, Notre Dame, South Bend, United States. doi: 10.1109/RE51729.2021.00040.

[3] "AIDOaRt deliverable D3.2 - AIDOaRt Core Infrastructure and Framework - Initial Version". URL: https://www.aidoart.eu/download/18.7d39d59b181260bc66b1233f/1654680145439/D3.2.pdf

[4] B.Said et al, "Towards AIDOaRt Objectives via Joint Model-based Architectural Effort", *Proc. of RCIS-WS&RP 2022 RCIS 2022*, CEUR-WS.org, online ceur-ws.org/Vol-3144/RP-paper14.pdf

[5] A. Colantoni et al., "Leveraging Model-Driven Technologies for JSON Artefacts: The Shipyard Case Study," *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems* (MODELS), 2021, pp. 250-260, doi: 10.1109/MODELS50736.2021.00033.

MORPHEMIC - Optimization of the Deployment and Life-Cycle Management of Data-Intensive Applications in the Cloud Computing Continuum

Alessandra Bagnato

Softeam, Softeam Software, Docaposte Groupe, 3 avenue du Centre 78280 Guyancourt.; Tel: +33 1 30 12 18 58; e-mail: Alessandra.bagnato@softeam.fr

Józefina Krasnodębska

7bulls.com Sp. z o.o., ul. Aleja Szucha 8, 00-582 Warsaw, Poland; e-mail: jkrasnodebska@7bulls.com

Abstract

In their Cloud strategy companies are choosing more and more multi-cloud computing giving them the opportunity to distribute its assets, redundancies, software, applications, and anything it deems worthy not only on one Cloud-hosting environment, but rather across several. The model of using multiple Cloud services to host the business's functions and features has a list of advantages that can provide security, flexibility, cost-effectiveness and more to increase business's efficiency and ensure it stays up and running 24 hours a day. The paper presents the MORPHEMIC H2020 project and its unique way of adapting and optimising Cloud computing applications. The Morphemic project covers several features from modelling cross-cloud applications, continuous and autonomous optimization and deployment and providing access to several cloud capabilities for data intensive applications. The outcome of the project will be implemented in the form of an open-source platform covering all the data intensive applications deployment phases starting from modelling, through profiling, optimization, runtime reconfiguration and monitoring.

Keywords*: Cloud services, Cloud computing, model-driven engineering, multi-cloud platform.*

1 Introduction

While organisations are using multiple clouds, this does not necessarily mean that individual applications can be deployed across different clouds and most commonly applications are siloed on the different clouds. Furthermore, while multi-cloud is becoming the preferred solution few organisations are taking advantage of multi-cloud management tools which become fundamental for cost-effective management of cloud resources, governance and security. New modelling techniques and mechanisms are needed to compose and coordinate resources across heterogeneous Clouds and computational infrastructure allowing for multivariate deployment of applications. The MORPHEMIC project [1] described in this paper, aims at optimising the deployment and life-cycle management of data-intensive applications in the Cloud computing continuum. With the MORPHEMIC approach the initial deployment of Cloud application components is performed and then proactively scaled according to the incoming workload and the defined service level objectives on forecasted metrics about usage and workload level. The project is an extension of MELODIC multi-cloud platform developed in the H2020 project[2] and it introduces two novel concepts to it: "polymorphing architecture" that will allow for dynamic adaptation of the architecture of application to the current workload and "proactive adaptation «that will allow the reconfiguring of the application based forecasted metrics about usage and workload level.

2 The MORPHEMIC Project

Funded by the European Union Horizon 2020 research and innovation programme, the project MORPHEMIC began in January 2020 for a period of 36 months. The project involves 12 partners from 7 countries belonging to both the academic and the industrial world.

On the academic side, it includes FORTH[1], UiO[2,] UPRC[3], ICCS[4].

From the industrial side, it includes Softeam[5], ICON[6], IS-Wireless[7], CHUV[8], InAccel[9], Activeon[10], 7bulls[11], Engineering[12].

[1] https://www.ics.forth.gr/
[2] https://www.uio.no/
[3] https://www.unipi.gr/unipi/en/
[4] https://www.iccs.gr/
[5] https://www.softeamgroup.fr
[6] https://www.iconcfd.com
[7] https://www.is-wireless.com/
[8] https://www.chuv.ch/fr
[9] https://inaccel.com/
[10] https://www.activeeon.com/fr
[11] https://www.7bulls.com/en
[12] https://www.eng.it/en/

The MORPHEMIC open-source platform [11] provides to its users the optimization of the deployment and life-cycle management of data-intensive applications in the Cloud computing continuum. Specifically, MORPHEMIC undertakes the initial deployment of Cloud application components, it proactively scales them or even changes the deployment model morph, according to the incoming workload and the defined service level objectives.

If we consider an application composed by different components (services), MORPHEMIC optimises the deployment of each service by defining which deployment environment (Cloud, Fog) and which application form (e.g., container, virtual machine, serverless) to apply maximising the business benefit provided by the application and optimising the usage of the deployment resources. Furthermore, MORPHEMIC aims at keeping the benefits constantly high by predicting the future context and proactively modifying the deployment model and the application form.

With its innovative features, MORPHEMIC aims at extending the MELODIC multi-cloud platform developed in the H2020 project MELODIC [2]. Through the MORPHEMIC project, the goal is to simplify Cloud application modelling and continuously optimise and morph the deployment model to take advantage of beneficial Cloud capabilities. This ensures that adaptation can be done effectively and seamlessly for the users of the application.

The main pillars being extended in the MORPHEMIC are:

- **Polymorphing architecture**: Ability to run and deploy a component, depending on its requirements and workload, in different technical forms (i.e., in a Virtual Machine (VM), in a container, as a big data job, or as serverless components, etc.) and different environments. This maximises the utility of the application deployment and the satisfaction of the user as the benefits provided by each of the deployment environments for the application are maximised.
- **Proactive adaptation**: better deployment, automation and orchestration of Big Data and Cloud applications in the Cloud. Forecast of future resource needs and deployment configurations to ensure that adaptation can be done effectively and seamlessly for the users of the application. Quick and proactive re-deployment is ensured to maintain a high optimization level.
- **Flexible and resilient monitoring**: through a dedicated event management system which actively follows, aggregates and processes application and infrastructure level metrics across the Cloud computing continuum while it can cope with node failures or network issues by self-healing its monitoring modules.

The outcome of the project will be implemented in the form of the complete solution, starting from modelling, through profiling, optimization, runtime reconfiguration and monitoring.

Figure 1 presents the MORPHEMIC optimization flow using MELODIC [2] as the real-time application manager. The result of executing this flow is a deployed and continuously optimized application configuration. Their detailed description can be found in the deliverable D4.1[6].

The flow contains 4 main parts that can be mapped to the autonomic computing MAPE-K loop:

- Profiling - is the pre-processing step as it handles the preparation of the CAMEL model, the utility function creation based on the information in the model, and the application profiling used for the polymorphic adaptation.
- Reasoning - is the core process of the platform. It includes both Analysis and Planning from the MAPE-K loop finding the best deployment configuration within the given operational constraints and considering the application's execution context for the desired reconfiguration time point. The history of execution contexts represents the knowledge required for the reasoning.
- Executing - deals with the adaptation and orchestration of the deployment model to be deployed across the chosen Cloud providers.
- Monitoring - is responsible for gathering data used in autonomous optimization.

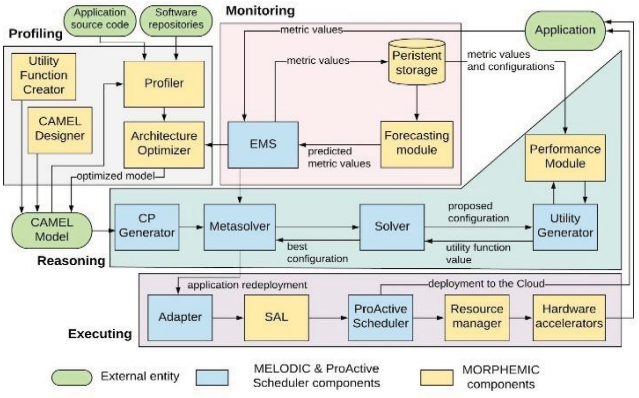


Figure 1: Overview of MORPHEMIC architecture

The following elements are appearing in Figure 1 and are part of the solution:

- Utility Function Creator used for creation of the utility function in a user-friendly way,
- CAMEL Designer used for the creation of the CAMEL model,
- Profiler responsible for constructing and maintaining the profile of an application,
- Architecture Optimizer responsible for optimizing the application architecture,

- Constraint Programming (CP) Generator which converts the CAMEL application model to a CP problem,
- Metasolver which triggers the reconfiguration when any the constraints of the CP problem is violated, and selects the appropriate Solver,
- Solvers which can solve the CP problem,
- Utility Generator that calculates the utility of the proposed CP solution candidate,
- Adapter responsible for orchestrating the deployment of the application,
- ProActive Scheduler that deploys and reconfigures the application to the Cloud providers using Scheduling Abstraction Layer (SAL) as an abstraction layer,
- Resource Manager which is a sub-component of ProActive Scheduler responsible for deployment and configuration of the infrastructure resources exploited by the applications,
- Hardware Accelerator that deploys to hardware accelerated Cloud computing resources.
- The Event Management Services (EMS) is used for distributed event processing and
- collection of metric values,
- Forecasting module forecasts metric values into the EMS infrastructure,
- Persistent storage is responsible for storing real and forecasted metric values, and
- Performance module is responsible for maintaining a performance model for an application and Cloud resources.

3 The MORPHEMIC CAMEL Models

The deployment and polymorphic adaptation of cross-cloud applications in the MORPHEMIC project is based on Cloud Application Modelling and Execution Language (CAMEL).

CAMEL [12] is a multi-domain-specific language (DSL) [13] allowing users to specify multiple aspects/domains related to multi-/cross-cloud applications, such as the domains of deployment, requirement, metric, scalability, security, organisation, and execution.

Furthermore, CAMEL will be extended within the MORPHEMIC project to cover the polymorphic modelling concepts that allow applications to have several possible deployment configurations.

The CAMEL Modelling task is performed by CAMEL Designer [3] tool which is a module for Modelio Open-Source modelling tool [4]. Modelio is an extensible tool and modules are the means of defining, implementing, and deploying extensions for Modelio. They can be seen as the equivalent of extensions for Firefox. They are used to extend and adapt Modelio Modeller by providing additional functionalities and services that meet specific needs. The defined CAMEL model can be exported from the Modelio modeling tool and then uploaded and used by the MORPHEMIC platform.

Figure 2 shows the Modelio CAMEL Designer Interface and an example of a CAMEL Security Model for Application and Interface Security [10], including the AIS_04 Security Control where “Policies and procedures shall be established and maintained in support of data security to include (confidentiality, integrity and availability) across multiple system interfaces, jurisdictions and business functions to prevent improper disclosure, alteration, or destruction.". MORPHEMIC will improve the security and reliability of cloud applications by using the security features of candidate infrastructures as one of the decision parameters to select the most appropriate deployment environments. More details on MORPHEMIC Security and design implementation can be found at [17].

The management of the execution task of applications deployment and the polymorphic adaptation is implemented by the Web User Interface Client [7] which includes managing heterogeneous resources such as cloud offers and material accelerators, managing CAMEL Models, optimising, deploying and monitoring the Cross-Cloud Application [5].

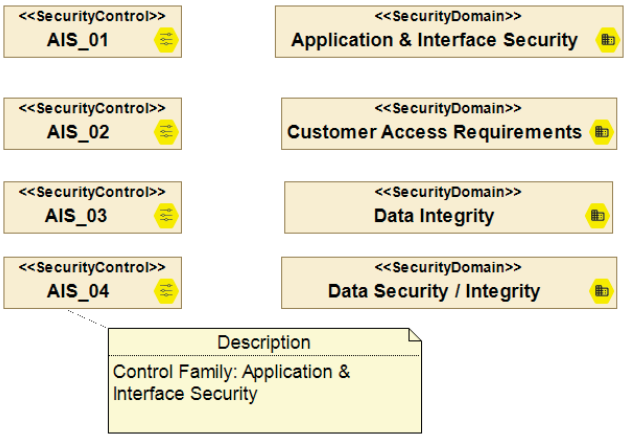


Figure 2: CAMEL Security Model

4 The MORPHEMIC Case Studies

MORPHEMIC project relies on three Use Case studies to validate its solution of continuous optimising and polymorphing cloud-based applications.

The Use Cases cover the following areas of industry: medical research, telecommunication, and fluid mechanics' research. They highlight how MORPHEMIC considers new needs that technology and real-world applications nowadays should meet. Specifically, they need on one hand more and more resources, and, on the other hand, to support different models, such as Edge and Fog, besides the Cloud.[8]

These use case studies are provided by different stakeholders:

- ICON (ICON Technology & Process Consulting Limited) operates in the high-tech field of

Computational Fluid Dynamics (CFD) and provides blue-chip multi-sector engineering companies, their suppliers, and consultants with the ability to predict fluid flow using 3D computer simulation; [14]

- Centre Hospitalier Universitaire Vaudois (CHUV): Lausanne university hospital is one of the five Swiss university hospitals. Through its collaboration with the Faculty of Biology and Medicine of the University of Lausanne, CHUV plays a key role in the areas of medical care, bio-Prototype medical research and education; [15]
- IS-Wireless is an SME based in Poland with expertise in developing algorithms, protocols and tools for 4G and 5G mobile networks. [16]

Design of the use cases scenarios, including their description, the scenario requirements and targets, the infrastructures and platforms that will be utilised and the description on how MORPHEMIC will handle each individual use case within the project is described at [9].

5 MORPHEMIC Features for data-intensive applications

The project addresses the described case studies but also any type of data intensive application offering a complete set of Cloud computing services and an optimal usage of ICT resources thanks to the automatic management and deployment of applications and their proactive adaptation in time.

Data Intensive Cloud applications performance and usability are maximised by the deployment in the best environment, and application up time is improved due to predictive analysis of the application's future state.

Data intensive application are offered a selection of cloud providers with potential cost saving and vendor lock avoidance due to predictive analysis of the application's future state.

Seamless Cloud Acceleration is also provided-MORPHEMIC allows the easy deployment and utilisation of hardware accelerators (i.e., FPGAs) in the Cloud. That way, users can speed up their applications using the available accelerators and reduce the total cost of operations (COTS) by using more cost-efficient platforms.

Moreover, thanks to usage of the security features of candidate infrastructures as one of the decision parameters, cloud data intensive applications using MORPHEMIC can select the most appropriate deployment environments improving reliability and security.

Conclusions

Nowadays, Cloud application modelling languages do not supply the polymorphic application components and do not provide polymorphic infrastructure models. Therefore, MORPHEMIC goes beyond the state-of-the-art to introduce automatic DevOps capabilities for the efficient application life-cycle management in the dynamic Cloud computing continuum. Such capabilities are going to increase the satisfaction of Cloud end-users as they avert from vendor lock-in, reduce Cloud costs and give the opportunity to exploit the full potential of available Edge devices and constantly maintain the required application (QoS). Polymorphic and Proactive Modelling, Planning and Adaptation, basis of the MORPHEMIC platform, respond to these requests. The project aims to achieve great added value and innovation in polymorphic adaptation which is the ability of MORPHEMIC Platform to optimise and adapt the deployment configuration of cross-cloud applications by its components underneath. Moreover, it aims to achieve a great user experience with this product to allow users to fully benefit from this innovative concept, user evaluation is expected in the next twelve months and will be detailed in the Final Evaluation Deliverable due at the end of the project.

References

[1] MORPHEMIC website: Available at: www.morphemic.cloud (Accessed: 22-04-2022)

[2] G. Horn and P. Skrzypek, "MELODIC: Utility Based Cross Cloud Deployment Optimisation", 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), Doi: 10.1109/WAINA.2018.00112.

[3] Modelio Camel Designer, Available at: github.com/Modelio-R-D/CamelDesigner (Accessed: 22-04-2022)

[4] Modelio website: Available at: www.modelio.org (Accessed: 22-04-2022)

[5] A.Moussaoui, A.Bagnato, E.Brosse J. Krasnodębska and P. Skrzypek, "The MORPHEMIC Project and its Unified User Interface", *Proc. of RCIS-WS&RP 2022 RCIS 2022 Workshops and Research Projects Track*, Spain, May 2022, CEUR-WS.org, online: ceur-ws.org/Vol-3144/RP-paper13.pdf

[6] D4.1 Architecture of pre-processor and proactive reconfiguration, Available at: https://www.morphemic.cloud/deliverables/ (Accessed: 22-04-2022)

[7] D5.1 User Interfaces Specification, Available at: https://www.morphemic.cloud/deliverables/ (Accessed: 22-04-2022)

[8] D6.1 Industrial requirements analysis, https://www.morphemic.cloud/deliverables/ (Accessed: 22-04-2022)

[9] D6.3 Use cases definition and preparation, Available at: https://www.morphemic.cloud/deliverables/ (Accessed: 22-04-2022)

[10] CAMEL Security Model Example, Available at: https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel/examples/security.camel?at=refs%2Fheads%2Fcamel_2.5 (Accessed: 22-04-2022)

[11] Morphemic Platform on OW2 GitLab, Available at: https://gitlab.ow2.org/melodic/morphemic-preprocessor (Accessed: 22-04-2022)

[12] CAMEL DSL Documentation, Available at: http://camel-dsl.org/documentation/ (Accessed: 22-04-2022)

[13] Modelio Open Source, Available at: https://github.com/ModelioOpenSource/Modelio (Accessed: 22-04-2022)

[14] Computational Fluid Dynamics simulation Background and challenges, Available at: https://www.morphemic.cloud/computational-fluid-dynamics-simulation/(Accessed: 22-04-2022)

[15] e-BrainScience Background and challenges, Available at: https://www.morphemic.cloud/e-brainscience/ (Accessed: 22-04-2022)

[16] 5G Cloud-RAN Background and challenges, Available at: https://www.morphemic.cloud/virtualized-base-station/ (Accessed: 22-04-2022)

[17] D4.2 Security design and implementation https://www.morphemic.cloud/deliverables/ (Accessed: 22-04-2022)

5G Communication and Security in Connected Vehicles

Antonio Imbruglia

STMicroelectronics, Stradale Primosole 50, 95121, Catania, Italy; email:antonio.imbruglia@st.com

Daniela Cancila

Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France; email: daniela.cancila@cea.fr

Marina Settembre

Fondazione Ugo Bordoni, Viale del Policlinico 147, 00161, Roma, Italy; email:msettembre@fub.it

Abstract

The widespread diffusion of Cyber-Physical Systems and their capability to interact with the physical world depend also on the availability of 5G network. The exponential development of intelligent and interconnected IoT and autonomous systems, combined with the development of 5G networks, presents new challenges from a cyber-security perspective. The paper, without claiming to be exhaustive, offers insights and reflections on the very broad topic that integrates innovative devices, 5G and cybersecurity by illustrating the main European directions indicated by both the Strategic Research and Innovation Agenda for Electronic Component and Systems (ECS-SRIA) and the evolution of 5G network standards. Some of the afore-mentioned issues will be reanalysed through use cases based on Vehicles to X (V2X) scenario, where connectivity, safety and cybersecurity play a key interworking.

Keywords: ECS-SRIA, 5G, Cybersecurity, Safety, V2X

1 Introduction

The exponential development of IoT systems, electric embedded devices and cyber-physical systems (CPS), with stronger intelligence and interconnection requirements, takes advantage of the development of 5G networks. In spite of new emerging application scenarios, new thorny issues in cyber-security should be considered due to an increased attack surface and evolving threat landscape. These systems are going to operate in a more dynamic and open environment respect to traditional systems, with increasing needs to exchange information with other systems. Consider electric and autonomous vehicles, where the connection with both other vehicles and the environment (e.g. infrastructure, roads, traffic lights pedestrians) can improve road safety, reduce environmental impact, and provide a new driving experience. However, if susceptible to cyber-attacks, these systems could have catastrophic consequence. Moreover, in traditional systems, specialized operators manage the system. For example, in the railway application domain, a team of experts manages the system (including train maintenance). In this new scenario, the actors involved in the management can be more heterogenous, belonging to different organizations and not always so specialized to recognize the nominal or degraded behavior of the system, following for example a cyber-attack.

The paper, without claiming to be exhaustive, offers insights and reflections on the very broad topic that integrates innovative devices, 5G and cybersecurity by illustrating the main European directions indicated by both the Strategic Research and Innovation Agenda for Electronic Component and Systems (ECS-SRIA), [1] and some insights into the evolution of 5G network standards. For the sake of concreteness, some considerations will be reanalyzed in Vehicles to X (V2X) case studies.

2 ECS-SRIA

The Strategic Research and Innovation Agenda for Electronic Component and Systems (ECS-SRIA) aims at describing the "Major Challenges, and the necessary Research & Development & Innovation efforts to tackle them, in micro- and nanoelectronics for smart systems integration all the way up to embedded and cyber physical systems, and System of Systems" [1].

ECS-SRIA is developed by three industrial associations: AENEAS [2], Inside Industry Association [3] and EPOSS [4], with the support of a European team of experts in the discipline coming from European industries, Research and Technology Organizations and Academics in Europe.

ECS-SRIA is based on three main areas (see Figure 1):

- Key application areas, such as Mobility, (Green colour)
- Fundamental Technologies, such as Embedded software and Beyond, (Blue colour)
- Cross-Sectional technologies, such as cybersecurity, safety, connectivity, (purple colour).

Some of the main ECS-SRIA objectives address:

- To boost industrial competitiveness through interdisciplinary technology innovations [1]

- To ensure EU digital autonomy through secure, safe and reliable ECS supporting key European application domains [1]
- To establish and strengthen sustainable and resilient ECS value chains supporting the Green Deal [1].

The following sections discuss some insights related to the chapters of the ECS SRIA in which the authors are more active.

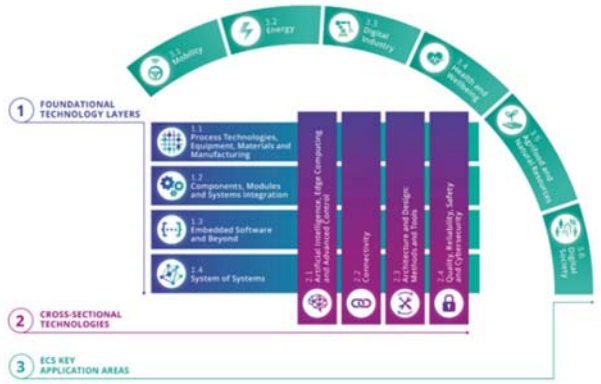


Figure 1. ECS-SRIA 2022 structure [1]

3.1 The Cyber-Security and Privacy Challenge

The new generation of CPS are closed to the non-expert human via societal applications. For example, CPS may be customized to individual, including an embedded combination of heterogeneous subsystems of different quality, artificial Intelligence (AI) and connectivity functionality. In this context, ECS-SRIA identifies the cyber-attacks risks and the potential leakage of sensitive data, highlighting a research direction towards a "robust root of trust system, with unique identification enabling security without interruption from the hardware level right up to the applications, including AI involved in the accomplishment of the system's mission in dynamic unknown environments" [1]. A particular emphasis is placed on trustworthiness and the hardware security of such systems. Many organizations in Europe, including the IC and Digital System Division, at CEA LIST, and STMicroelectronics, are working in this direction.

The challenge also highlights the need to achieve a common recognized certification scheme under the composition of modular trusted hardware and trusted software. Electric and autonomous vehicles, for example, include several heterogeneous sensors, actuators and embedded devices, that can change or updated over time. Evolution of certification dealing with system changing in some parts over time is a challenge both in terms of cost and security, and, hence, competitiveness.

Ensuring privacy is another relevant theme in the challenge. This focus includes not only privacy-by-design approach, but also of quantum-safe cryptography modules everywhere in the system.

Finally, the challenge addresses "ensuring both security and safety" properties. In the case of autonomous vehicle, enhanced connectivity can be useful for driving safety, but, if susceptible to cyber security attacks, may have catastrophic consequences. Ensuring both safety and security requirements has been also capital during the Covid-19 pandemic. Then, in order to achieve a greater level of trustworthiness, it is necessary to properly manage safety and cyber-security issues in the same system.

3.2 The Connectivity Challenge

The main ECS SRIA challenges to ensure European leadership in terms of connectivity technologies as well as associated hardware technology supporting the development of connectivity solutions can be summarized as follows:

- Strengthening the EU connectivity technology portfolio to maintain leadership, secure sovereignty and offer an independent supply chain.
- Investigate innovative connectivity technology (new spectrum or medium) and new approaches to improving existing connectivity technology to maintain the EU's long-term leadership
- Autonomous interoperability translation for communication protocol, data encoding, compression, security and information semantics.
- Architectures and reference implementations of interoperable, secure, scalable, smart and evolvable IoT and SoS connectivity
- Network virtualisation enabling run-time engineering, deployment and management of edge and cloud network architectures

Finally, secure communication and control by powerful computation system applying AI are fundamental.

4. Innovation Aspects in the 3GPP 5G Standard Evolution

The innovative aspects of 5G networks are not only evolutionary in nature, basically characterized by performance improvements (e.g. capacity, mobility management, connection density, spectral efficiency, latency, energy efficiency), but also present many revolutionary elements (e.g., network softwerization and programmability, network slicing, cloud native approach, more flexible and expanded use of frequencies, artificial intelligence, new approach to cybersecurity). 5G can be considered as a multiplicity of dedicated, flexible, and intelligent networks open to new actors to connect anything and to dynamically serve different verticals, exploiting the concept of slicing. In such a complex scenario the standardization, regulatory and institutional bodies, vendors, and operators should face many challenges to ensure security, safety, privacy, net neutrality and transparency for all users [6]. The third-generation partnership project, 3GPP [6], which is the reference standard for mobile communications, identifies three phases in 5G evolution [7]. During phase 1, the first set of 5G specifications on the new radio have been completed both for non-standalone solutions, when the new radio interacts with an existing 4G/LTE core network, and for the standalone solution, when

the new radio interconnects within a 5G core network. The new radio truly represents a step forward from previous generations of radios both in terms of flexibility, throughput, latency, and reliability to meet even mission critical service requirements, exploiting a frequency spectrum that expands from low frequencies below 1 GHz to mmWave, flexible subcarrier spacing up-to 400MHz Channel bandwidths for a single-component (CC) carrier, 3D beamforming for improvement of spectral efficiency, dynamic time-division duplexing (TDD). New radio plays an important role in the autonomous driving application scenario that we will consider in the Section 4. In Phase 2, 3GPP introduced the new concept of Service Base Architecture (SBA) for the 5G core network (Release 16) [8], aiming at providing unprecedented flexibility agility respect to the traditional architecture. In Phase 3, Release 17 and beyond, new 5G enhancements are considered both on the radio part (e.g., MIMO, positioning, side-link) and on the core network and slicing, extending it to the access part [9]. New security features have been introduced compared to previous generations, such as a stronger cryptographic algorithm for 256-bit encryption, better air interface security, user privacy protection and enhanced roaming security, but SBA is a completely new concept and leads to completely new security challenges. The SBA is composed of services. Each Network Function (NF) can be regarded as a service [6]. In the upper green part of the Figure 2 are represented the NFs of the control plane, while the lower part represents the user plane and access networks. It is out of the scope of the present paper entering into details of each network function, (a detailed description can be found in [9]), but for improving the understanding the NFs are, schematically and not rigorously, grouped by functionality.

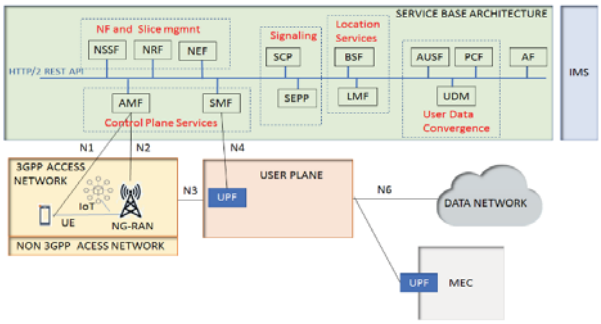


Figure 2. SBA architecture view, [6]

NFs are self-contained, independent and can be connected to a service bus, using common Internet protocol, [10-12]. Each NF is a software running on virtual machines or containers efficiently deployed as a Virtual Network Function (VNF) by exploiting cloud computing. NFV and SDN allow the implementation and automation of customized services on a fully programmable platform. Several NFs can be combined to form a logical block, named slice, addressing a specific purpose, with specific QoS and latency features. 3GPP has already standardized some specific slices as massive Machine Type (mMTC), enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency (URLLC) and Vehicle-to-X (V2X) communications, but many others can be defined as well. It is undoubtable that the integration of network slicing, NFs, NFV, SDN allows a powerful, flexible, fast, and dynamic service deployment, but its management is complex, and the security should be properly addressed,[6]. 5G threat landscape is complex and continuously evolving, [12]. Potential source of attacks (threat actors) could come from end devices, untrusted networks, roaming networks, internet application service providers or insider attack. These attacks constitute different threats to network assets, that can be schematically categorized as loss of availability, confidentiality, integrity, and control [13].

4.1 V2X scenario: communication, safety, and security issues

The automotive industry is at the center of a real revolution. 5G with its low latency, higher bandwidth and great flexibility can provide many previously unachievable features to automotive sector. In recent years vehicles have been equipped with an increasing number of electronic Advanced Driver Assistance Systems (ADAS) developed to increase the level of safety and driving comfort, exploiting four types of sensors: radar, lidar, camera and ultrasonic shortrange sensors, [14]. However, these sensors have limitations. They are unable to see a pedestrian around the corner or to alert drivers to hazards or slowdowns before they meet on the road: they do not communicate. To complement ADAS systems there are systems called Vehicle to X (V2X) overcoming intrinsic limitation of ADAS systems and, hence, providing features of 360° vision, Non-Light of Sight view, extended range, and the ability to communicate between vehicles, [15]. There are basically two types of communication for connected vehicles. Dedicated short-range communication (DRSC) is the first standard of V2X technology, based on the IEEE 802.11p standard and operating at 5.9 GHz. It provides the ability for vehicles to communicate with other vehicles and infrastructure around them, exchanging basic safety messages to prevent collisions, but it is not a system for long range communications (about 300 m). Subsequently, 3GPP has introduced the cellular V2X (C-V2X) starting with Release 14, which in turn was based on Release 12 related to Device to Device (D2D) communications. 3GPP specifies four types of C-V2X communication: Vehicle to Vehicle (V2V), Vehicle to Infrastructure (V2I), Vehicle to Network (V2N), and Vehicle to Pedestrians (V2P). The 3GPP's C-V2X standard supports two complementary communication methods: a cellular network-based communication, that uses the air LTE Uu interface and uses the cellular communication bands for long-range transmissions to connect to the network and thus to services; and a direct or side-link that uses the PC5 interface for short-range (less than 1 km) V2V and V2I communications. 3GPP Release 14 defines the foundations of C-V2X communication for basic security message exchange. New features are added in Release 15, but a real step forward is possible with Release16 with the use of 5G New Radio. 5G NR V2X provides lower latency, ultra-reliable communication, and high data rate useful for addressing challenging autonomous driving requirements (e.g. high throughput sensor sharing,

intent/trajectory sharing, real time local updates to build detailed maps and share them, coordinated driving etc.), [15]. In the connected vehicles scenario some critical challenges and possible C-V2X solutions are:

- high relative speed of two vehicles driving in opposite directions causes a Doppler shift and frequency offset. C-V2X provides an improved signal design an additional reference signal symbols for better channel estimation.
- high vehicle density can lead to radio resource congestion. C-V2X can adopt algorithms that detect available resources, selecting the least congested ones.
- Loss of out-of-coverage synchronization. C-V2X is inherently a synchronous system using GNSS.

An application scenario for connected vehicles with challenging requirements on the radio component is platooning, where the lead vehicle of a convoy relays information to the vehicles behind it. Platooning offers many advantages for long-distance travel since vehicles can drive at a constant speed with a short distance in between, reduced air resistance and, hence, a reduction in fuel consumption and CO2 emissions. Moreover, platooning improves safety by, reducing response times and minimizing the risk or, at least, the impact of accidents. ETSI (European TLC Standards Institute) define, the requirements in terms of end-to-end latency, reliability, and data rate, for enhanced V2X scenarios, (e.g. platooning, as driving with high or full automation, remote driving, sharing large amounts of sensor data). It results that 5G NR V2X represents the most suitable solution for Advanced Safety Automated Driving scenarios, as reported in [15], but the increased connectivity and automation expose them to several cyber threats. [17]. ITU-T Recommendation X.1372 provides security guidelines for different V2X scenarios, some depicted in Figure 3, [18]. Specifically, ITU-T Recommendation X.1372 identifies cybersecurity threats, security requirements to mitigate these threats and describes possible implementations of secure V2X communications. Security issues deal with: i) identification, authenticity to authorize access to services and information, ii) message integrity to ensure that information is accurate and reliable, iii) availability of services and information, iv) confidentiality & privacy of users, their data and their actions from eavesdropping and exploitation and v) non-repudiation and accountability of the source of information. In the case of identification, authenticity, and integrity the main threats to be considered are message manipulation by an attacker providing false information, credential manipulation allowing access to information without authorization, manipulation of sensor data and information, causing traffic congestion, accidents etc., and interception of message and malicious reply in place of the authorized user. In the case of availability, the main threats consist of i) jamming and distributed denial of service (DDoS) attacks caused malicious service requests congesting the channel capacity and so impacting the reliability or availability of C-V2X services, ii) timing attack aiming at delaying the delivery of a safety message to other vehicles and iii) hacking of sensors providing incorrect values determining transient or permanent faults. For confidentiality and privacy issues, the main threats are due to eavesdropping of V2V, V2I, V2P, and V2D messages or leaking of personally identifiable information as identity, position, actions, trajectories of a user of the V2X service. Finally, for non-repudiation case main threats may occur if an attacker manipulates the certification database or accesses the private key to a certificate without authorization. The analysis of possible mitigations to the above-mentioned threats for C- V2X is out of the scope of the present paper.

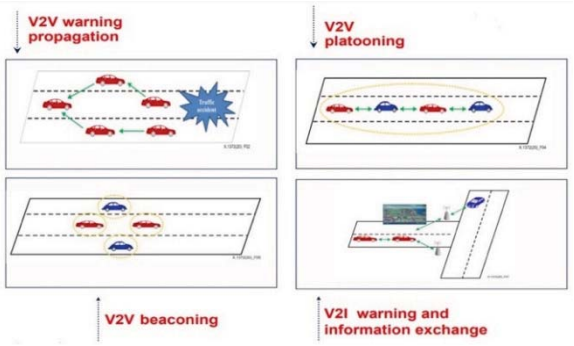


Figure 3. V2X communication scenarios, [18]

Conclusions

In this paper, without claiming to be exhaustive, some insights from the ECS-SRIA and the evolution on 5G networks have been discussed focusing mainly on connectivity, cybersecurity, and safety issues. The cellular-V2X scenario has been analysed as a relevant use case where connectivity, safety and security have a key interwork.

Acknowledgment

This paper has been originated from a lecture the authors gave in the framework of the course 0921SIC10, organized by AMES Society - AEIT (Italian Association of Electrical, Electronics, Automation, Information and Communication Technology). The authors wish to thank AMES -AEIT for supporting this activity.

References

[1] ECS-SRIA. Electronic Components and Systems - Strategic Research and Innovation Agenda, 2022.

[2] https://aeneas-office.org/

[3] Home | Inside (inside-association.eu)

[4] EPoSS (smart-systems-integration.org)

[5] M. Settembre, "A 5G Core Network Challenge: Combining Flexibility and Security", 2021 AEIT International Annual Conference 2021, pp. 1-6.

[6] https://www.3gpp.org/

[7] https://www.3gpp.org/release-16

[8] https://www.3gpp.org/release-17

[9] 3GPP TS 23.501, "System architecture for the 5G System (5GS)", version 17.4.0, March 23, 2022.

[10] NGMN Alliance, "Service Based Architecture in 5G", (Final deliverable (approved-P Public), January 2018.

[11] D. Borsatti, L. Spinacci, C. Grasselli, M. Settembre, W. Cerroni, F. Callegati, “A Network Slicing Architecture for Mission Critical Communications”, IEEE WiMob 2020 Workshop on ICT Systems for PPRR, Oct. 2020.

[12] ENISA Report “Threat Landscape for 5G Networks”, update December 2020.

[13] ENISA Report - Security in 5G Specifications, Feb. 2021.

[14] M. Duncan, A. Imbruglia, S. Gligerini, "The way forward and opportunities towards autonomous driving, AEIT, March, pp. 50-55, 2019.

[15] NGMN Alliance, "V2X: white paper ", 2019.

[16] 3GPP TS 22.186 version 15.3.0 Release 15.

[17] ENISA Report "ENISA good practices for security of smart cars", November 2019.

[18] Recomm. ITU-T X.1372 "Security guidelines for vehicle-to-everything (V2X) communication", March 2020.

Managing Non-functional Requirements in an ELASTIC Edge-Cloud Continuum

Rita Sousa, Luis Miguel Pinho, António Barros
ISEP, Polytechnic Institute of Porto, Portugal

Marco Gonzalez-Hierro, Cristina Zubia
Ikerlan Technology Research Centre, Basque Research and Technology Alliance (BRTA), Spain

Eudald Sabate, Elli Kartsakli
Barcelona Supercomputing Center (BSC), Spain

Abstract

The ELASTIC European project addresses the emergence of extreme-scale analytics, providing a software architecture with a new elasticity concept, intended to support smart cyber-physical systems with performance requirements from extreme-scale analytics workloads. One of the main challenges being tackled by ELASTIC is the necessity to simultaneously fulfil the non-functional properties inherited from smart systems, such as real-time, energy efficiency, communication quality or security. This paper presents how the ELASTIC architecture monitors and manages such non-functional requirements, working in close collaboration with the component responsible for the orchestration of elasticity.

Keywords: Non-functional requirements, elasticity, compute continuum.

1 Introduction

Smart Cyber-Physical Systems (CPS) are require processing a vast amount (in volume and variety) of data from distributed, heterogenous, sources. Generally, data streams from sensing the physical system enable decision-making tasks, both control loops at the edge, and optimization applications in the cloud. Examples of these smart systems can be found in several domains, such as railway, automotive, smart factories, etc. Notwithstanding the importance of the functional requirements of such systems, non-functional properties, such as real-time, energy-efficiency, communication quality and security, need also to be considered and appropriately managed, for the correct functioning of applications [1].

The ELASTIC project[1] addressed the challenge of extreme-scale analytics, with a new software architecture that incorporates the concept of elasticity (matching the amount of resources allocated to a service with the amount of resources it actually requires, avoiding over- or under-provisioning), enabling smart systems to satisfy the performance requirements of extreme-scale analytics workloads. The vision of ELASTIC is that by extending the elasticity concept across the compute continuum in a fog computing environment and combining it with the usage of advanced hardware architectures at the edge side, the capabilities of the extreme-scale analytics can significantly get increased integrating both responsive data-in-motion and latent data-at-rest analytics into a single solution.

In order to fulfil the non-functional properties inherited from smart systems, it is necessary that the ELASTIC architecture includes mechanisms which allow monitoring and adaptation of allocated computing resources to applications, triggering configuration changes upon detection of non-functional requirements violations.

This paper provides a brief overview of the features implemented in the ELASTIC software architecture to perform such management. This paper is structured as follows: the next section provides a summary of the overall software architecture. The specific components to deal with non-functional requirements are presented in Section 3, and briefly evaluated in section 4.

2 Overview of the ELASTIC Software Architecture

One of the main objectives of the ELASTIC software architecture [2] is to manage the interplay of applications and data flows, to provide the required elasticity. Figure 1 presents a simplified view of this software architecture.

The Distributed Data Analytics Platform provides support to different analytic applications (online and offline) in edge and cloud environments. Hence, the applications based on real-time values are typically executed in the edge, while offline analysis , which are based on edge's historical values, run in the cloud. COMPSs [3], an open-source framework developed by Barcelona Supercomputing Center (BSC) is responsible for the deployment and scheduling of workflows in the edge-cloud compute continuum, guaranteeing the required performance. DataClay [4] is an open-source distributed data store also developed by BSC, that provides, among various functionalities, Edge-to-Cloud data storage.

[1] "A Software Architecture for Extreme-Scale Big Data Analytics in Fog Computing Ecosystems", http://www.elastic-project.eu

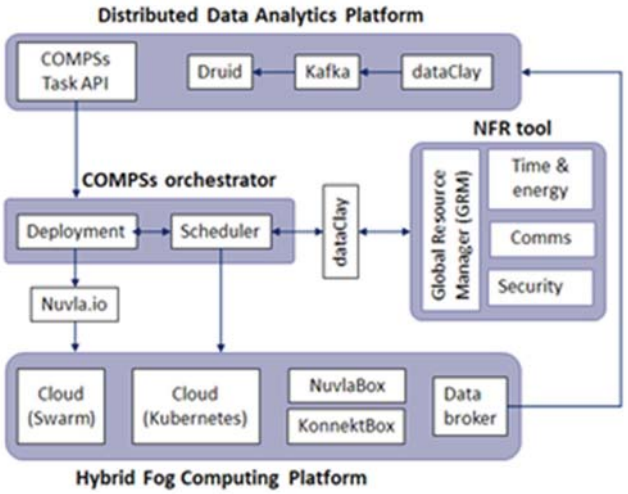


Figure 1. ELASTIC Software Architecture

DataClay is used for storing both application data and internal data for the software architecture management itself.

The Fog Computing Platform is the layer responsible for abstracting the computing resources across the compute continuum, including cloud-based CaaS technologies for resource auto-scaling, IoT cyber-secure communication and network protocols and the underlying highly parallel and energy-efficiency embedded platforms.

Nuvla [5] is both an online open-source platform (Nuvla.io) and an edge platform software (NuvlaBox) developed at SixSq, to transparently manage applications across the compute continuum, providing edge computing as a service. KonnecktBox [6] is an industry-oriented digitization platform, from Ikerlan, built over EdgeXFoundry [7].

The NFR tool is the main purpose of this paper and is presented next.

3 Managing Non-Functional Requirements

The NFR (Non-Functional Requirements) tool is the component which monitors the behaviour of the system and recommends and informs COMPSs of required changes to the system configuration, when detecting specific requirement violations. This real-time monitoring tool is distributed across different edge nodes, monitoring several system properties such as the CPU usage, the energy consumption, the quality of the communications link among the edge devices and between the edge devices and the cloud, and the devices' security [7].

The integration of the tool with COMPSs, is provided through dataClay. The NFR tool reads the system information from dataClay and updates the intended configuration of the system also through the same component.

Figure 2 provides the architecture of the NFR tool. The tool is conceptually constituted by (i) probes, which are applications or system tools of the fog platforms that provide monitoring data, (ii) per property NFR monitor and resource managers, which detect requirements violations, and (iii) a Global Resource Manager (GRM), which proposes changes to the system configuration.

The probes are in charge of interfacing with the underlying platform to collect the required information, which will be published through the Data Router and will be used to detect NFR violations. The NFR Monitor and resource managers are per-property specific components, which, based on the information from the probes, and the application information, determines if some requirement is not being met. There are four NFR Monitors: time monitor, energy monitor, communication quality monitor and security monitor.

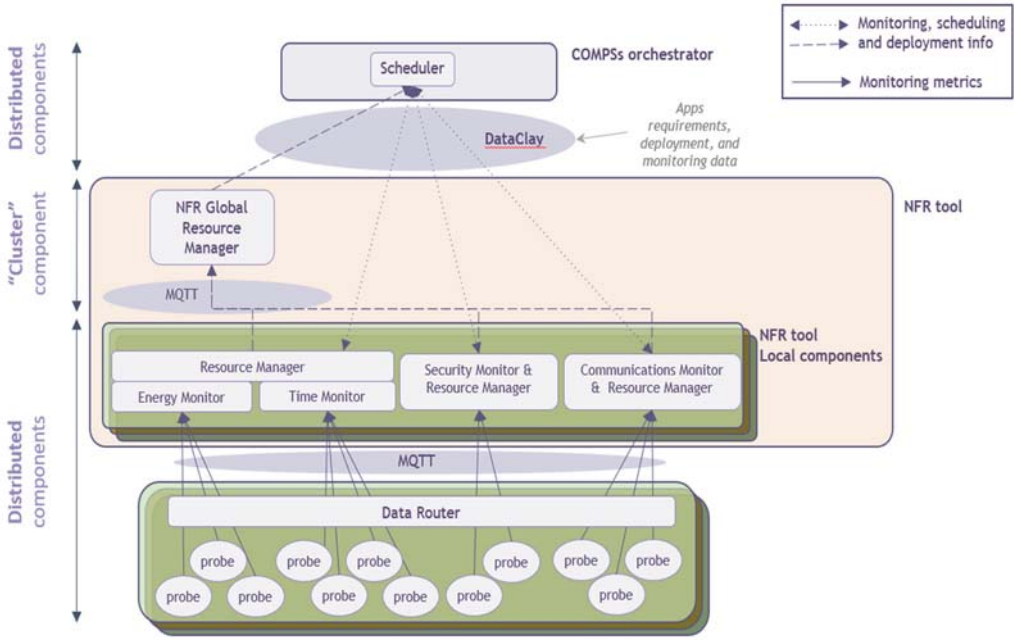


Figure 2. General view of the NFR tool

The GRM is the component that makes recommendations on the allocation of resources to applications; in ELASTIC, the resources are the computing units of system nodes, and the role of the GRM is to recommend any change to the mapping of application workers in the nodes, to guarantee non-function requirements accomplishment within a holistic approach, that is, considering simultaneously all non-functional properties. As decisions are global to system nodes, the GRM is single for a "cluster", a set of nodes working together. An example, from ELASTIC smart city scenarios is a crossing between vehicles and a tram line, with both local nodes (city cameras, traffic lights), as well as nodes in the tram and in the cloud.

The GRM is not in charge of the actual change (this is the responsibility of the COMPSs orchestrator), but solely to update the configuration information in dataClay, which is what the orchestrator reads.

3.1 Time Monitoring

In order to deal with the real-time requirements of applications, it is necessary to monitor different timing properties in all nodes of the ELASTIC system. This ranges monitoring actual CPU utilization and execution time of applications, to detection of deadline violations or memory accesses [2]. The current implementation [9] focuses on the former, fundamental for soft real-time requirements, with the latter being foreseen for hard real-time requirements.

In this implementation, CPU monitoring is obtained using the capabilities of the NuvlaBox platform [5], which provides information on the current load in the nodes, managing workers as Docker containers. This monitoring allows ELASTIC to dynamically adjust the system resources to which the application is mapped, depending on the actual load of the system.

3.2 Energy Monitoring

The energy requirements in the ELASTIC architecture mainly concern the need to monitor the energy consumption of the system nodes, to enable allocation strategies and run-time mechanisms that consider energy information (so that power consumption can be optimized over the complete continuum).

Therefore, the implementation of the energy property focuses on monitoring the energy consumption of each node individually, which may trigger application configuration changes. As with the time dimension, the information is obtained from NuvlaBox.

Time utilization and energy consumption are managed in an integrated form in the current implementation of ELASTIC, since these are two properties that interrelate. Therefore, they share the same resource manager.

3.3 Communication Quality Monitoring

The ELASTIC software architecture must be capable of tracking the communication QoS for every link between each pair of nodes in the system. For this purpose, several metrics are monitored, e.g. roundtrip time (RTT), packet loss rate (PLR), etc. This information is obtained from the Telemetry Daemon of the fog platform KonnektBox.

Such attributes are compared to the corresponding thresholds, which are predefined regarding the specific type of application. In case of any threshold violation, a message is sent to the GRM, which assists COMPSs in order to make the final scheduling decisions. Additionally, the values of data throughput for all communication links are written to dataClay to enable COMPSs to take into consideration additional system metrics. A communication cost function is updated for all workers in the node, helping the GRM to prioritize the applications to reconfigure.

3.4 Security Monitoring

Verifying applications correctly comply with security mechanisms and do not contain vulnerabilities is essential. To guard against security threats, ELASTIC continuously monitors the systems and applications deployed, incorporate security upgrades to software and deploy updates to existing configurations.

The security monitoring is performed by using Wazuh [10], a free and open source host-based intrusion detection system (HIDS). Security requirements differ with other non-functional requirements in the sense that end devices must not initiate the connection to the NFR tool itself. This fact would imply a security risk, as a hypothetical "infected" device could lie about its security status. This detail makes it better to follow a different approach, a central monitor node that controls the security parameters for the rest of the nodes. At the device end, a set of distributed agents feed the centralized tool with the corresponding security audit results. A benefit of this approach is that input files are transferred to the target system and after the scan finishes result files are transferred back. No temporary data remains on the remote machine.

This allows to determine if a node is found to be secure, or if a threat is detected. This information is stored in the dataClay configuration, and a notification is sent to the global resource manager to remove from this node applications which should always execute in a secure environment.

3.5 Global Resource Management

The Global Resource Manager (GRM) is the component which takes into consideration the information from the other NFR components, with the following responsibilities/functionalities:

- Listens to all NFR Monitors for violation notifications.
- Uses simple heuristics to define possible violations treatment.

[2] Memory accesses can be used to provide information on contention accessing shared memory, providing a more accurate timing analysis for hard real-time applications.

- Writes suggestions/recommendations for handling violations on dataClay, assisting COMPSs in the configuration of applications.

Specifically, the GRM maintains a list of all Elastic System Nodes sorted by the highest resources availability to receive tasks, hereafter called List of Sorted Nodes (LSN). This reordering will happen periodically so that the list is always up to date, and thus the GRM suggestions will be faster.

When violations start to be detected in a node, the GRM will gradually reduce the computation load in the node, by reducing the computation units which are provided by COMPSs to the applications with lower priority and lower load. Eventually, if needed, COMPSs is informed that an application must be removed from the node and started in a new node (from the LSN). Security is a special case, since when a node is detected not to be secure, it overrides any other decision, as applications that requires security must immediately be moved to a new node (also from the LSN).

The current implementation provides a reactive behaviour, as a simple, and fast solution. It already provides stable workflows, but real usage will eventually introduce higher dynamics, which can potentially lead the reactive approach to create instability in the operation of the system. Therefore, effort is also placed in the development of a more complex approach [9].

4 Evaluation

The functionalities provided by the Non-Functional requirements components were evaluated in both laboratorial prototypes, as well as partly in the final demonstrator of the project.

The performed evaluation allowed to identify that the performed work allowed to fulfil the greatest majority of the ELASTIC requirements, as initially put forward in [1]. The NFR tool components are able to, together with the orchestrator, manage the non-functional properties of the applications, providing these with the required computing resources, meeting performance needs, and guaranteeing non-functional requirements [11].

The advances and results of the work, as well as of the full ELSATIC project, can be seen in the project final event workshop at [12].

5 Conclusions

The ELASTIC software architecture implements an elasticity concept, which enables the support to performance requirements of extreme-scale analytics workloads in an edge-cloud computing continuum. An important aspect of the architecture is how it manages the fulfilment of non-functional properties of the system (real-time, energy efficiency, communication quality and security). This paper briefly describes the architecture and current implementation of the ELASTIC software architecture components which are dedicated to the management of non-functional requirements.

Acknowledgements

This work has been financially supported by the European commission through the ELASTIC project (H2020 grant agreement 825473).

References

[1] L. Nogueira, A. Barros, C. Zubia, D. Faura, D. Gracia Pérez, L. M. Pinho, “Non-functional Requirements in the ELASTIC Architecture”. Proceedings of the Workshop on Challenges and New Approaches for Dependable and Cyber-Physical Systems Engineering 2019. Ada User Journal. 2020, Vol 41(1), pp. 51-56.

[2] M. A. Serrano, C. A. Marín, A. Queralt, C. Cordeiro, M. Gonzalez, L. M. Pinho, and E. Quiñones, An Elastic Software Architecture for Extreme-Scale Big Data Analytics in E. Curry et al. (eds.), Technologies and Applications for Big Data Value, 2022, https://doi.org/10.1007/978-3-030-78307-5_5.

[3] F. Lordan et al., “ServiceSs: An Interoperable Programming Framework for the Cloud,” J. Grid Comput., vol. 12, pp. 67–91, 2013, doi: 10.1007/s10723-013-9272-5.

[4] J. Martí, A. Queralt, D. Gasull, A. Barceló, J. José Costa, and T. Cortes, “Dataclay: A distributed data store for effective inter-player data sharing,” J. Syst. Softw., vol. 131, pp. 129–145, 2017, doi: 10.1016/j.jss.2017.05.080.

[5] SixSq, “Edge and Container Management Software”, https://sixsq.com/products-and-services/nuvla/overview (accessed June 2022).

[6] Ikerlan, Ikerlan KonneKt, https://www.ikerlan.es/en/ikerlankonnekt (accessed April, 2022).

[7] EdgeXFoundry, The Preferred Edge IoT Plug and Play Ecosystem – Enabled Open Software Platform, https://www.edgexfoundry.org/ (accessed June 2022).

[8] J. Fanjul, X. Perez, R. Sousa, A. Barros, E. Kartsakli, “D4.3 Non-functional components”, ELASTIC project (H2020 grant agreement 825473), October 2021.

[9] R. Sousa, L. Nogueira, F. Rodrigues, L. M. Pinho, “Global Resource Management in the ELASTIC Architecture”, 5th IEEE International Conference on Industrial Cyber-Physical Systems (ICPS 2022).

[10] Wazuh, The Open Source Security Platform, https://wazuh.com/ (accessed June 2022).

[11] R. Sousa, A. Barros, L.M. Pinho, M. Gonzalez, E. Sabate, Elli Kartsakli, D. Faura, M. C. Zubia, “D4.4 Evaluation of non-functional components”, ELASTIC project (H2020 grant agreement 825473), May 2022.

[12] ELASTIC Final Dissemination Event, May 31, 2022, https://www.youtube.com/watch?v=CpNv9VGRlEI, (accessed June 2022)..

Containers for Specification in SPARK

Claire Dross
AdaCore, 46 rue d'Amsterdam, 75009 Paris, France ; email: dross@adacore.com

Abstract

The SPARK tool analyzes Ada programs statically. It can be used to verify both that a program is free from runtime exceptions and that it conforms to a specification expressed through contracts. To facilitate dynamic analysis, Ada contracts are regular expressions which can be evaluated at execution. As a result, the annotation language of the SPARK tool is restricted to executable constructs. In this context, high-level concepts necessary for specification by contracts need to be supplied as libraries. For example, the latest version of the Ada language introduces unbounded integers and rational numbers to the standard library. In this article, we present the functional containers library which provides collections suitable for use in specification. We explain how they can be used to specify and verify complex programs through concrete examples that have been developed over many years.

Keywords: containers, specification, program verification.

1 Introduction

With software taking on increasingly large roles in critical embedded systems, it has become critical to verify software in an efficient way. This leads more and more industrial software companies to deploy formal verification techniques [1, 2]. The SPARK tool [3] performs static analysis of Ada programs. It can be used to verify that a program is free from runtime exceptions, including but not limited to division by zero, buffer overflows, null pointer dereferences, etc. High-level functional properties can also be verified by the tool. These properties need to be expressed as contracts - pre and postconditions, type invariants, etc.

The SPARK tool performs *deductive analysis*: It takes as its input an Ada program, annotated with contracts, and generates from it logical formulas, called *verification conditions*. These verification conditions are then given to automated solvers. If all the conditions are verified, then the Ada program correctly implements its contracts. Deductive analysis works modularly on a per-subprogram basis[1], using the subprogram's contract to summarize its behavior while analyzing callers. As a result, it is necessary for the user to manually annotate her subprograms with contracts for the tool to work. For both the analysis and the annotation process to remain tractable, some features of Ada have to be restricted; the SPARK toolset rejects Ada programs including these features as being non-conformant. In particular, SPARK does not support side-effects in expressions (but they can occur in statements) nor aliasing (when modifying one object can change the value of another object).

Since 2012, contracts are part of the Ada language. They are mostly used for dynamic analysis and can be verified at runtime. Therefore, they have the same semantics as regular Ada expressions. In the SPARK tool, we keep the executable semantics of contracts. It makes it easier for developers to write the contracts, both because they do not have to learn a new language, and because the contracts can be tested and debugged like normal code. However, it has the side-effect of restricting the annotation language to executable constructs. To alleviate this limitation, high-level concepts necessary to write certain specifications can be added as libraries. Unbounded integers and rational numbers have been introduced recently into the Ada runtime. They can be used to avoid overflows in contracts, or to reason about the rounding error in floating point computations [4].

Another concept which is commonly used in specification is a collection: set, sequence, map etc. Collections used for specification are different from their counterparts used during development. They are more of a mathematical concept, and less concerned about efficiency. In this article, we present the *functional containers library* which was introduced for this purpose in 2016. We explain how it can be used to enhance the specification and verification of complex programs through concrete examples that have been developed through the years.

2 The Formal and Functional Containers

The standard library of Ada provides implementations of commonly-used standard containers: vectors, doubly-linked lists, as well as sets and maps, both ordered and hashed. These containers come in various flavors: bounded to avoid dynamic allocations on embedded systems, indefinite to hold elements of variable sizes etc. To allow for efficient access, these containers implement a notion of iterators, named *cursors*. Cursors are basically pointers giving direct access to an element in the container. They provide an easy way to iterate over all the elements of a container. While cursors are desirable in terms of usability, they are unfortunately not compatible with the restrictions imposed on input to the SPARK tool. Indeed, each cursor involves an alias of the container it belongs to, as modifying the container might cause the cursor to become invalid or designate a different element, and SPARK does not support aliases.

[1] In Ada, *subprogram* is a generic term meaning a function or a procedure.

To alleviate this issue, SPARK-compatible versions of the standard containers [5] have been implemented. They are called *formal containers* and are designed to be as close to the standard containers as possible. They provide cursors like the Ada containers, but these cursors are nothing more than indexes in an array constituting the underlying memory of the container. As a result, the formal container API is slightly different from the standard one, as the container needs to be passed along with the cursor to determine its validity or access the corresponding element, as can be seen in Figure 1. Note that the formal containers are not themselves verified using SPARK, but they are compatible with its restrictions and their primitives have been annotated with contracts allowing user code to be analyzed.

```
function Element
  (Position : Cursor) return Element_Type;
-- Function to access an element in a standard map
function Element
  (Container : Map;
   Position  : Cursor) return Element_Type
with Pre  ⇒ Has_Element (Container, Position),
     Post ⇒ ...;
-- Function to access an element in a formal map
```

Figure 1: The `Element` function is used to access an element in a standard or a formal map. As the cursors no longer hold a reference to a container in the formal container library, the `Element` function takes the container as an additional parameter. It is annotated with a pre and a postcondition that can be used to verify user code.

At the beginning, we attempted to use the formal containers in high-level specifications, but we quickly found out that it was not tractable. Indeed, these containers are onerous for verification, as they pull with them numerous secondary considerations, like the order of iteration, or the validity and position of cursors. A new library of containers, named *functional containers*, was introduced to alleviate this issue. They are designed to be light-weight in terms of specifications. They only offer a small number of functional operations, with as few constraints as possible. They are unbounded, might contain any kind of elements (even with variable sizes) and can be used easily (no need to provide a hash or compare function for sets and maps in particular).

The functional containers library provides sets, maps, and sequences. Their API consists of functions creating new containers, as opposed to procedures modifying existing ones. Even if they are mostly used for specifications, these containers are executable. To remain reasonably efficient, their implementation involves several levels of sharing. Quantification over these containers is possible, but iteration only makes sense for sequences that define a relevant order on their elements. Figure 2 shows a part of the API of functional maps. They only offer three properties, a function `Has_Key` to check whether a key has an association in the map, a function `Get` to retrieve this association, and a function `Length` returning the number of keys with an association in the map. Other primitives are specified in terms of these properties, like the "$\leq$" operator which returns true when all associations in a map are also included in the other.

```
function Length (M : Map) return Big_Natural;
function Has_Key
  (M : Map;
   K : Key_Type) return Boolean;
function Get
  (M : Map;
   K : Key_Type) return Element_Type
with
  Pre ⇒ Has_Key (M, K);
function "≤" (M1, M2 : Map) return Boolean with
  Post ⇒ "≤"'Result =
    (for all K of M1 ⇒ Has_Key (M2, K)
       and then Element (M1, K) = Element (M2, K))
```

Figure 2: Part of the API of functional maps. The functions `Length`, `Has_Key`, and `Get` are the only basic properties of a functional map. All other primitives, like "$\leq$" here, are specified in terms of these properties.

There is no special handling for these containers in the verification tool. The container type and its primitives are treated as an abstract type with uninterpreted functions. The only information known about the type and its primitives are those coming from the Ada contracts in the functional container API.

3 Specifying Data-Structures

When the functional containers were designed, the first objective was to specify the formal containers library. Indeed historically, the formal containers were axiomatized in WhyML, the input language of the Why3 tool used as part of the SPARK backend [6]. This required special handling so the formal containers were recognized specifically and linked to the correct WhyML module. This mechanism had the advantage of making it possible to use the rich specification features offered by WhyML (abstract logic functions, unrestricted quantifiers, axioms etc.) most of which cannot be mirrored in SPARK as they are not executable. However, the maintenance cost was prohibitive, as the mechanism had to be kept up-to-date through successive updates of both SPARK and WhyML.

Replacing this special handling by regular SPARK contracts without degrading the provability was a challenge. We decided to go for *model functions*, returning functional containers. A model function is a *ghost* function, meaning it can only be used in specifications. It takes as a parameter a concrete object and returns its model: another object, generally simpler to reason with. The operations on the concrete object are then described in terms of their effects on the abstract model. In Figure 3, the model of a ring buffer is a sequence giving the elements of the buffer in the order in which they will be retrieved. Using this model, its primitive operations can be specified in a straightforward way.

As the formal containers are relatively complex, we decided to use several model functions for their specification. Each formal container provides a main model function called `Model` which returns a functional container giving a high-level view of the data-structure. We use sequences for vectors and doubly-linked lists, and functional sets and maps for ordered and hashed sets and maps respectively. Unfortunately, this

```
function Model (R : Ring_Buffer) return Sequence
with Ghost;

procedure Enqueue
  (R : in out Ring_Buffer;
   E : Integer)
with
  Pre  => not Is_Full (R),
  Post => Model (R) = Add (Model (R)'Old, E);
  --  The new model of R is its old model with E
  --  added at the end.

procedure Dequeue
  (R : in out Ring_Buffer;
   E : out Integer)
with
  Pre  => not Is_Empty (R),
  Post => Model (R) = Remove (Model (R)'Old, 0)
    and then E = Get (Model (R)'Old, 0);
  --  The new model of R is its old model without
  --  the first element. E is set to the first
  --  element of the old model of R.
```

Figure 3: The model of a ring buffer is a functional sequence of elements in the order in which they were added to the buffer. The `Dequeue` and `Enqueue` functions are defined in terms of their effect on the model of their parameter.

```
procedure Set_All_To_Zero (M : in out Map) with
  Post =>
    (for all K of Model (M)'Old =>
       Has_Key (Model (M), K))
    and then
      (for all K of M =>
         Has_Key (Model (M)'Old, K)
         and then Get (Model (M), K) = 0);
```

Figure 4: The postcondition of the procedure `Set_All_To_Zero`. It states that the keys of `M` are preserved by the call and that every key in the map is associated to 0 after the call. This can be expressed using only the `Model` function. The order of iteration and the validity of cursors is not relevant here.

```
pragma Loop_Invariant
  (for all P in 1 .. Get (Positions (M), Cu) - 1 =>
     Get (Model (M), Get (Keys (M), P)) = 0);
```

Figure 5: A loop invariant used to verify the procedure `Set_All_To_Zero`. It uses the `Positions` map to query the position of the current cursor `Cu` and the `Keys` sequence to retrieve the keys situated before this position in the map.

high-level model is not enough to verify subprograms using cursors to iterate over a formal container. Indeed, it does not represent the cursors, nor the order in which the elements occur during an iteration over a set or a map. To alleviate this issue, one or two additional model functions are defined for each container. The `Positions` function returns a functional map which associates the cursors that are valid in a container to an integer standing for their position in the container. For sets and maps, the `Elements` or `Keys` function returns a sequence of elements or keys to model their order in the container. Note that this order is defined both for the hashed and ordered containers, as both define an order of iteration on their elements.

This layered approach allows users of the formal containers library to choose the level of granularity they need. As an example, the procedure `Set_All_To_Zero` sets the elements associated with each key in a map to 0. Its postcondition is given in Figure 4. It only uses the high-level model of the map, as it does not care about the cursors or order of iteration in the container. However, these considerations are necessary when verifying its implementation. The loop invariant in Figure 5 is one of the annotations used to verify the loop setting each element to 0 in its body. It uses the `Positions` function to get the position of the current cursor, and then the `Keys` function to state that the elements associated with all the keys occurring before this position have already been replaced.

4 Verifying Data-Structures

As explained in the previous section, the functional containers can be used as models to annotate subprograms dealing with complex data-structures. Even though it was not done for the formal containers library, it is possible to use SPARK to verify these annotations. For the verification to be possible, it is necessary to describe precisely the link between the underlying data-structure and its model in the postcondition of the model function. In Figure 6, the function `Valid_Model` links the value of a ring buffer implemented as an array with a first index and length field to the value of the sequence which models it. It can be used in the postcondition of `Model`, so the contracts in Figure 3 can be verified.

This method was used successfully in case-studies of various sizes through the years. The first use of the functional containers to verify a SPARK program was developed to showcase the capability in 2016 [7]. It features a simple allocator inside a memory array modeled using a set of allocated cells and a sequence for the free list. A substantially more complex example is the proof of the insertion inside a red-black tree encoded inside a memory array [8]. The complexity of the specification is handled by building the concrete structure incrementally, starting from binary trees, to search trees, to finally implement and verify the insertion in a red-black tree. More recent examples use functional containers to model pointer-based data-structures, which have been supported by the SPARK tool only for the last couple of years [9]. To support this new use-case, the functional containers library had to be updated. Indeed, sequences are bounded by the machine integer type used to index them, and functional sets and maps used to have a theoretical bound on their cardinality due to the machine integer type used for their `Length` function. The restriction on sets and maps was lifted earlier this year by replacing the return type of their `Length` functions by unbounded integers, and a new type of sequence indexed by unbounded integers was introduced in the library.

On complex data-structures, it is possible to use several levels of models to perform a proof by refinement. Basically, a lower-level model, close to the concrete data-structure, will be used to annotate and verify the basic operations. Then, one

```
type Ring_Buffer is record
   Content : Content_Array := (0 .. Max - 1 ⇒ 0);
   First   : Positive range 0 .. Max - 1 := 0;
   Length  : Natural  range 0 .. Max := 0;
end record;

function Valid_Model
  (R : Ring_Buffer;
   M : Sequence) return Boolean
is
  (Length (M) = R.Length
   and then
     (for all I in 0 .. R.Length - 1 ⇒
        R.Content ((R.First + I) mod Max) =
        Get (M, I)))
with Ghost;

function Model
  (R : Ring_Buffer) return Sequence
with
  Ghost,
  Post ⇒ Valid_Model (R, Model'Result);
```

Figure 6: The function `Valid_Model` links the element of the ring buffer to their corresponding value in the model sequence. It is used as a postcondition of the `Model` function. It makes it possible to verify the implementation of `Enqueue` and `Dequeue` procedures presented in Figure 3.

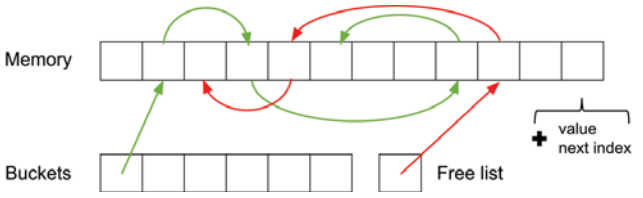


Figure 7: Concrete implementation of a formal hashed set. The content of the set is stored in a bounded memory array. Each cell of this array contains both a value and a `Next` field, used to represent linked lists in the memory. The array of buckets holds the heads of the lists associated to each hashed value. The cells which are not allocated yet are linked together in the same way, and their head is stored separately.

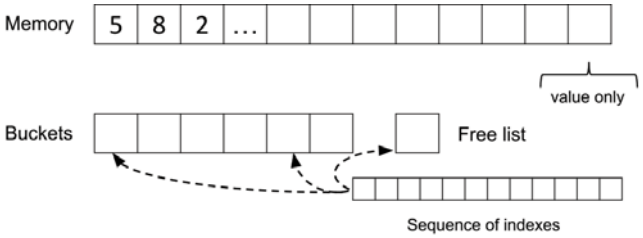


Figure 8: Low level model of a formal hashed set. The values contained in the set are still stored in a bounded array. The linked structure however has been removed from the memory. Instead, each bucket now uses a functional sequence to store the indexes of the corresponding values in the order of iteration. The free list is also represented as a sequence.

or several higher-level models might be introduced to further abstract away the operations. As an example, we are currently working on using SPARK to verify the implementation of the bounded formal hashed sets. As schematized in Figure 7, these sets are implemented inside an array. A hash function is used to choose a bucket for each element of the set. Each bucket is the head of a list implemented through a `Next` field in the memory array. For the verification to remain tractable, we have introduced two levels of models. The lower-level model keeps a memory array, but only to store the values, as represented in Figure 8. The buckets contain functional sequences that store the corresponding allocated indexes. The notion of buckets disappears completely in the higher level model, see Figure 9. It simply represents the set as the memory for values and a big sequence, containing the allocated indexes in the order in which they will be traversed when iterating over the set. The final objective is to be able to verify the specification written in terms of the three model functions `Model`, `Elements`, and `Positions` presented in Section 2.

5 Going Further

Modeling the content of other data-structures is by far the most common use-case for functional containers in our experience. However, more exotic use-cases also exist. In particular, the container might model a state which is not actually present in the program, but represents a concept used in the specification only. It makes it possible to address properties which are not generally in the domain of SPARK. In this section, we focus on two such use cases.

The SPARK tool enforces an ownership policy to ensure non-aliasing when dealing with pointers. As a result, the builtin support for pointers does not allow verifying programs that rely on aliasing. In particular, data-structures involving sharing or cycles - doubly-linked lists, direct acyclic graphs etc. - cannot be handled by the SPARK tool. To work around this restriction, it is possible to hide the pointers and model them as indexes in a memory map. The implementation still uses pointers though, so the map does not represent any actual structure in the code. It makes it possible to reason about pointers with aliasing by annotating explicitly which pointers can be aliases of each other. Using a single memory object standing for all allocated data makes both annotation and verification more difficult however, so the built-in support of pointers stays more efficient when it applies.

Figure 10 shows the contracts provided for the `Allocate` procedure. It allocates a memory region, initialized with the provided value, and returns a pointer to this memory region. Its contract expresses that it modifies a global ghost object called `Memory`. This object is the functional map standing for the model of the actual program heap. In the postcondition of `Allocate`, it is necessary to describe its effect on the whole abstract memory map. We use two universally quantified formulas to state that `P` is the only newly allocated cell, and that the values designated by other pointers are preserved. For comparison, the procedure `Allocate` in Figure 11 uses the built-in support for pointers in SPARK. It is not considered to read or modify any global state as allocated cells are treated as parts of the pointer that owns them, so its contract is far simpler.

As another example, functional sequences can be used to model a restricted form of temporal logic. Here, a history

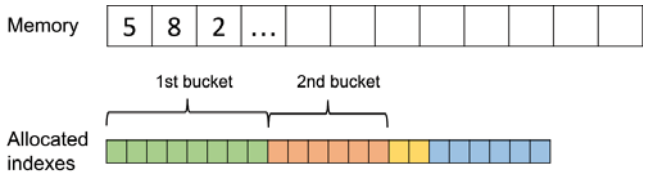


Figure 9: High-level model of a formal hashed set. The memory array of values remains the same as in the low level model. The buckets and the free list are not represented anymore. Instead, we use a single functional sequence containing all the allocated indexes in the order of iteration.

```
type Pointer is private;
procedure Allocate
  (O : Object; P : out Pointer)
with
  Global ⇒ (In_Out ⇒ Memory),
  --  P is a valid pointer in Memory designating
  --  the value O.
  Post ⇒ Has_Key (Memory, Address (P))
  and then Get (Memory, Address (P)) = O
  --  Every pointer previously valid in Memory
  --  remains valid and keeps designating the
  --  same value.
  and then (for all K of Memory'Old ⇒
              Has_Key (Memory, K)
              and then Get (Memory, K) =
                       Get (Memory'Old, K))
  --  P is the only address allocated by the call
  and then (for all K of Memory ⇒
              Has_Key (Memory'Old, K)
              or else K = Address (P));
```

Figure 10: The function `Allocate` allocates a new memory region for its input object `O`. After the call, its parameter `P` is a pointer to this newly allocated region. The fact that `P` is a pointer is hidden from the SPARK tool using privacy. The effect on the program heap is modeled through a ghost `Memory` map.

is represented as a ghost sequence of events, where events would be for example a call to a particular subprogram or the reception of a message. Each time an event occurs, it is added at the end of the sequence. Using the ghost sequence, it is then possible to express properties over the order in which the events occurred. The SPARK tool can be used to verify that these properties are maintained through the program. The snippet in Figure 12 is extracted from the code of OpenUxAS, a framework developed by Air Force Research Labs for mission-level autonomy for teams of cooperating unmanned vehicles [10]. This framework is implemented as several services communicating through message passing. In this example, the history records the emission and reception of messages. The function `No_Route_Request_Lost` uses the history to express that all received messages of kind `Route_Request` have been handled: they are either in the set of pending requests or a response has been sent. As can be seen in the contract of `Handle_Route_Request`, this property is stated both in preconditions and in postconditions of subprograms handling messages in the service. It allows the SPARK tool to verify that it is an invariant maintained by the service.

```
type Builtin_Pointer is access Object;
procedure Allocate
  (O : Object;
   P : out Builtin_Pointer)
with
  -- P is not null and designates the value O
  Post ⇒ P ≠ null and then P.all = O;
```

Figure 11: The function `Allocate` allocates a new memory region for its input object `O`. After the call, its parameter `P` is a pointer to this newly allocated region. As built-in pointers are handled through ownership by the SPARK tool, the part of the program heap designated by `P` is treated as a part of `P` for the verification. Therefore, there is no need to model the memory in the contracts of `Allocate`.

6 Related Work

Collections are generally considered to be useful when writing program specifications. As a result, most verification tools support some kind of specification-oriented collections in their input language. For example, the standard library of the Why3 language used as a backend of SPARK provides polymorphic lists implemented as abstract data types, as well as polymorphic sets and multisets, specified through an axiomatization [11]. Dafny, a verification language developed at Microsoft Research and currently used to verify protocols at Amazon Web Services also provides collections such as sequences and sets with comprehensions [12].

Satisfiability Modulo Theory (SMT) solvers are generally used as a backend of program verifiers. As a consequence, significant research effort has been invested in coming up with efficient decision procedures for collections in these solvers. For example, a decision procedure for a theory of sets has been supported by the CVC4 solver [13] since 2016, and the last version of the prover, cvc5, also supports sequences that can be used to model arrays and vectors [14].

Using simple collections as ghost models of complex data-structures is a widely used technique in proof of programs. As an example, the specification of the Eiffel-Base2 general-purpose containers library verified by Polikarpova et al. uses collections coming from the Mathematical Model Library (MML) : sequences as models of linked lists, maps for hash tables, etc. These collections benefit from custom support in the underlying solver AutoProof [15]. This technique was used successfully to verify numerous data-structures in Why3, such as hashed tables or AVL trees [16]. Blanchard et al. also resorted to using ghost arrays as a model of their lists for their verification of the linked list module of the Contiki OS with the WP plugin of Frama-C [17].

7 Conclusion

Functional containers are used to represent collections in the specification of SPARK programs. They offer a simple, high-level representation of a container, that is both easy to understand for a reader and easy to reason with for the solvers. They are executable, so it is possible to test the specification, or even to use them in actual code.

Through the years, they have been used to annotate or verify various kinds of containers, from the formal container library

```
function No_Route_Request_Lost
  (Pending_Routes : Set) return Boolean
is
  (for all E of History ⇒
      (if E.Kind = Receive_Route_Request then
            Contains (Pending_Requests, E.Id)
         or else Route_Response_Sent (E.Id)));

procedure Handle_Route_Request
  (Data    : Route_Aggregator_Configuration_Data;
   Mailbox : in out Route_Aggregator_Mailbox;
   State   : in out Route_Aggregator_State;
   Request : Route_Request)
with
   Pre ⇒ ...
     --  History invariants
     and then No_Route_Request_Lost
       (State.Pending_Routes)
     and then ...,
   Post ⇒ ...
     --  The request has been added to the history
     and then History'Old < History
     and then
       Get (History, Last (History)'Old + 1).Kind
       = Receive_Route_Request
     and then
       Get (History, Last (History)'Old + 1).Id =
       Request.Request_ID
     --  History invariants
     and then No_Route_Request_Lost
       (State.Pending_Routes)
     and then ...;
```

Figure 12: Extract of the OpenUxAS code base. The function `No_Route_Request_Lost` uses the `History` sequence to express a safety invariant of the service: Every `Route_Request` received by the service is either pending or a response has been sent. The procedure `Handle_Route_Request` performs the treatment when a `Route_Request` is received. It stores the event in the history. Its contract states that it maintains the `No_Route_Request_Lost` invariant.

of SPARK (which are specified with functional containers but not proved) to ownership based recursive data-structures. Proof by refinement, though not natively supported in SPARK, can be achieved using these containers by creating several layers of models.

Going further, functional containers provide a way to describe state which only exists in the specification. It makes it possible to model properties which are not generally in the domain of SPARK, like some restricted form of temporal logic, through a history, or aliasing and simple memory separation.

References

[1] J. Backes, P. Bolignano, B. Cook, A. Gacek, K. S. Luckow, N. Rungta, M. Schaef, C. Schlesinger, R. Tanash, C. Varming, *et al.*, "One-click formal methods," *IEEE Software*, vol. 36, no. 6, pp. 61–65, 2019.

[2] P. W. O'Hearn, "Continuous reasoning: Scaling the impact of formal methods," in *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science*, pp. 13–25, 2018.

[3] J. Barnes, *SPARK: The Proven Approach to High Integrity Software*. Altran Praxis, 2012.

[4] C. Dross and J. Kanig, "Making proofs of floating-point programs accessible to regular developers," in *Software Verification*, pp. 7–24, Springer, 2021.

[5] C. Dross, J.-C. Filliâtre, and Y. Moy, "Correct code containing containers," in *International Conference on Tests and Proofs*, pp. 102–118, Springer, 2011.

[6] J.-C. Filliâtre and A. Paskevich, "Why3—where programs meet provers," in *European symposium on programming*, pp. 125–128, Springer, 2013.

[7] C. Dross and Y. Moy, "Abstract software specifications and automatic proof of refinement," in *International Conference on Reliability, Safety, and Security of Railway Systems*, pp. 215–230, Springer, 2016.

[8] C. Dross and Y. Moy, "Auto-active proof of red-black trees in spark," in *NASA Formal Methods Symposium*, pp. 68–83, Springer, 2017.

[9] C. Dross and J. Kanig, "Recursive data structures in spark," in *International Conference on Computer Aided Verification*, pp. 178–189, Springer, 2020.

[10] M. A. Aiello, C. Dross, P. Rogers, L. Humphrey, and J. Hamil, "Practical application of spark to openuxas," in *International Symposium on Formal Methods*, pp. 751–761, Springer, 2019.

[11] T. W. D. Team, "Why3 documentation," Apr. 2022.

[12] R. L. Ford and K. R. M. Leino, "Dafny reference manual," 2017.

[13] K. Bansal, A. Reynolds, C. Barrett, and C. Tinelli, "A new decision procedure for finite sets and cardinality constraints in smt," in *International Joint Conference on Automated Reasoning*, pp. 82–98, Springer, 2016.

[14] Y. Sheng, A. Nötzli, A. Reynolds, Y. Zohar, D. Dill, W. Grieskamp, J. Park, S. Qadeer, C. Barrett, and C. Tinelli, "Reasoning about vectors using an smt theory of sequences," *arXiv preprint arXiv:2205.08095*, 2022.

[15] N. Polikarpova, J. Tschannen, and C. A. Furia, "A fully verified container library," in *International Symposium on Formal Methods*, pp. 414–434, Springer, 2015.

[16] M. Clochard, "Automatically verified implementation of data structures based on avl trees," in *Working Conference on Verified Software: Theories, Tools, and Experiments*, pp. 167–180, Springer, 2014.

[17] A. Blanchard, N. Kosmatov, and F. Loulergue, "Ghosts for lists: a critical module of contiki verified in frama-c," in *NASA Formal Methods Symposium*, pp. 37–53, Springer, 2018.

Rigorous Pattern Matching as a Language Feature

S. Tucker Taft

AdaCore, Lexington, MA, USA; email:taft@adacore.com

Abstract

Structural pattern-matching as a language feature has become more common in programming languages over the past decade. This paper will consider more generally the challenge of adding pattern matching as a programming language feature, from the points of view of language design, rigorous static error detection, and effectiveness. In this context, a pattern matching language feature can be seen as providing a more rigorous approach to handling the complex conditionals that arise in processing highly structured input.

Keywords: Pattern matching, language design, static error detection, rigorous software development.

1 Introduction

Many programming languages now include a pattern-matching feature, often introduced with the keyword **match**, e.g. OCaml [1], Python [2], Haskell [3] (Haskell doesn't require any keyword -- every function is considered a pattern match). These are not primarily focused on string pattern matching, but more on structure pattern matching, where the matching starts from an object of some structured type, and the individual patterns select particular structural patterns for specified actions.

These pattern matching features can be seen as a generalization of the **case** or **switch** statement available in most third-generation programming languages. But they typically include the ability to associate an identifier with some or all of the pattern, which is then usable inside the handler for the given pattern, knowing that that identifier refers to some part of the original object that satisfies the given part of the pattern.

One of the great benefits of a pattern matching language feature is that it can be used to implement logic that otherwise might require a long **if/elsif/elsif/…/else** chain. The pattern-matching equivalent to such a chain will generally be easier to read, understand, and maintain.

Furthermore, it is possible to impose additional rules on pattern matching (including when it is something as simple as a conventional **switch/case** construct) that will foster more rigorous software development processes, and thereby allow more complete program verification at compile time. Here are the three most important such properties:

1. Complete/Exhaustive -- Every possibility is covered by some pattern.

   This is easily accomplished if the programmer includes a final *catch-all* pattern (e.g. **others/default** option), but it may be valuable in some cases for the programmer to omit such a *catch-all* and have the compiler complain unless the other more specific patterns provided cover all interesting cases.

2. Unambiguous -- There should be no two patterns where a given object could match both, unless the pattern that comes lexically second covers a strict superset of objects of the earlier pattern. The later, superset pattern is analogous to the **others/default** branch of a **case** or **switch** statement, which in many languages is required to come after all more specific patterns.

3. Nonredundant -- There are no patterns which are redundant/useless, in that no object will *reach* that pattern, since it is fully covered by earlier patterns.

   This would be analogous to an **others/default** branch of a **case/switch** statement that is never reached, and could be considered misleading. Violating this property might be treated more as deserving a warning rather than an error, at least for a true *catch-all* pattern. For a pattern that is not a true *catch-all* but which is being used as a *fall-back* pattern for one or more earlier more specific patterns, if this *fall-back* is never reachable, then it might well be considered a *redundancy* error.

This paper will consider more generally the challenge of adding pattern matching as a programming language feature, but will focus on the additional rigor provided by the guarantees inherent in the enforcement of the three "desirable properties" identified above.

2 Syntax and Semantics

A pattern matching language construct is generally introduced with the reserved word **match**, though some languages (e.g. Java [6]) are choosing to extend an existing **switch/case** construct rather than introducing a wholly new pattern matching construct. In some languages (e.g. Haskell [3]), and in some contexts (e.g. the **if let** construct in Rust [7]), no special keyword is used – pattern matching is an implicit part of the syntax. In all cases, a pattern is specified using a particular set of pattern components, generally allowing *wildcard* (irrefutable) components, and generally allowing the ability to *bind* a new identifier to some or all of the matched pattern. After the pattern, some action is specified, typically using the normal expression or statement syntax of the language, with the environment augmented with any bindings that have been made to parts of the matched pattern. Patterns typically allow multiple alternatives, separated by '|' or other analogous character, with the same identifiers (if any) required to be bound in each alternative.

Languages vary in terms of how rich is the pattern sublanguage, but for a language that has the notion of variants or multiple constructors for the same type, the pattern sublanguage certainly supports matching against the particular variant or constructor represented by the object being examined. Some languages support matching against subranges of scalar types, and some support matching against arrays, vectors, or arbitrary structures [1,2,3,6,7].

Historically, pattern matching has been associated with textual strings, usually making use of regular expressions as the basis for the matching, with AWK being an example of a language where such pattern matching drives the program logic [8]. In general-purpose programming languages, particularly functional programming languages such as Haskell, more emphasis has been placed on matching against structured data types, with the use of regular expressions, if any, typically limited to array/vector-type structures.

As indicated above, one of the main advantages of including pattern matching as a language feature is that it can help reduce the number of long, potentially confusing, and difficult to verify **if/elsif/…/else** chains. One criterion for evaluating a pattern matching feature is therefore its ability to replace such long **if/elsif** chains with a clearer and more rigorously checked pattern matching construct. We will use code samples in this paper to evaluate the effectiveness of pattern matching features in accomplishing this goal.

2.1 Pattern Matching against Abstract Data Types

In many languages, pattern matching is limited to matching against the visible properties of data types, such as their value if an integer, their visible components (and run-time type identification, if any) if a record, struct, or class, and the specific components of an array or map. Typically any properties that require invoking a query function of some sort are not available for direct matching, and must be handled with an additional boolean condition on the match (e.g. OCaml [1]). Alas, this approach can defeat some of the rigor provided by pattern matching, by re-introducing some of the problems of **if/elsif** chains, where the success of an earlier condition can hide a more appropriate match that might occur later. We therefore investigate approaches that allow uses of query functions for abstract properties, directly within patterns.

2.2 Proposed Pattern Matching Syntax

In this paper we suggest the following basic syntax for pattern matching, generalizing on the **case** construct of Pascal or Ada [9], and incorporating capabilities from languages like OCaml [1] and Python [2].

```
case_construct ::=
  case expression {, expression} is
    when pattern {, pattern} =>
      seq_of_statements
   {when pattern {, pattern} =>
      seq_of_statements}
   [when others =>
      seq_of_statements]
  end case;
```

```
pattern ::=
   simple_pattern {| simple_pattern}

simple_pattern ::=
    [identifier :] unlabeled_pattern
  | <identifier>

unlabeled_pattern ::=
    <>
  | static_expression
  | static_range
  | subtype_mark
  | property_pattern
  | sequence_pattern
  | map_pattern

property_pattern ::=
  ( [ pattern with ]
    property => pattern{,
    property => pattern} [,
    others => pattern] )

sequence_pattern ::=
  [ pattern [*|+]{,
    pattern [*|+]} [, others => pattern] ]

map_pattern ::=
  [ key_expression => pattern {,
    key_expression => pattern } [,
    others => pattern] ]

property ::=
    component_name
  | function_name
  | function_name
       ( seq_of_actual_parameters )
  | abs
  | mod expression
  | rem expression
```

In the above BNF, `expression`, `seq_of_statements`, `range`, and `subtype_mark` are presumed to be already defined for the base language. If there are multiple `expressions` in the initial part of the **case** construct, then there should be the same number of `patterns` in each arm of the **case** construct. A **when others** is allowed as the final arm, which acts as a catch-all matching any set of input values.

Within a pattern, the <> notation is used for a wildcard (irrefutable) pattern. If an identifier is specified either using the `identifier` : syntax or the `<identifier>` syntax, a new variable is introduced that is visible in the corresponding `seq_of_statements`, denoting the part of the input `expression` that the identified part of the pattern matches. If a `pattern` includes a set of `simple_patterns` separated by | then the `pattern` matches if any of the `simple_patterns` match.

Matching against a `subtype_mark` can be used when the `expression` being matched is polymorphic ("class-wide" in Ada parlance), or when the `subtype_mark` incorporates some constraint or predicate relative to its type (though the nature of these constraints or predicates would be limited to what could have been expressed with one or more `unlabeled_patterns`).

The property_pattern is used for types with named components, or types for which abstract properties are of interest. The optional pattern with part allows a pattern for the value as a whole to be given in addition to patterns for each of the properties or components. The optional final **others =>** pattern specifies a pattern that applies to any visible components not mentioned previously. Unless the pattern is <>, all such components must be of the same type. An example of a property_pattern, representing a pattern matching a binary expression with operator plus and right operand zero, would be:

```
(Binary with Operator => Plus,
             Left => <L>, Right => 0)
```

In the seq_of_statements associated with such a pattern, the variable L would denote the left operand subtree of this binary expression. Note that Operator, Left, and Right could be component names or function names. If function names, they would be expected to denote functions with a single argument of (sub)type Binary.

In addition to a component or single-parameter function name, a property can be a function with multiple parameters, the first of which must be of the type being matched (and which is then omitted from the seq_of_actual_parameters). In addition, there are three operators that are allowed as properties, the unary **abs**, and the binary **mod** and **rem** operators, where the expression given to the binary operators is the divisor. **Mod** and **rem** are operators that return the remainder after truncating toward negative infinity, and toward zero, respectively. These operators are interesting because they reduce the variability of the input to a simpler range, which can be useful for pattern matching. For example, to match only even numbers that are of the form 3K + 1, in the range 0 .. 100, the following pattern could be used:

```
Val:(0 .. 100 with mod 2 => 0, mod 3 => 1)
```

In the seq_of_statements associated with this pattern, the variable Val would be bound to the value matching the pattern.

The sequence_pattern is used for arrays and array-like structures such as vectors, and provides rudimentary support for regular expressions over sequences of values. Each pattern matches some number of components. A suffix of * means that the pattern can match zero or more sequential components. A suffix of + means the pattern can match one or more sequential components. An optional final **others =>** pattern indicates there might be further components, all of which must match the given pattern. An example of this form, corresponding to a sequence of characters matching the C hex integer syntax, would be:

```
['0', 'x' | 'X', others =>
 '0'..'9' | 'a' .. 'f' | 'A' .. 'F']
```

The map_pattern is used for a map-like structure, where the result of retrieving the element from the map with each specified key matches the specified pattern. An optional final **others =>** pattern in the pattern means there are allowed to be additional elements in the map, all expected to match the specified pattern. An example of a map_pattern, corresponding to a retrieval of the optimization level from a command-line argument map, could be:

```
["-O" => Optim_Level:0 .. 3, others => <>]
```

In the seq_of_statements associated with this pattern, the variable Optim_Level would denote the value retrieved from the argument map, using the key "-O". The argument map may contain other elements, having any value.

2.3 Red-Black Tree Balancing Example

One frequent example of the power of pattern matching is for balancing a red-black tree, suggested by Chris Okasaki in [14]. This example can be represented using the proposed property_pattern, as follows:

```
case T is
  when (Node with Color => Black,
        Left => (Node with Color => Red,
                 Left => (Node with Color => Red, Left => <A>, Value => <X>, Right => <B>),
                 Value => <Y>, Right => <C>),
        Value => <Z>, Right => <D>)
     | (Node with Color => Black,
        Left => (Node with Color => Red, Left => <A>, Value => <X>,
                 Right => (Node with Color => Red, Left => <B>, Value => <Y>, Right => <C>)),
        Value => <Z>, Right => <D>)
     | (Node with Color => Black, Left => <A>, Value => <X>,
        Right => (Node with Color => Red,
                  Left => (Node with Color => Red, Left => <B>, Value => <Y>, Right => <C>),
                  Value => <Z>, Right => <D>))
     | (Node with Color => Black, Left => <A>, Value => <X>,
        Right => (Node with Color => Red, Left => <B>, Value => <Y>,
                  Right => (Node with Color => Red, Left => <C>, Value => <Z>, Right => <D>)))
    => return New_Node (Red, New_Node (Black, A, X, B), Y, New_Node (Black, C, Z, D));
when <X> => return X;
end case;
```

In the above example, there are four (top-level) property_patterns in the '|' combination pattern for the first alternative of the **case** construct, each of which matches a different (unbalanced) tree structure and binds the variables A, B, C, X, and Y, which are then used in the associated **return** statement to build a balanced subtree. The second alternative is a fall back, where we use an irrefutable pattern that binds the input as the variable X, and just returns that input value.

3 Rigorous Error Detection

Languages with a pattern matching construct differ in how they handle the three desirable properties identified above, namely being exhaustive, unambiguous, and nonredundant. In some languages, leaving some inputs unhandled simply means a default action occurs, such as a null statement (e.g. C's switch statement). In others, a run-time error is signaled, and possibly a warning at compile time (e.g. OCaml [1]). And still others make it a compile time error if the set of patterns is not exhaustive (e.g. Java [6]).

Languages differ even more in how they deal with potentially ambiguous patterns. In several languages, the first match is used, even if there exists a "better" match later (e.g. Java[6]). In part this is because the pattern sublanguage does not fully define what it means for patterns to be ambiguous, particularly in the presence of additional conditions (or "guards" as Java[6] calls them). Some languages require that if a pattern clearly matches a superset of the values matched by another pattern occurring later, the second pattern is flagged at compile-time as being unreachable. But if the two patterns overlap, but neither "dominates" the other, then no error is signaled, and the textual order determines which pattern matches. So such languages are enforcing the Nonredundant property, but not the Unambiguous property. If switching from long **if/elsif/.../else** chains to pattern matching is going to add rigor to the software development process, then the Unambiguous property is an important one to enforce.

3.1 Related Work on Pattern Error Detection

There has been some work on determining how to identify ambiguity (and exhaustiveness) given particular pattern sublanguages. One widely cited paper by L. Maranget [4] about OCaml, describes several methods to provide useful warnings, adopting a matrix representation for a set of patterns. In another earlier paper, related to Standard ML and Haskell, M. Pettersson recognizes that pattern matching can be formulated as a finite automaton, though his emphasis is more on efficient code generation than on the quality of the warning messages provided, and the starting point is again a matrix representation for a set of patterns. A more recent paper by M. Fahnrich and J. Boyland focuses on defining a pattern sublanguage that supports full compile-time checking [13]. The actual algorithms used to provide pattern-matching-related warnings for most languages supporting pattern matching are not described in the literature, as far as we could determine.

3.2 General Automata Approach to Error Detection

Because of our emphasis on the rigor provided by pattern matching, presuming the three desirable properties are enforced, and because of the open-ended nature of our property-based patterns, we have investigated the systematic use of finite automata for pattern-matching error detection, without first translating to a matrix representation. Fundamentally we see a pattern matching construct as a machine for classifying a tree of values into one of several buckets. The matching process can quite directly be represented as a non-deterministic finite automaton (NFA), which can then be translated into a deterministic finite automaton (DFA) using the standard Rabin-Scott powerset construction approach [11]. If appropriate, the DFA can be further minimized using Hopcroft's algorithm [12].

For our purposes, it turns out that the straightforward process for converting an NFA to a DFA can directly provide the information needed to enforce the three "desirable" properties. This process proceeds as follows:

1. If there is no **others** alternative, add one with action "Error -- nonexhaustive coverage".
2. Construct the NFA directly from the patterns, with a separate end state for each distinct case alternative (seq_of_statements).
3. Construct the DFA from the NFA, making no *preferences* so that at each (merged) end-state you have the *set* of case alternatives that *match* the corresponding input.
4. Look at the end-states which include more than one case alternative:
    a. if all states that have alternative A also have a later alternative B, then this implies that B is a superset of A, and we can safely remove B from all such states (**others** is by definition a superset of all non-**others** alternatives);
    b. after removing all superset alternatives from such combined states, if we have any states with multiple alternatives, we have improper ambiguity.
5. If there is still a state that identifies the **others** alternative, the construct is nonexhaustive, if this alternative was inserted by step 1 above as an *error* action.
6. If there is an alternative (other than the **others** alternative) that has no end state, then that is a redundant alternative.

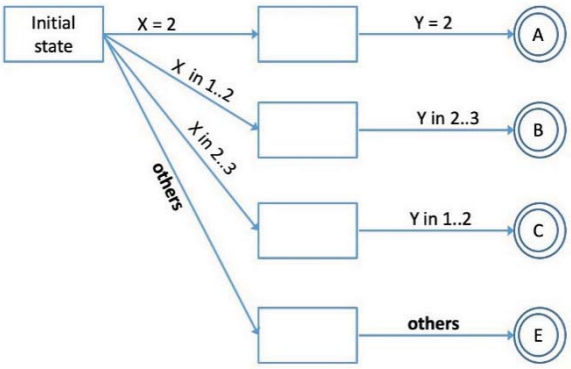
Here is an example of applying the above process, using a simple pattern match against two integers X and Y, presuming:

```
X, Y: Integer range 1 .. 3;
...
case X, Y is
  when 2, 2       => A;
  when 1..2, 2..3 => B;
  when 2..3, 1..2 => C;
end case;
```

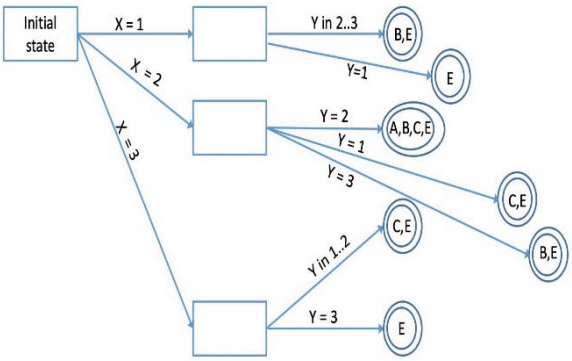
The six-step process described above, proceeds as follows for this example:

1. Add a **when others** => E; (error) alternative because there is no explicit **others** alternative.

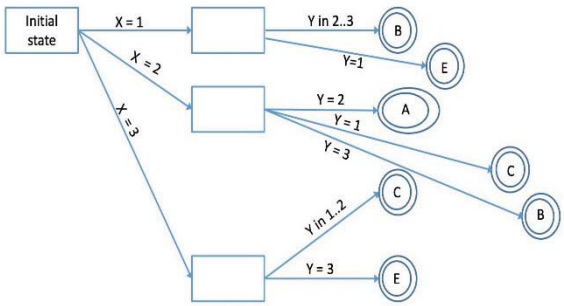
2. Construct the NFA:



3. Convert the NFA to a DFA:



4. Remove the superset alternatives from the ambiguous states, using superset relationships E >= {A,B,C} and {B,C} >= A, to produce this *winnowed* set of end states:



➔ No ambiguous end-states left, so construct is *unambiguous*

5. We have some inputs that reach (E), so since there was no explicit **others** in the original construct, the construct violates the exhaustiveness property.

6. Every non-**others** alternative has at least one end state, so construct is *not redundant.*

In the above **case** construct example, if we omit the A alternative, then we would end up with the end-state for X,Y inputs of [2,2] being (B,C), which would indicate unresolved *ambiguity*.

4 Replacing if/elsif/elsif/.../else chains

As indicated above, one of the goals of adding support for pattern matching is to enable a more rigorous approach to handling complex conditionals. Here is a snippet of a real subprogram, produced by selecting lines with **if**, **elsif**, or **else** on them, to illustrate typical complex conditional logic:

```
if Is_Add_To_Constant (Val1)
elsif Is_Subtract_Of_Constant (Val1)
elsif Is_Multiply_By_Constant (Val1)
   if VN_Equal (Second_Operand (Val1), Val2)
   elsif Is_Multiply_By_Constant (Val2)
      if ...
      elsif VN_Equal (...)
elsif Is_Subtract (Val1)
   if VN_Equal (Val2, Second_Operand (Val1))
   elsif Is_Subtract (Val2)
      if ...
      elsif VN_Equal (...)
   elsif Is_Add (Val2)
      if ...
```

If we try to replace this with an equivalent pattern matching construct, we quickly notice that the various Boolean predicates such as Is_Add_To_Constant and Is_Subtract_Of_Constant do not work well with an attempt to provide an unambiguous, exhaustive set of patterns, because the semantic relationship between these predicates is not clear. If one is True, is the other necessarily False?

What we really want are query functions that produce an enumeration of mutually-exclusive possibilities, such as one that returns the particular operator that is present, coupled with a query function that indicates whether one of the parameters is a compile-time-known constant, or is rather a dynamic value. If we presume such new query functions, we arrive at an equivalent set of patterns that can more likely be analyzed for ambiguity and exhaustiveness:

```
case Val1, Val2 is
   when (Op => Add, Has_Const => True), <> => ...
   when (Op => Subtract, Has_Const => True), <> =>...
   when (Op => Multiply, Has_Const => True),
      (VN_Equal (Second(Val1)) => True) => ...
   when (Op => Multiply, Has_Const => True),
      (Op => Multiply, Has_Const => True) => ...
   when (Op => Subtract, Has_Const => False),
      (VN_Equal (Second(Val1)) => True) => ...
   when (Op => Subtract, Has_Const => False),
      (Op => Subtract) => ...
   when (Op => Subtract, Has_Const => False),
      (Op => Add) => ...
end case;
```

The claim we would make is that the process of transforming the logic to pattern matching, including defining new query functions, produces a more understandable and maintainable result.

5 Conclusions and Future Work

With compile-time checking of desirable properties, pattern matching can provide a more rigorous approach to complex conditional logic. Pattern matching simplifies understanding and maintenance because the individual patterns can be more independent, meaning patterns can be understood individually, and patterns can be added or removed without significantly disturbing the interpretation of the other patterns.

Compile-time checking for desirable properties can be implemented by using the results of converting an NFA for the pattern matching construct into a DFA, and then

the pattern matching construct into a DFA, and then winnowing the states by removing superset states, leaving a DFA where any ambiguities, redundancies, or non-exhaustiveness are simple to identify.

The ultimate goal is to replace complex conditionals using **if/elsif/.../else** chains with pattern matching constructs. When performing simple experiments with existing code, it becomes clear that such a transformation may require a new set of query functions to expose the exhaustiveness of the alternatives in a pattern match.

In the near future, we plan to formalize this proposal as a possible generalization of the Ada programming language **case** construct [9]. A prototype implementation of the compile-time checks is now underway, and will be finalized as part of implementing support for this feature.

Acknowledgments

This work originated with a Pattern Matching language study group within AdaCore, including colleagues Raphaël Amiard, Claire Dross, Stephen Baird, Romain Béguet, and Boris Yakobowski.

References

[1] X. Leroy, D. Doligez, A. Frisch, J. Garrigue, D. Rémy and J. Vouillon, Chapter 9, The OCaml Language, Section 6, Patterns, *The OCaml Manual,* https://ocaml.org /manual /patterns.html, retrieved 25-Feb-2022, 2021.

[2] Python Software Foundation, 8.6 The Match Statement, *The Python Language Reference*, https://docs.python.org/3/reference/compound_stmts.html#the-match-statement, retrieved 25-Feb-2022.

[3] Simon Marlow (ed.), 3.17 Pattern Matching, *Haskell 2010 Language Report*, https: //www.haskell.org /onlinereport /haskell2010 /haskellch3.html, retrieved 25-Feb-2022.

[4] L. Maranget, Warnings for pattern matching. *Journal of Functional Programming, 17*(3), 387-421, 2007. doi:10.1017/S0956796807006223, http://moscova.inria.fr/~maranget/papers/warn/index.html, retrieved 7-Jul-22.

[5] L. Fabrice e Fessant and L. Maranget, Optimizing Pattern Matching. *ACM SIGPLAN Notices*. 36. 10.1145/507635.507641, 2001.

[6] G. Bearman, JEP 427: Pattern Matching for switch (Third Preview), OpenJDK, https://openjdk.org/jeps/427, retrieved 7-Jul-22.

[7] S. Klabnik, C. Nichols, *et al*, All the places patterns can be used, *The Rust Programming Language,* 2022. https://doc.rust-lang.org/book/ch18-01-all-the-places-for-patterns.html, retrieved 7-Jul-22.

[8] M. Brennan, Regular Expressions, *The GNU AWK User's Guide*, 2014. https://www.gnu.org/software/gawk/manual/html_node/Regexp.html, retrieved 7-Jul-22.

[9] T. Taft, R. Duff, et al, 5.4 Case Statements, *Ada Reference Manual*, 2002, http://www.ada-auth.org/standards/2xrm/html/RM-5-4.html, retrieved 8-Jul-22.

[10] M. Pettersson, A Term Pattern-Match Compiler Inspired by Finite Automata Theory. In *Workshop on Compiler Construction*. Springer-Verlag, Lecture Notes in Computer Science 641, 1992.

[11] M. O. Rabin and D. Scott, Finite automata and their decision problems. *IBM Journal of Research and Development.* **3** (2): 114–125, 1959. doi:10.1147/rd.32.0114. ISSN 0018-8646.

[12] J. E. Hopcroft, An *n* log*n* algorithm for minimizing the states in a finite automaton, In: Kohavi, Z. (ed.) *The Theory of Machines and Computations*, pp. 189–196. Academic Press, 1971.

[13] M. Fahndrich and J. Boyland, Staticallly Checkable Pattern Abstractions. *Proceedings of the International Conference on Functional Programming (ICFP '97)* | June 1997. https://www.researchgate.net/publication/236160052_Staticallly_Checkable_Pattern_Abstractions, retrieved 11-Jul-2022

[14] C. Okasaki, Red–black trees in a functional setting, *Journal of Functional Programming*. **9** (4): 471–477, 1992. doi:10.1017/S0956796899003494. ISSN 1469-7653.

Finding Locally Smallest Cut Sets using Max-SMT

Daniel Larraz, Cesare Tinelli
The University of Iowa, USA; email: {daniel-larraz, cesare-tinelli}@uiowa.edu

Abstract

Model-based development (MBD) is increasingly being used for system-level development of safety-critical systems. This approach allows safety engineers to leverage the system model created during the MBD process to assess the system's resilience to component failure. In particular, one fundamental activity is the identification of minimal cut sets (MCSs), i.e, minimal sets of faults that lead to the violation of a safety requirement. Although the construction of a formal system model enables safety engineers to automate the generation of MCSs, this is usually a computationally expensive task for complex enough systems. We present a method that leverages Max-SMT solvers to efficiently obtain a small set of faults based on a local optimization of the cut set cardinality. Initial experimental results show the effectiveness of the method in generating cut sets that are close or equal to globally optimal solutions (smallest cut sets) while providing an answer 5.6 times faster on average than the standard method to find a smallest cut set.

Keywords: Safety Analysis, Minimal Cut Set, SMT-based Model Checking, Max-SMT.

1 Introduction

Safety analysis is a crucial and well established activity in the design of critical systems that is often mandated by certification regulations. It aims at proving that a given system operates within some level of safety in the presence of faults. Traditionally, safety analysis has been performed manually based on informal design models, making the analysis highly subjective and dependent on the skill of the practitioner. However, in recent years there has been a growing interest in Model-based Safety Analysis (MBSA) [1]. This is an approach in which the design and safety engineers share a common system model created using a Model-based development (MBD) process. In MBD, the development is centered around a formal specification, or model, of the system. This model can then be subject to various kinds of rigorous analysis and synthesis such as completeness and consistency analysis, model checking, test case generation, etc. MBSA uses the system model to assess the system's resilience to component failure, and construct safety analysis artifacts such as minimal cut sets and fault trees.

In this context, a *minimal cut set (MCS)* is a minimal set of faults, a.k.a *basic events*, that lead to the violation of a safety requirement or some other failure, the so called *top level event (TLE)*. These sets of faults, or *fault configurations*, can be arranged in a *fault tree*, a tree making use of logical gates to depict the logical interrelationships linking such events with the TLE. Finding cut sets is important to assess the fault tolerance level of a system design, and investigate how failures propagate through the system.

In this paper, we focus on the safety analysis of behavioral models of infinite-state reactive systems. In this setting, safety requirements are (LTL) regular safety properties of the intended system model, which can always be recast as invariant properties. A system model consists of a *nominal* model, which specifies the behavior of the system in the absence of faults, and a set of faulty behaviors, which augment the nominal behavior whenever their corresponding faults are present. Thus, we consider the problem of (dis)proving safety properties in the presence of faults, and computing minimal cut sets, if any, for the violation of a safety property.

Typically, system models can be faithfully encoded as logical formulas. For such systems the problem above can be addressed with logic-based model checking techniques, such as k-induction [2] and IC3 [3], that capitalize on the power of solvers for satisfiability modulo theories (SMT). When these model checking techniques disprove a safety property under failure conditions, they also produce a counterexample demonstrating how faults lead to a failure. These counterexamples can be used to reason about the evolution of faults over time, and extract a fault configuration, although a non-necessarily minimal one.

We work under the *monotonicity assumption*, commonly adopted in safety analysis, that additional faults cannot prevent the violation of an already violated safety property. Under this assumption, a minimal cut set for a property is preferable to a super-set of it, since the latter will still cause the property to fail. Moreover, smaller MCSs are preferable over larger ones, since, in practical cases, the smaller a MCS the greater the probability that the safety property can be violated. This leads to the standard practice, in particular for complex systems, of computing only MCSs up to a maximum cardinality. However, that may still be computationally expensive and therefore its application may be pushed only to late phases of the development process. As a consequence, to help safety engineers with early detection of design issues it is fundamental to be able to resort to more efficient methods for computing small cut sets.

The key observation of this work is that *ensuring* minimality or minimal cardinality with respect to *all* fault configurations and counterexamples that lead to the violation of a safety

property is not always required for the early detection of problems in a system design. By definition, a safety property is one that fails to hold if and only if it is violated by a *finite counterexample*, i.e., a finite execution of the system. For this reason, it is often sufficient to compute a cut set with minimal cardinality over all counterexamples of a *given* length n. This often results in a small cut set that is close or equal to a globally optimal solution (a smallest MCS) and, as such, is enough to point to flaws in the system design.

To illustrate this point and the other concepts introduced so far, we will use a simple example of an aircraft controller derived from previous work [4]. The example is introduced in Section 3, but first, in the next section, we give a brief description of the notions and notations that will be used throughout the paper. The rest of the paper is organized as follows. Section 4 describes how to encode faulty behaviors into a nominal system. Section 5 presents the base method to compute a (not necessarily minimal) cut set using a faulty model. Section 6 describes how to compute a cut set with minimal cardinality. Section 7 presents the method we propose to obtain efficiently a small cut set based on a local optimization of the cut set cardinality. Experimental results comparing both approaches are reported in Section 8. Section 9 presents related work, and Section 10 concludes with a discussion of further research.

2 Preliminaries

2.1 SAT, Max-SAT, and Max-SMT

Let $\mathcal{P}$ be a finite set of *propositional variables*. If $p \in \mathcal{P}$ then p and $\neg p$ are *literals*. A *clause* is a disjunction of literals. A *propositional formula (in conjunctive normal form)* is a conjunction of clauses. The problem of *propositional satisfiability (or SAT)* consists in determining whether or not a given formula is *satisfiable*, or has a *model*: an assignment of truth values to its variables that makes it true.

A generalization of SAT is the *satisfiability modulo theories* (SMT) problem [5], which consists in deciding the satisfiability of a given (typically) quantifier-free first-order formula with respect to a background theory $\mathcal{T}$. In this setting, a model (which we may also refer to as a solution) is an assignment of values from the theory to the formula's variables that satisfies the formula and interprets function and predicate symbols consistently with the axioms of $\mathcal{T}$. Here we will consider the theories of *linear real/integer arithmetic (LRA/LIA)*, where literals are linear inequalities over real and integer variables, respectively.

Another generalization of SAT is the *Max-SAT* problem [5] which considers formulas in conjunctive normal form where each conjunct or *clause* is labeled as a *hard* or a *soft* constraints and each soft constraint is assigned a positive weight. The problem consists in finding an assignment of truth values for the formula's variable that satisfies all the hard constraints and maximizes the sum of the weights of the satisfied soft constraints, or dually, that minimizes the sum of the weights of the soft constraints it falsifies.

Max-SMT [6] is the natural extension of the Max-SAT problem to SMT where formulas can contain variables over additional data types other than the Booleans, and so the sought maximizing assignments are over such variables as well.

In general, if F is a formula and $\boldsymbol{x}$ is a tuple of variables, we write $F[\boldsymbol{x}]$ to indicate that the elements of $\boldsymbol{x}$ are free in F. If then $\boldsymbol{t}$ is a tuple of terms of the same type as $\boldsymbol{x}$, we denote by $F[\boldsymbol{t}]$ the formula obtained from F by simultaneously replacing every occurrence of a variable from $\boldsymbol{x}$ by the corresponding term in $\boldsymbol{t}$.

2.2 Transition Systems, Invariants, and LTL specifications

We represent a system model as a state transition systems $S = \langle \boldsymbol{s}, I[\boldsymbol{s}], T[\boldsymbol{s}, \boldsymbol{s}'] \rangle$ where $\boldsymbol{s}$ is a vector of typed state variables, I is the initial state predicate over the variables $\boldsymbol{s}$, and T is a two-state transition predicate over the variables $\boldsymbol{s}$ and $\boldsymbol{s}'$ where $\boldsymbol{s}'$ is a renamed version of $\boldsymbol{s}$ denoting the next state. We will use $\langle I, T \rangle$ to refer to transition system S when the vector of state variables $\boldsymbol{s}$ is clear from the context or not important. We will assume without loss of generality that T has the structure of a top-level conjunction, that is, $T[\boldsymbol{s}, \boldsymbol{s}'] = T_1[\boldsymbol{s}, \boldsymbol{s}'] \wedge \cdots \wedge T_n[\boldsymbol{s}, \boldsymbol{s}']$ for some $n \geq 1$. Notice that this is the norm in specification languages, like Lustre [7], where the modeled system is expressed as the synchronous product of several subcomponents, each of which is in turn formalized as the conjunction of one or more constraints. The conjunctive formulation is also common in languages that express the transition relation as a set of guarded transitions. By a slight abuse of notation, we will then identify T with the set $\{T_1, \ldots, T_n\}$ of its top-level conjuncts.

A *state property* $P[\boldsymbol{s}]$ for a system $S = \langle \boldsymbol{s}, I[\boldsymbol{s}], T[\boldsymbol{s}, \boldsymbol{s}'] \rangle$, expressed as a predicate over the variables $\boldsymbol{s}$, is *invariant for* S if it holds in every reachable state of S.

We use standard notions and notation from Linear Temporal Logic (LTL) to formalize temporal properties of transition systems.

2.3 Bounded Model Checking

Bounded Model Checking (BMC) [8] is a method involving checking potential executions of a system model in an incremental fashion against the negation of a state property by encoding them as propositional satisfiability formulas. Although BMC was originally developed for propositional encodings of finite-state systems, the technique has been successfully extended and applied to SMT encodings of (in)finite-state systems. Given a transition system $S = \langle \boldsymbol{s}, I[\boldsymbol{s}], T[\boldsymbol{s}, \boldsymbol{s}'] \rangle$, a state property P, and a bound k, BMC unrolls the system k times to produce a SMT formula φ_k such that φ_k is satisfiable iff P has a counterexample of length k or less:

$$\varphi_k = I[\boldsymbol{s}_0] \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} (T[\boldsymbol{s}_j, \boldsymbol{s}_{j+1}] \wedge \neg P[\boldsymbol{s}_i])$$

where $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_k$ are each a fresh renaming of $\boldsymbol{s}$. Formula φ_k is given to an SMT solver to be checked for satisfiability. If it is satisfiable, then the SMT solver will provide an assignment that satisfies φ_k. With this assignment, the counterexample is constructed using the values extracted from variables $\boldsymbol{s}_i$. If φ_k is unsatisfiable, that means no state is reachable in k steps or less such that the state violates property P.

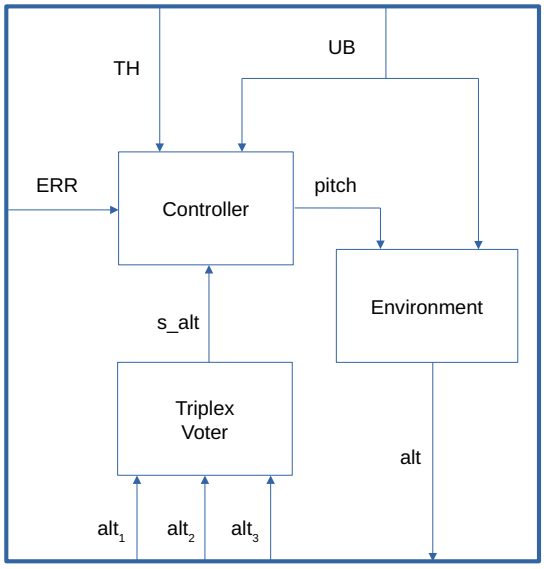


Figure 1: Diagram of the System Model

We will use BMC_Encoding(I, T, P, k) to denote a call to a function that returns the formula φ_k given as input a transition system $S = \langle I, T \rangle$, a state property P, and a bound k.

3 Motivating Example

Suppose we want to design a component for an airplane that controls the pitch motion of the aircraft, and suppose one of the system safety requirements is that the aircraft should not ascend beyond a certain altitude. The controller must read the current altitude of the aircraft from a sensor, and modify the next position of the aircraft's nose accordingly. Moreover, we want the system to be fault-tolerant to sensor failures. One way to improve system fault-tolerance is to introduce some redundancy. In particular, we can equip the system with three different altimeters so the controller receives three independent altitude values. Then the controller, with the help of a dedicated component, a *triplex voter*, takes the average of the two altitude values that are closest to each other — as they are more likely to be close to the actual altitude. Following a model-based design, we model an abstraction of the system's environment to which the aircraft's controller will react. We also model the fact that the system relies on a possibly imperfect reading of the current altitude by an altimeter sensor to decide the next pitch value. Finally, we provide a specification for the controller's behavior so that it satisfies the system requirement of interest.

A diagram of our model is shown in Figure 1. The main component, represented by the outermost rectangle, is an *observer* component that represents the full system consisting in this case of just three subcomponents, for simplicity: one component modeling the controller, one modeling a triplex voter, and another one modeling the environment. The observer has three inputs: alt_1, alt_2, and alt_3, representing the altitude values from each altimeter, and an output alt, representing the actual current altitude of the aircraft, which we are modeling as a product of the environment in response to the pitch value generated by the controller.

The system model makes a series of assumptions on the altitude values provided by the sensors and on a number of symbolic numeric constants (TH, UB and ERR) which act in effect as model parameters. Constant TH represents a threshold of the altitude value, constant UB models an upper bound on the change in altitude from one execution step to the next, and constant ERR is a bound on the sensor measurement error (more details below). The first assumption, $C1$, establishes that constants TH and UB are positive, and constant ERR is non-negative. The three next assumptions, $S1$, $S2$, and $S3$ account for the fact that, while the altitude value produced by each altimeter is not 100% accurate in actual settings, its error is bounded by a constant (ERR). That is, the system assumes the satisfaction of LTL formulas $S_i \equiv \Box(\mathsf{alt} - \bigcirc\mathsf{alt}_i \leq \mathsf{ERR})$ for $1 \leq i \leq 3$. Under those assumptions, the system must satisfy the LTL property $R1 \equiv \Box(\mathsf{alt} \leq \mathsf{TH})$, that formalizes the requirement that aircraft maintain its altitude below a certain threshold TH at all times.

Let $M = \min(|\mathsf{alt}_1 - \mathsf{alt}_2|, |\mathsf{alt}_1 - \mathsf{alt}_3|, |\mathsf{alt}_2 - \mathsf{alt}_3|)$. As explained above, triplex voter takes the sensor values and computes an estimated altitude for the controller satisfying the following specification:

$$\mathsf{s\_alt} = \begin{cases} (\mathsf{alt}_1 + \mathsf{alt}_2)/2, & \text{if } M = |\mathsf{alt}_1 - \mathsf{alt}_2| \\ (\mathsf{alt}_1 + \mathsf{alt}_3)/2, & \text{if } M = |\mathsf{alt}_1 - \mathsf{alt}_3| \\ (\mathsf{alt}_2 + \mathsf{alt}_3)/2, & \text{if } M = |\mathsf{alt}_2 - \mathsf{alt}_3| \end{cases}$$

We abstract the dynamics of the Controller and the Environment by omitting details that are not relevant for the satisfaction of the safety requirement $R1$. In the Controller's case, we model the guarantee that the controller will produce a negative pitch value whenever the sensor altitude indicates that the aircraft is getting *too close* to the threshold value TH, by which we mean that the difference between the current altitude and TH is smaller than UB + ERR:

$$L1 \equiv \Box(\mathsf{s\_alt} > \mathsf{LIMIT} \Rightarrow \mathsf{pitch} < 0)$$

with $\mathsf{LIMIT} = \mathsf{TH} - (\mathsf{UB} + \mathsf{ERR})$.

If alt represents the actual altitude of the aircraft, the Environment satisfies the following specification:

- $E1 \equiv \mathsf{alt} = 0$
- $E2 \equiv \Box(\mathsf{alt} \geq 0)$
- $E3 \equiv \Box(\bigcirc\mathsf{pitch} < 0 \Rightarrow \bigcirc\mathsf{alt} \leq \mathsf{alt})$
- $E4 \equiv \Box(\bigcirc\mathsf{pitch} < 0 \Rightarrow \bigcirc\mathsf{alt} \geq \mathsf{alt} - \mathsf{UB})$
- $E5 \equiv \Box(\bigcirc\mathsf{pitch} > 0 \Rightarrow \bigcirc\mathsf{alt} \geq \mathsf{alt})$
- $E6 \equiv \Box(\bigcirc\mathsf{pitch} > 0 \Rightarrow \bigcirc\mathsf{alt} \leq \mathsf{alt} + \mathsf{UB})$
- $E7 \equiv \Box(\bigcirc\mathsf{pitch} = 0 \Rightarrow \bigcirc\mathsf{alt} = \mathsf{alt})$

The specification captures salient constraints on the physics of our model by specifying that a positive pitch value (which has the effect of raising the nose of the aircraft and lowering its tail) makes the aircraft ascend, a negative value makes it descend, and a zero value keeps it at the same altitude. The specification also states that the actual altitude starts at

zero, is always non-negative, and does not change by more than a constant value (UB) in one sampling frame, where a sampling frame is identified with one execution step of the synchronous model (one global clock tick) for simplicity. The latter constraint on the altitude change rate captures physical limitations on the speed of the aircraft.

A model checker can easily prove that safety requirement $R1$ is satisfied by the system model. This provides evidence that the system satisfies the safety requirement in the absence of faults. However, this result is not enough to determine whether the introduced redundancy mechanism makes the system more fault tolerant. To check this, we can consider different faulty behaviors, that is, different ways of injecting faults into the sensors. For this example, we will consider a very general faulty model where any of the sensors can fail and provide a value that does not satisfy assumptions S_i at some step. This way, if one of the altimeters fails, in the sense that it produces an altitude reading with an error greater than the maximum expected error, the other two values should allow the system to compensate for that error. To confirm this, we can compute a smallest cut set and verify the cardinality of the cut set is two, i.e., at least two of the assumptions S_i must fail to hold to trigger the TLE. Perhaps surprisingly though, if we compute a smallest cut set using an algorithm like the one we will present in next section, we see that there exists a smallest cut set of cardinality one that consists of only one of the assumptions S_i. That is, a single sensor failure is enough to lead the system to the violation of safety requirement $R1$. Put differently, property $R1$ requires *all three sensors* to behave according to their specification despite the use of a triplex voter.

As we will see in next section, computing a smallest cut set usually requires finding a series of smaller cut sets and counterexamples associated with it *and* eventually proving there is no counterexample of *any length* with a smaller cut set than the last cut set found so far. However, to spot a flaw like the one described above sometimes it is enough to find a counterexample and a cut set without imposing restrictions on the cardinality of the cut set, and then, look for a counterexample of the same length that minimizes the number of cut set elements. Unlike first approach, which performs *global optimization*, the second approach considerably narrows down the search space by considering only counterexamples of a fixed length, determined by the first counterexample found. When applied to our example, this *local optimization* approach can find the *same* cut set of cardinality one as the first approach but it can compute it much more efficiently, as we will see in next section.

After reviewing the model in light of the existence of a cut set of size one, however computed, a designer may conclude that to benefit from the triplex voter it is necessary to decrease the safety limit value LIMIT in the controller's contract. In particular, it is enough to decrease it by doubling the error bound value: $\mathsf{LIMIT} = \mathsf{TH} - (\mathsf{UB} + 2 * \mathsf{ERR})$. After this change, both approaches to compute cut sets return one of cardinality two, consisting of two of the assumptions S_i. In this case though, only the global optimization approach provides the guarantee that no smaller cut set exists.

As shown for our example, however, the local approach we propose, because of its lower computational cost, enables users to check for and discover design issues early in the modeling process. This difference in performance can be increasingly significant as the scale of analyzed systems grows. We present later initial experimental evidence suggesting that the advantages of our approach extend beyond the example given here.

4 Encoding Faulty Behavior

Faulty behavior in a system is specified as an extension of its nominal model. Starting from a nominal model $S = \langle \boldsymbol{s}, I[\boldsymbol{s}], T[\boldsymbol{s}, \boldsymbol{s'}]\rangle$ with $T = \{T_1, \ldots, T_n\}$, the system designer identifies m disjoint non-empty subsets $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_m$ of T corresponding to m possible faults the system can suffer from so that every model component T_j in $\boldsymbol{F}_i$ is affected when fault i occurs. To simplify the exposition, we assume here that each fault affects different parts of the system. Also for simplicity, we assume that faults do not effect the behavior of the system in its initial state(s).[1]

Now, for all faults $i = 1, \ldots, m$, the designer provides an alternative, faulty behavior specification $\widehat{T}_j$ of every model component T_j in $\boldsymbol{F}_i$. This faulty behavior is enabled by a Boolean flag f_i that represents the occurrence of fault i. Let $T^{\diamond}$ collect the components of T affected by none of the modeled faults, that is, $T^{\diamond} = T \setminus (\boldsymbol{F}_1 \cup \cdots \cup \boldsymbol{F}_m)$. Then, the faulty model is given by transition system $S^{\star} = \langle \boldsymbol{z}, I[\boldsymbol{z}], T^{\star}[\boldsymbol{z}, \boldsymbol{z'}]\rangle$, where the vector of typed variables $\boldsymbol{z}$ extends $\boldsymbol{s}$ with the fresh Boolean constants $f_1, \ldots, f_m$ and the transition predicate is defined by

$$T^{\star}[\boldsymbol{z}, \boldsymbol{z'}] \quad = \quad T^{\diamond} \cup \{\, T_j \vee (f_i \wedge \widehat{T}_j) \mid 1 \leq i \leq m,\ T_j \in \boldsymbol{F}_i \,\}$$

For convenience and generality, the system $S^{\star}$ is defined so that the presence of a fault i in a system execution (corresponding to the flag f_i having value true) may or may not trigger the faulty behavior in the affected components at any particular step of the execution. Note that, in effect, we can obtain the nominal model S from $S^{\star}$ by conjoining the constraints $\neg f_1, \ldots, \neg f_m$ to the initial state predicate.

5 Computing a cut set and a counterexample

Consider again a transition system $S = \langle \boldsymbol{s}, I[\boldsymbol{s}], T[\boldsymbol{s}, \boldsymbol{s'}]\rangle$ with faulty behavior specifications $\boldsymbol{F}_1, \ldots, \boldsymbol{F}_m \subseteq T$ enabled by faults $\boldsymbol{f} = \langle f_1, \ldots, f_m\rangle$, respectively. Given a state property P expected to be invariant for S, we can look for a cut set and a counterexample that leads the system to the violation of P by checking whether P is also invariant for the extended system $S^{\star}$ defined as in the previous section. If we can disprove P, the constant values assigned to $\boldsymbol{f}$ in any trace that leads $S^{\star}$ to the violation of P determines a cut set, namely, the set of all f_i's that are true. Such cut set describes a fault configuration that jeopardizes the invariance of P. On the other hand, if we prove P invariant, we can conclude that the system is robust to all faults as far as P is concerned.

[1] An extension to the case in which a component's behavior may be affected initially and possibly by more than one fault can be done with a slightly more complex formalization.

Algorithm 1
ComputeGlobalCutSet($\langle \boldsymbol{s}, I, T\rangle$, $\langle f_i, \boldsymbol{F}_i\rangle_{1\leq i\leq m}$, P)

```
1:  T^◇ := T \ (F_1 ∪ ··· ∪ F_m)
2:  T^○ := { T_j ∨ (f_i ∧ T̂_j) | 1 ≤ i ≤ m, T_j ∈ F_i }
3:  T* := T^○ ∪ T^◇
4:  t := m; res := unknown; θ := ∅; f := ⟨f_1, ..., f_m⟩
5:  do
6:      I* := I ∧ AtMostK(f, t)
7:      res, θ' := Verify(s, I*, T*, P)
8:      if res = unsafe then
9:          t := t − 1; θ := θ'        ▷ Store last counterexample
10:     end if
11: while t ≥ 0 ∧ res = unsafe
12: if t < m then                      ▷ Unsafe with t + 1 faults
13:     if t < 0 then
14:         return ⟨∅, θ, true⟩        ▷ Nominal system itself unsafe
15:     else
16:         C := ExtractCutSet(θ, f)
17:         is_a_smallest_sol := (res = safe)
18:         return ⟨C, θ, is_a_smallest_sol⟩
19:     end if
20: else
21:     if res = unknown then
22:         return Unknown
23:     else
24:         return NoSolution
25:     end if
26: end if
```

Algorithm 2
ComputeLocalCutSet($\langle \boldsymbol{s}, I, T\rangle$, $\langle f_i, \boldsymbol{F}_i\rangle_{1\leq i\leq m}$, P)

```
1:  Let T* be as in Algorithm 1
2:  res, θ := Verify(s, I, T*, P)
3:  if res = unknown then
4:      return Unknown
5:  else
6:      if res = safe then
7:          return NoSolution
8:      else
9:          k := length(θ);
10:         φ_k := BMC_Encoding(I, T*, P, k)
11:         SmtAssertHard(φ_k)
12:         for i := 1 to m do
13:             SmtAssertSoft(¬f_i, 1)
14:         end for
15:         SmtCheckSat()
16:         θ' := GetCounterexample()
17:         C := ExtractCutSet(θ', f)
18:         return ⟨C, θ'⟩
19:     end if
20: end if
```

In this work, we assume we have access to a black-box procedure Verify to perform these invariance checks. From a theoretical standpoint, Verify is an oracle since the invariance problem is undecidable in the infinite-state case. In practice, however, SMT-based model checking techniques, such as k-induction and IC3, yields incomplete versions of Verify that often provide a sound answer in reasonable time. In our concrete implementation, we make the verification check terminating by imposing a time limit and extending the type of the returned result with an additional value (`unknown`) to account for the timeout being reached.

6 Computing a globally smallest cut set

Building on top of the basic ideas described in the previous section, we can compute a smallest cut set and an associated counterexample, or determine that no such cut set exists, using Algorithm 1. The procedure can return Unknown due to the undecidability of the underlying model checking problem. It can also return a cut set that does not necessarily have minimal cardinality, which is indicated by a flag.

The key idea of the algorithm is to add to the initial state predicate I a cardinality constraint AtMostK$(\boldsymbol{f}, k)$ over the fault flags $\boldsymbol{f}$ that restricts possible solutions to fault configurations with at most k (present) faults. Lines 5-11 use this reduction to find an MCS of minimal cardinality. If none exists, or the first call to Verify returned `unknown`, the condition in line 12 is false, and the algorithm returns the corresponding result in each case in lines 21-25. Specifically, if the first call to Verify reaches the time limit without determining the invariance of property P under the faulty conditions, the algorithm returns Unknown at line 22. If the first call to Verify returns `safe`, i.e., proves that P is invariant under faulty conditions, the algorithm returns there is no cut set solution (at line 24).

If the nominal system S itself does not satisfy P, condition in line 14 is true, and the unique MCS is the empty set. Otherwise, a cut set is extracted at line 16 from θ, which is the last error trace found in the do-while loop. This is done simply by collecting all the flags f_i that are assigned value true by the error trace. Before returning the cut set and the counterexample in line 18, the procedure determines whether the cut set has minimal cardinality by checking if the last call to Verify returned `safe` (line 17).

7 Computing a locally smallest cut set

Algorithm 1 from Section 6 ensures that the cut set returned at line 18 has minimal cardinality provided that none of the calls to Verify return `unknown`. But it does that at the cost of having to solve multiple model checking problems, and having to prove the input property invariant for the provided system in the last call to Verify, which is often a harder problem to solve than disproving the satisfaction of a property.

In this section we describe an alternative method which does not ensure global optimality but can nonetheless find a small cut set with cardinality close or equal to a globally optimal solution while needing to call Verify only once.

In addition to Verify, the procedure implementing this method and described in Algorithm 2, relies on an external, off-the-self Max-SMT solver. It starts by checking whether cut sets exist at all by solving the model checking problem described in Section 5. If the call to Verify reaches the time limit without determining the invariance of property P under the faulty conditions, the algorithm returns Unknown at line 4. If the

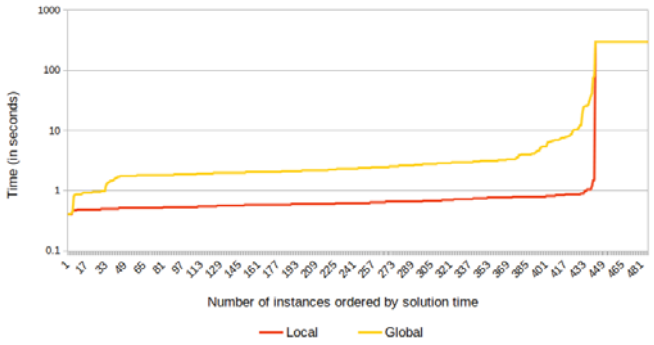


Figure 2: Comparison between local and global optimization

call to Verify returns `safe`, i.e., proves that P is invariant under faulty conditions, the algorithm returns there is no cut set solution (at line 7). Otherwise, we know there exists a cut set. With k being the length of the counterexample θ returned by Verify, the procedure builds a Max-SMT problem to find the smallest cut set among all cut sets with an associated counterexample of length exactly k (at lines 9-18).[2]

The main idea is to first create a standard Bounded Model Checking encoding (as described in Section 2.3) with the faulty model and invariance property P for a bound k equal to the length of θ. The resulting formula is asserted as a hard constraint to the Max-SMT solver (at line 11). The optimization problem for the Max-SMT solver consists in minimizing the number of active faults needed to violate property P. This is done by asserting to the solver one soft constraint per fault flag f_i, all with weight 1, stating that the flag is false ($\neg f_i$ at lines 12-14). Note that Max-SMT problem so obtained, checked at line 15, is guaranteed to have a solution since the counterexample to P invariance found by the call to Verify is a solution to this problem as well. By the optimality of the solution θ' returned by the Max-SMT solver (at 16), the cut set C computed at line 17 is guaranteed to have smallest cardinality among all cut sets associated to counterexamples of length k, the length of the original counterexample. The procedure returns both the cut set and its associated counterexample, similarly to the global optimization one but without any claims about the cut set's global optimality.

8 Experimental Evaluation

The main premise of this work is that the approach for computing small cut sets based on local optimization, as described in Section 7, is more efficient in practice than the computation providing global guarantees as described in Section 6. Moreover, the solutions computed by the local optimization approach have a cardinality equal or close to the optimal one.

To test our hypothesis, we implemented both approaches in our model checker KIND 2 [9], and compared their performance on a fairly large set of benchmarks. Because of the practical difficulty of finding publicly available models readable by KIND 2 and specifically designed with fault tolerance in mind, we retooled a previous set of safety benchmarks to the purposes of our evaluation. Each benchmark in the starting set, largely based on one introduced by Kahsai and Tinelli [10][3] consists of a multi-component model with a single invariance property. We selected benchmarks from the starting set by running KIND 2 on a cluster with ten Intel(R) Xeon(R) E3-1240, 3.4GHz, 4 cores, 16 GB memory machines. We kept the benchmarks for which KIND 2 was able to prove the property valid with a 5 minute timeout, which yielded 486 instances. Then, we considered the problem of proving each property under the possibility of each model component failing to provide outputs consistent with its low-level specification. More specifically, we modeled the faulty behavior simply by allowing the failing component to return any value at all among those allowed by its output type.[4]

We ran KIND 2 on the selected problems using each approach with 15 minute timeout. Figure 2 shows that the local optimization approach significantly out-performs the global optimization one, providing an answer 5.6 times faster on average. Analyzing the individual responses to each problem, we found that both approaches were able to detect the absence of cut sets for 5 of the problems, and the local approach was able to find a cut set with optimal cardinality in 434 out of the 436 cases where both approaches found a cut set. In the two cases where the local approach returned a suboptimal solution, it was off just by one with respect to the size of the optimal solution.

These results should be taken with a grain a salt due to the synthetic nature of the benchmarks, a majority of which had highly fault-intolerant systems. In the majority of cases, the globally optimal cut sets had cardinality one and in the rest it had cardinality two. More seriously perhaps, in most of the benchmarks almost any one of the considered faults would lead, by itself, to the violation of the property, making it easy for our local optimization method to stumble into a globally optimal solution. Although more experiments are needed, we find that our evaluation provides nevertheless encouraging preliminary evidence of the usefulness of the local optimization method.

9 Related Work

The use of Bounded Model Checking to find a cut set and an associated counterexample was first proposed by Abdulla et al. [11]. The focus of that work, however, is not on computing a single cut set but *all* MCSs. To prevent the generation of non-minimal solutions, the paper proposes the computation of cuts sets of increasing cardinality. This approach has also the advantage of generating smaller MCSs before larger MCSs. Thus, one can generate a single smallest MCS just stopping after the first optimal solution is generated. The search for a single smallest MCS is very similar to the one performed by the method presented in Section 6, the main difference being the direction of the search. The technique of Abdulla et al. progressively tries all numbers of faults from 0 to m

[2] Alternatively, we could consider cut sets with associated counterexample of length *at most* k, like in the standard formulation of BMC. However, typical implementations of Verify based on k-induction and IC3 usually provide counterexamples whose length is already minimal or close to it most of the time.

[3] Available at `https://github.com/kind2-mc/kind2-benchmarks`.

[4] That is, we did not consider the possibility of an ill-typed result.

(forward enumeration), whereas our algorithm tries values from m to 0 (backward enumeration). The rationale of our strategy is to minimize the number of solved problems for which the property is valid, which is often a harder problem to solve than disproving the satisfaction of a property, especially when the length of the generated counterexamples is not very long. As we describe in other work [4, 9], KIND 2 implements the approach above using backward enumeration to compute all MCSs, (and a single globally smallest MCS). Unlike the encoding of faulty models presented in this paper, the actual implementation in KIND 2 only supports faulty behavioral specifications equivalent to true. More specifically, KIND 2 allows the user to choose a set of model elements (assumptions and guarantees, node calls, equations in node bodies, ...) of its input language, an extension of the dataflow Lustre language [7], and KIND 2 will compute minimal sets of those elements whose violation leads the system to an unsafe state. However, this is not really a limitation since the more general case presented in this work can reduced to the more specific case supported by KIND 2: once a faulty model is built following the encoding described in Section 4, the user just have to specify assumptions stating that f_i should be false, and choose those assumptions as the model elements.

Another algorithm for computing all MCSs is described by Bozzano et al. [12]. It too forces the algorithm to proceed by layers of increasing cardinality. Thus, it may also be used to compute a globally smallest cut set. The method relies on a IC3-based routine for parameter synthesis to compute all the solutions in each layer. Therefore, instead of relying on a black-box Verify procedure to solve multiple ordinary model checking queries, they use a specialized algorithm. The main advantage in that case is that the information learnt to block a particular counterexample can be reused when considering new ones.

A different approach to computing all MCSs is the method presented by Stewart et al. [13]. It exploits the duality between the set of *Minimal Inductive Validity Cores (MIVCs)* [14], minimal sets of model elements that are sufficient to prove a property, and the set of Minimal Cut Sets for the same property and computes the latter from the former. This is convenient when the goal is to compute *all* MCSs since one can use an offline algorithm for enumerating all MIVCs [15], which may offer better overall performance than computing one MIVC at a time. The downside of this solution is that, unlike the techniques described earlier, it cannot be used to compute a single MCS for a property without paying the cost of computing all of them.

10 Conclusion

We presented a method that leverages behavioral modeling and Max-SMT solvers to obtain efficiently a small set of faults that lead to the violation of safety requirements. The method computes a cut set with minimal cardinality over all counterexamples of a *given* length by reducing the problem into an optimization problem over an SMT formula. Initial experimental results are very encouraging in terms of the effectiveness of the method in generating cut sets that are close or equal to globally optimal solutions, and the speed up achieved compared to the standard method for computing a (globally) smallest cut set.

As future work, we want to investigate further the effectiveness of the proposed technique by applying the method to a broader set of benchmarks, and evaluating its performance in determining not only the tolerance of a system against faults, but also its resilience to cyber-attacks. We also want to explore the possibilities of setting different weights for the soft constraints in the Max-SMT problem, which leads to a natural way of establishing preference between difference solutions beyond the cardinality of the cut sets. Specifically, in Algorithm 2, the soft constraints stating that a particular fault should not occur all have weight 1. One can imagine assigning a higher weight to a subset of the soft constraints to give preference to solutions that do not include faults in the subset.

References

[1] A. Joshi, S. Miller, M. Whalen, and M. Heimdahl, "A proposal for model-based safety analysis," in *24th Digital Avionics Systems Conference*, vol. 2, pp. 13 pp. Vol. 2–, 2005.

[2] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," in *Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings* (W. A. H. Jr. and S. D. Johnson, eds.), vol. 1954 of *Lecture Notes in Computer Science*, pp. 108–125, Springer, 2000.

[3] A. R. Bradley, "Sat-based model checking without unrolling," in *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA, January 23-25, 2011. Proceedings* (R. Jhala and D. A. Schmidt, eds.), vol. 6538 of *Lecture Notes in Computer Science*, pp. 70–87, Springer, 2011.

[4] D. Larraz, M. Laurent, and C. Tinelli, "Merit and blame assignment with kind 2," in *Formal Methods for Industrial Critical Systems - 26th International Conference, FMICS 2021, Paris, France, August 24-26, 2021, Proceedings* (A. Lluch-Lafuente and A. Mavridou, eds.), vol. 12863 of *Lecture Notes in Computer Science*, pp. 212–220, Springer, 2021.

[5] A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds., *Handbook of Satisfiability - Second Edition*, vol. 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

[6] R. Nieuwenhuis and A. Oliveras, "On SAT modulo theories and optimization problems," in *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings* (A. Biere and C. P. Gomes, eds.), vol. 4121 of *Lecture Notes in Computer Science*, pp. 156–169, Springer, 2006.

[7] N. Halbwachs, F. Lagnier, and C. Ratel, "Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE," *IEEE Trans. Software Eng.*, vol. 18, no. 9, pp. 785–793, 1992.

[8] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings* (R. Cleaveland, ed.), vol. 1579 of *Lecture Notes in Computer Science*, pp. 193–207, Springer, 1999.

[9] D. Larraz, M. Laurent, and C. Tinelli, "Merit and blame assignment with kind 2," *CoRR*, vol. abs/2105.06575, 2021.

[10] T. Kahsai and C. Tinelli, "Pkind: A parallel k-induction based model checker," in *Proceedings 10th Int'l Workshop on Parallel and Distributed Methods in verifiCation, PDMC 2011*, vol. 72 of *EPTCS*, pp. 55–62, 2011.

[11] P. A. Abdulla, J. Deneux, G. Stålmarck, H. Ågren, and O. Åkerlund, "Designing safe, reliable systems using scade," in *Leveraging Applications of Formal Methods, First International Symposium, ISoLA 2004, Paphos, Cyprus, October 30 - November 2, 2004, Revised Selected Papers* (T. Margaria and B. Steffen, eds.), vol. 4313 of *Lecture Notes in Computer Science*, pp. 115–129, Springer, 2004.

[12] M. Bozzano, A. Cimatti, A. Griggio, and C. Mattarei, "Efficient anytime techniques for model-based safety analysis," in *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I* (D. Kroening and C. S. Pasareanu, eds.), vol. 9206 of *Lecture Notes in Computer Science*, pp. 603–621, Springer, 2015.

[13] D. Stewart, M. W. Whalen, M. P. E. Heimdahl, J. Liu, and D. D. Cofer, "Composition of fault forests," in *Computer Safety, Reliability, and Security - 40th International Conference, SAFECOMP 2021, York, UK, September 8-10, 2021, Proceedings* (I. Habli, M. Sujan, and F. Bitsch, eds.), vol. 12852 of *Lecture Notes in Computer Science*, pp. 258–275, Springer, 2021.

[14] E. Ghassabani, A. Gacek, and M. W. Whalen, "Efficient generation of inductive validity cores for safety properties," in *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016* (T. Zimmermann, J. Cleland-Huang, and Z. Su, eds.), pp. 314–325, ACM, 2016.

[15] E. Ghassabani, M. W. Whalen, and A. Gacek, "Efficient generation of all minimal inductive validity cores," in *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017* (D. Stewart and G. Weissenbacher, eds.), pp. 31–38, IEEE, 2017.

Basic Formal Verification of a Waypoint Manager for Unmanned Air Vehicles in SPARK

Laura Humphrey
Air Force Research Laboratory, Control Science Center of Excellence, Ohio, USA; email: laura.humphrey@us.af.mil

Abstract

As software becomes more complex, it becomes more difficult to verify its correctness. This poses a particular problem for autonomous systems, since they are software-intensive and will also require strong evidence of correctness in order to be allowed to operate in the real world. One way to help address this problem is through the use of formal methods, i.e. mathematically-based tools for software and hardware verification. In this paper, we perform formal program verification on a service in OpenUxAS, a free and open source software framework for mission-level, multi-vehicle autonomy. More specifically, we apply the SPARK language and verification toolset to a service that sanity-checks and segments long sequences of vehicle waypoints to prove that it is free of runtime errors.

Keywords: Formal methods, program verification, unmanned air vehicles, SPARK.

1 Introduction

As software becomes more complex, it becomes more difficult to verify its correctness. In particular, concerns have been raised about how to better verify increasingly autonomous systems [1]. In order for autonomous systems to operate in the real world, there will need to be strong evidence that they meet their requirements, especially those related to safety. However, given the huge number of behaviors they can exhibit, the proportion of behaviors that can feasibly be covered by testing is relatively low [2]. A possible solution for certain aspects of this problem lies in the use of formal methods, i.e. mathematically-based tools and approaches for software and hardware design and verification. Toward this end, we explore the application of the SPARK language and formal verification toolset to part of OpenUxAS, a software framework for mission-level, multi-vehicle autonomy. In particular, we verify that the "Waypoint Plan Manager" service, which sanity-checks and segments long sequences of waypoints so that they can be safely sent to an autopilot, is free of runtime errors. Section 2 provides background on SPARK and OpenUxAS. Section 3 gives details on the Waypoint Plan Manager, including its role in the system, how it and other services written in a mix of Ada and SPARK are structured, the implementation of its core logic in SPARK, and SPARK verification results. Section 4 concludes the paper.

2 Background

2.1 SPARK

SPARK is both a programming language and associated toolset for formal program verification [3]. As a language, SPARK is based on Ada [4]. Ada is a statically-typed, imperative, object-oriented language with a strong type system whose design philosophy is intended to encourage reliability, maintainability, and efficiency. These characteristics are inherited by SPARK, which both removes some features from Ada that make verification difficult and adds a small number of features that facilitate verification. Several good resources are available for learning SPARK and Ada [3, 4, 5].

In terms of program verification, SPARK performs two forms of sound static analysis on source code. The first is flow analysis, which checks for initialization of variables and compliance with user-specified data flow contracts. The second is proof, which checks for both functional correctness, i.e. compliance with user-specified behavioral contracts, and the absence of runtime errors. Some colloquial terms are used to describe the different levels to which one can use SPARK to verify a program [6], namely "stone," "bronze," "silver," "gold," and "platinum." Stone level is achieved when code is accepted by SPARK, since the SPARK language has stricter legality rules than Ada. Bronze is achieved when flow analysis returns with no error. Silver is achieved when proof shows that the code cannot produce runtime errors. Gold is achieved when proof shows that the code satisfies user-defined contracts that partially describe the desired behavior of the code. Platinum is achieved when proof shows that the code satisfies user-defined contracts that completely describe the desired behavior of the code.

At a high level, SPARK performs verification by translating SPARK programs, including checks and contracts to be verified, to the Why3 platform for deductive program verification [7]. Why3 then uses a weakest-precondition calculus to generate verification conditions (VCs), i.e. logical formulas whose validity would imply soundness of the code with respect to its checks and contracts. Why3 then uses multiple provers, including satisfiability modulo theory (SMT) solvers CVC4, Alt-Ergo, and Z3, to attempt to prove the validity of the VCs. Though SPARK aims to fully automate verification through this process, it is often necessary for the user to provide additional annotations in the code, e.g. assertions, loop invariants, and type invariants, to create additional VCs that help guide the provers toward a proof of the original checks and contracts to be verified.

2.2 OpenUxAS

OpenUxAS is a publicly released version of the Unmanned Systems Autonomy Services (UxAS) [8] that is available free and open-source on GitHub [9]. It is a software framework for mission-level autonomy for teams of unmanned vehicles, with the "x" indicating support for different types of vehicles, including air, ground, and sea surface vehicles. It has been used as an academic platform for building new autonomy capabilities [10, 11], performing studies on human-automation interaction [12, 13], and as a case study for new verification approaches [14, 15, 16, 17]. While OpenUxAS has support for different types of vehicles, its primary emphasis is on unmanned air vehicles (UAVs) carrying out intelligence, surveillance, and reconnaissance (ISR) missions. This paper will therefore focus on its application to UAVs.

Missions in OpenUxAS consist of sequences of tasks[1], i.e. different types of vehicle behaviors, with optional constraints on the order in which tasks can be assigned and which vehicles they can be assigned to. For example, OpenUxAS provides services for planning and executing point, line, and area searches. Each type of task has optional parameters that constrain how the task can be performed, and each instantiation of a task can have different parameter values. Parameters vary by task type but generally include things like standoff distance from an entity of interest, the angle at which to approach the entity, search patterns to be used, and desired ground sample distance (i.e. resolution of obtained imagery measured in meters per pixel, which is affected by altitude, sensor gimbal angle, and sensor zoom level). [18] provides detailed descriptions of some common types of tasks and their parameters.

Note that OpenUxAS assumes each vehicle has an autopilot that can follow waypoints and a gimbal that can steer the sensor. A plan for an individual task is then essentially a sequence of waypoint and sensor steering commands (though some tasks generate commands "on-the-fly" during execution). A plan for an individual vehicle's mission is then a sequence of task plans, with waypoint-based paths that honor operating region constraints connecting the end and start of subsequent tasks. A mission plan for a team of vehicles is then a set of mission plans, one per vehicle, that jointly accomplish a collection of tasks while honoring possible constraints on task ordering and vehicle assignment, usually while optimizing some performance metric such as vehicle distance traveled. For path planning, OpenUxAS computes kinematically feasible waypoint-based paths using path planners that generally model vehicles as Dubins vehicles, i.e. vehicles whose maximum turn rate depends on minimum turn radius and current speed.

As an example, Figure 1 shows a plan for an instantiation of a UAV point search task. The UAV is depicted as a triangle and its sensor footprint as projected onto the ground as a trapezoid. The UAV starts at position and heading $\bar{\nu}_e$. Based on the UAV's altitude and its sensor zoom level and gimbal angle, the distances to the trailing and leading edges of its sensor

[1]With the meaning of "task" as commonly used in robotics, not the Ada "Task" keyword.

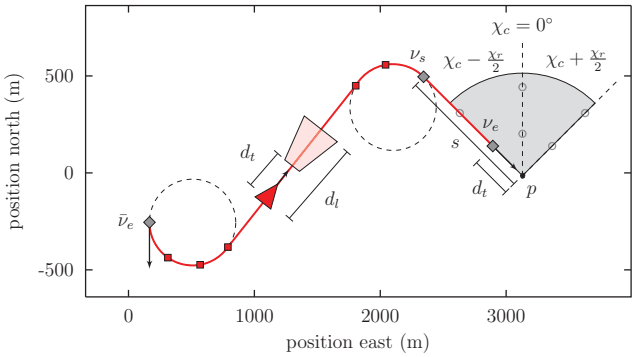


Figure 1: A point search task in OpenUxAS. From [18].

footprint are d_t and d_l, respectively. The point to be surveilled is p. For optional task parameters, a range of acceptable approach angles $\chi_c \pm \frac{\chi_r}{2}$ (measured clockwise from the north) and a standoff distance s have been specified. Given these parameter values, a kinematically feasible path is planned that has the vehicle approach p from the position and heading indicated by ν_s. (If no standoff distance had been specified, d_l would have been used instead.) The task ends when the trailing edge of the sensor footprint has passed point p, i.e. when the vehicle is distance d_t from the point. Planning in OpenUxAS generally attempts to minimize vehicle distance traveled, so approach angle $\chi_c - \frac{\chi_r}{2}$ is chosen in this case.

OpenUxAS can work in either a centralized mode in which a single instance controls multiple vehicles, e.g. through a multi-vehicle ground control station, or in a decentralized mode in which each vehicle runs its own instance of OpenUxAS. Decentralized mode is more robust to communications loss. When vehicles are able to communicate and share information about their state, decentralized planning algorithms optimize the assignment of tasks comprising a mission across eligible vehicles; when vehicles cannot share state information, tasks are simply duplicated across eligible vehicles. Note that a few specialized tasks are designed to handle intermittent or infrequent communication directly, e.g. [16].

2.3 OpenUxAS Implementation

OpenUxAS as currently written in C++ is implemented as a service-oriented architecture, with services running in separate threads. ZeroMQ [19] provides the communication transport layer between services, with most services connecting to each other using a publish/subscribe pattern. A TCP/IP bridge service allows OpenUxAS to communicate with other systems and applications, e.g. simulators and ground control stations. Zyre [20], an open-source framework for proximity-based peer-to-peer applications built on ZeroMQ, is used to send and receive messages between vehicles in decentralized mode. Instances of OpenUxAS and services within OpenUxAS communicate by sending and receiving messages whose types and fields are defined by Message Data Model (MDMs) expressed in XML. These MDMs conform to the standard of the Lightweight Message Communication Protocol (LMCP), which specifies rules for message data structure and serialization. In LMCP, message fields have types that

include enumerations, primitives (byte, bool, int32, uint32, int64, real32, real64), other messages, and arrays of any of these. Messages can also inherit from other messages in an object-oriented manner. The tool LmcpGen [21] generates libraries for representing, manipulating, serializing, and deserializing messages defined in a set of MDMs in several target languages, including C/C++, Python, Java, C#, and Ada. The main MDM or message set used by OpenUxAS is called the Common Mission Automation Services Interface (CMASI) [22], which allows it to interact with the OpenAMASE simulation environment [23, 24]. CMASI includes messages for vehicle configuration and state information, operating region constraints, task configuration, requests for mission planning, mission plans as sequences of waypoints and sensor steering commands, etc. Internally, OpenUxAS uses some additional message sets to coordinate between services.

OpenUxAS is designed to be extensible and configurable. The main executable takes as a command line option the name of an XML configuration file that specifies which services should be loaded and how they should be configured. It uses a "Service Manager" to start up each service specified in the file and pass it its configuration parameters. This makes it relatively easy to add new services. That being said, in its standard configuration, there are a core set of services that work together to plan and execute multi-vehicle missions. OpenUxAS receives external messages about vehicle configurations, current vehicle states, tasks to perform, and operating region constraints (keep-in and keep-out zones). It also receives *AutomationRequest* messages from external sources or sometimes internal services that result in mission plans being generated. These requests express which tasks should be performed, with optional constraints on the order in which they can be assigned and which vehicles are eligible to perform them. An "Automation Request Validator" (ARV) service checks whether a request is valid, i.e. the request only references entities and tasks that have been previously injected into the system through the appropriate messages. It also ensures that only one request is processed by the system at a time, queueing up new requests if planning for a current request is still in progress. The ARV passes valid *AutomationRequest* messages to the rest of the system as *UniqueAutomationRequest* messages. For each task listed in the request, an "Assignment Tree Branch and Bound" (ATBB) service sends requests for a set of task options, i.e. a set of valid plans for each eligible vehicle with estimated costs in terms of the distance the vehicle would have to travel from the start to the end of the task plan. To compute these, services for each type of task send batch path planning requests to a "Route Aggregator" (RA) service, which in turn makes requests to one or more path planning routines. Once the ATBB has received options for all tasks in a *UniqueAutomationRequest*, it computes the paths and corresponding costs required to travel between task options, and given constraints on task assignment ordering specified in the request, it uses a branch and bound algorithm to optimize the assignment of tasks to eligible vehicles. The result is ultimately an *AutomationResponse* that includes an array of *MissionCommand* messages, one per vehicle. Each *MissionCommand* contains an array of *Waypoint* messages containing both waypoint and associated sensor steering commands. This array of *Waypoint* messages includes all the commands required for the vehicle to complete its assigned sequence of tasks, i.e. its mission. Each vehicle has a "Waypoint Plan Manager" service that processes *MissionCommand* messages and sends potentially modified *MissionCommand* messages that have been sanity checked and broken down into segments short enough for an autopilot, as discussed in the next section.

OpenUxAS was originally written in C++. However, since ZeroMQ and LmcpGen support Ada, individual OpenUxAS services can be re-implemented in a mix of Ada and SPARK and run with other services still written in C++. To date, a few services have been re-implemened in Ada, and their core logic has been re-implemented and formally verified to different levels in SPARK. [17] describes the first iteration of a SPARK/Ada implementation of the ARV verified to the gold level, which has since been updated on the GitHub repository. The repository also has SPARK/Ada implementations of the RA verified to the gold level, the ATBB verified to the silver level (with ongoing efforts to verify it to the gold level), and the WPM verified to the silver level (as described in this paper).

3 The OpenUxAS Waypoint Plan Manager

As mentioned in the previous section, OpenUxAS includes a "Waypoint Plan Manager" (WPM) service that processes waypoint and associated sensor steering commands from an array of *Waypoint* messages stored within a *MissionCommand* so that they can be sent to an autopilot. There are two reasons that this processing is needed. The first is that most autopilots have a limit on the number of waypoints they can accept at a time. Whereas a *MissionCommand* might contain a sequence of hundreds or even thousands of waypoints, an autopilot might only accept a few dozen at a time. The second is that some sanity checking and safety constraints should also be applied. The WPM performs both of these functions.

Let us briefly summarize the behavior of the WPM. It receives *MissionCommand* messages. Field *MissionCommand.WaypointList* is an array of *Waypoint* messages that each include 1) the waypoint's ID number and 2) the ID number of the next waypoint to visit (among other information such as waypoint location and altitude, associated sensor steering commands, and travel speed). Conceptually, this array of *Waypoint* messages encodes a linked list of waypoints. Field *MissionCommand.FirstWaypoint* is the ID of the first waypoint that the vehicle should visit. Part of the WPM's job is to decompose the list of waypoints from a *MissionCommand* into smaller lists of waypoints that are short enough to send to an autopilot. We refer to these smaller lists of waypoints as "segments," and they are also encoded as *MissionCommand* messages. To ensure a smooth vehicle trajectory, segments should overlap by some service-configurable number of waypoints, with a minimum overlap of two. As a safety feature, the last waypoint in the segment should refer to itself as the next waypoint to visit. If OpenUxAS crashes or experiences degraded performance, this causes the vehicle to loiter at the last point in the segment. If OpenUxAS operates normally, segments are updated when the vehicle reaches the point of

overlap, and the waypoints in the original *MissionCommand* are followed as expected.

Unfortunately, the construction of segments is not quite as straightforward as it seems. First, the waypoints in array *MissionCommand.WaypointList* are not guaranteed to be ordered according to ID and next ID. Second, while each waypoint ID is supposed to be unique, there is no easy mechanism for enforcing this, so there may be waypoints with duplicate IDs. Third, there is no guarantee that the list contains a waypoint whose ID matches *MissionCommand.FirstWaypoint*. Fourth, there can be cycles in the list. In such a case, the WPM should command the vehicle to repeat the route specified by the cycle until the WPM receives a new *MissionCommand*. Fifth, segments should not just include the first waypoint and all subsequent waypoints, they should also include the first waypoint's predecessor. This is because a waypoint preceding the first helps set the path the vehicle should follow. Given two waypoints, the autopilot attempts to minimize crosstrack error of the vehicle along the path between them; given only the one, the autopilot attempts to fly directly to it.

Given that the WPM is the interface between OpenUxAS and a vehicle's autopilot, it should be verified to a high level of assurance. Currently, the C++ version has only been lightly tested, and it does not address all of the previous concerns. For instance, it assumes that waypoints are ordered according to ID and next ID, that waypoint IDs are unique, and that there are no cycles in the list (a separate flag is used to request a cycle over the whole list, neglecting the possibility of a cycle within the list itself). This motivates the re-implementation and formal verification of portions of the WPM in SPARK.

3.1 The Structure of OpenUxAS Services in Ada and SPARK

Past efforts have developed a basic framework for implementing OpenUxAS services in SPARK/Ada [17]. Since OpenUxAS is designed to be able to coordinate with other instances of OpenUxAS for decentralized mission planning, this framework is essentially a partial re-implementation of OpenUxAS in SPARK/Ada that is able to work with the remaining portions of the C++ version by exchanging LMCP messages over ZeroMQ.

Large portions of the SPARK/Ada version of OpenUxAS are written in Ada. These portions handle things like service creation and initialization, sending and receiving messages over ZeroMQ, and extraction of data from LMCP messages, since the Ada libraries generated by LmcpGen include types that are not SPARK-compatible, mainly *Ada.Containers.Vectors* and *access* types that do not necessarily follow an ownership policy. Certain portions of the system that do complex manipulation of data, i.e. logic processing that embodies key OpenUxAS behaviors, are implemented in SPARK so that formal verification can be performed.

The SPARK/Ada version is structured as follows. The main executable is analogous to the C++ "Service Manager." Given the name of an XML configuration file as a command line argument, it starts up the services specified in the file and passes to each one its corresponding configuration information. Each service inherits from an abstract tagged type *Service_Base* defined in Ada package *UxAS.Comms.LMCP_Net_Client.Service*, which declares functions and procedures for initializing and configuring services. By convention, each service is structured into three packages:

1. *<ServiceName>_Communication* – Provides basic subprograms to process and send LMCP messages in SPARK-compatible formats. Uses package *LMCP_Messages*, which defines SPARK-compatible types for each LMCP message using *Functional_Vectors* in place of *Vectors* and without *access* types. Uses package *LMCP_Message_Conversions*, which provides subprograms to convert between Ada and SPARK LMCP message types. Defines a type *<ServiceName>_Mailbox* that stores ZeroMQ configuration information for the service. Uses types and subprograms in package *UxAS.Comms.LMCP_Object_Message_Sender_Pipes* to define subprograms to send SPARK LMCP message types.

2. *UxAS.Comms.LMCP_Net_Client.Service.<Name>* – Defines a private tagged type *<ServiceName>_Service* that implements the abstract tagged type *Service_Base*. Overrides associated functions and procedures as appropriate. Uses packages *<ServiceName>_Communication* and *<ServiceName>*. Private fields of *<ServiceName>_Service* include *Mailbox : <ServiceName>_Mailbox*, *Config : <ServiceName>_Configuration_Data*, and *State : <ServiceName>_State*. The type definition for field *Mailbox* is from package *<ServiceName>_Communication* and the type definitions for fields *State* and *Config* are from package *<ServiceName>*. Fields *Config* and *State* hold SPARK-compatible data that are processed by portions of the service written in SPARK. *Config* is a record that stores service data that tend to be state invariant, while *State* is a record that stores service data that are manipulated by the service during processing. Defines a procedure *Process_Received_LMCP_Message* that performs service-specific handling of LMCP messages.

3. *<ServiceName>* – Contains SPARK subprograms and associated data structures used by the previous package. These tend to be called within the procedure *Process_Received_LMCP_Message* to update the *State* when key messages are received. Occasionally they are called when other events occur, e.g. an internal timer triggers. Encompasses core behavior/logic of the service.

3.2 Implementation of the Waypoint Plan Manager in Ada and SPARK

Let us now consider the WPM service specifically. There is one instance of the WPM per vehicle. Its parameters, whose values are set by the configuration file passed to OpenUxAS, include:

- *VehicleID* – The ID of this WPM's vehicle.
- *NumberWaypointsOverlap* – The number of overlapping waypoints between segments (≥ 2).

- *NumberWaypointsToServe* – The maximum number of waypoints in a segment (> *NumberWaypointsOverlap*).

Each WPM subscribes to *MissionCommand*, *AutomationResponse*, and *AirVehicleState* messages. Following the convention of other services, the WPM's implementation of procedure *Process_Received_LMCP_Message* calls other subprograms to handle specific types of messages. For an *AirVehicleState* message, it calls a simple Ada procedure that checks whether it corresponds to the WPM's vehicle, and if so, it updates the WPM's *State* to record whether the vehicle is headed for the first waypoint of the next segment. It also uses the *AirVehicleState* timestamp to update an internal timer used to enforce a minimum time between the broadcast of subsequent segments. If the WPM receives a *MissionCommand*, it calls a simple Ada procedure that checks whether it corresponds to the WPM's vehicle, and if it receives an *AutomationResponse*, it calls a simple Ada procedure that checks whether it contains a *MissionCommand* corresponding to the WPM's vehicle. If so, both procedures call a more complex SPARK procedure *Handle_MissionCommand* with a SPARK-compatible *MissionCommand* message that updates the WPM's *State*. Finally, *Process_Received_LMCP_Message* checks whether a new segment needs to be sent as a *MissionCommand* based on the internal timer and the WPM's *State*, in particular whether a new *MissionCommand* has been received or the vehicle is now headed toward the ID of the first waypoint of the next segment. If so, it calls a SPARK procedure *Produce_Segment*, which computes the segment, sends the *MissionCommand*, and updates the WPM's *State*.

Listing 1: *Waypoint_Plan_Manager* package specification.

```
with ...

package Waypoint_Plan_Manager with SPARK_Mode is

   Max : constant Ada.Containers.Count_Type := 2000;

   subtype Pos64 is Int64 range 1 .. Int64'Last;
   subtype Nat64 is Int64 range 0 .. Int64'Last;

   function Pos64_Hash (X : Pos64) return Ada.Containers.
      Hash_Type is (Ada.Containers.Hash_Type'Mod (X));

   package Pos64_WP_Maps is new Ada.Containers.
      Formal_Hashed_Maps (Pos64, Waypoint, Pos64_Hash);
   type Pos64_WP_Map is new Pos64_WP_Maps.
      Map (Max, Pos64_WP_Maps.Default_Modulus (Max));

   package Pos64_Nat64_Maps is new Ada.Containers.
      Formal_Hashed_Maps (Pos64, Nat64, Pos64_Hash);
   type Pos64_Nat64_Map is new Pos64_Nat64_Maps.
      Map (Max, Pos64_Nat64_Maps.Default_Modulus (Max));

   package Pos64_Vectors is new Ada.Containers.
      Formal_Vectors ( Positive , Pos64);
   type Pos64_Vector is new Pos64_Vectors.Vector (Max + 1);

   type Waypoint_Plan_Manager_Configuration_Data is record
      NumberWaypointOverlap : UInt32 := 2;
      NumberWaypointsToServe : Common.UInt32 :=
         Common.UInt32 (Max);
      ...
   end record;

   type Waypoint_Plan_Manager_State is record
      MC : MissionCommand;
      Id_To_Waypoint : Pos64_WP_Map;
      Id_To_Next_Id : Pos64_Nat64_Map;
      New_Command : Boolean;
      Next_Segment_Id : Nat64 := 0;
      Next_First_Id : Nat64 := 0;
      Prefix : Pos64_Vector;
      Cycle : Pos64_Vector;
      Segment : Pos64_Vector;
      Headed_To_First_Id : Boolean := False;
   end record;

   procedure Handle_MissionCommand
    (State : in out Waypoint_Plan_Manager_State;
     MC : MissionCommand)
   with Pre =>
      Length (MC.WaypointList) <= Max and then
      MC.FirstWaypoint > 0;

   procedure Produce_Segment
    (State : in out Waypoint_Plan_Manager_State;
     Config : Waypoint_Plan_Manager_Configuration_Data;
     Mailbox : in out Waypoint_Plan_Manager_Mailbox)
   with Pre =>
      State.Next_First_Id > 0
      and then State.Next_Segment_Id > 0
      and then Config.NumberWaypointsOverlap >= 2
      and then Config.NumberWaypointsOverlap <=
               UInt32 (Max) - 1
      and then Config.NumberWaypointsToServe >
               Config.NumberWaypointsOverlap
      and then Config.NumberWaypointsToServe <=
               UInt32 (Max);

end Waypoint_Plan_Manager;
```

The SPARK package *Waypoint_Plan_Manager* is shown in Listing 1. Constant *Max* defines the capacity of containers used to store waypoint information based on the maximum anticipated length of *MissionCommand.WaypointList*. Subtypes *Pos64* and *Nat64* define positive and natural ranges of 64-bit integers, since waypoint ID numbers should be positive, with "0" only used as a next ID to indicate there is no next waypoint. *Formal_Hashed_Maps* and *Formal_Ordered_Vectors* are used to define containers for holding waypoint information. Type *Waypoint_Plan_Manager_Configuration_Data* holds WPM configuration data. Type *Waypoint_Plan_Manager_State* includes the following fields:

- *MC* – The most recent SPARK-compatible *MissionCommand*.
- *Id_To_Waypoint* – A map from waypoint IDs to corresponding SPARK-compatible *Waypoint* messages taken from *MC*.
- *Id_To_Next_Id* – A map from waypoint IDs to corresponding next waypoint IDs taken from *MC*.
- *New_Command* – A Boolean indicating whether a new *MissionCommand* was just received.
- *Next_Segment_Id* – A 64-bit natural indicating the first waypoint ID of the next segment (0 indicates no more segments).
- *Next_First_Id* – A 64-bit natural indicating the *First-Waypoint* ID of the next segment (0 indicates no more segments).

- *Prefix* – An ordered list of waypoint IDs from *MC.WaypointList* based on *MC.FirstWaypoint* that do not contain a cycle but potentially precede one.
- *Cycle* – An ordered list of waypoint IDs from *MC.WaypointList* based on *MC.FirstWaypoint* that form a cycle after a potential prefix.
- *Segment* – An ordered list of waypoint IDs that form the current segment.
- *Headed_To_First_ID* – A Boolean indicating whether the vehicle is currently flying to the waypoint with ID *Next_First_Id*.

The procedure *Handle_MissionCommand*, whose body is shown in Listing 2, has as its parameters the WPM's *State* and *MC*, a SPARK-compatible *MissionCommand*. It updates *State* based on the contents of *MC*, using helper procedure *Extract_MissionCommand_Maps* (not shown) to compute maps *State.Id_To_Waypoint* and *State.Id_To_Next_Id*. This helper procedure simply iterates over all elements of *MC.WaypointList* from first index to last index, ignoring elements with duplicate or 0-valued waypoint IDs. The procedure *Handle_MissionCommand* then attempts to build a linked-list representation of ordered waypoint IDs in local vector *Id_List*, starting from *First_Id := MC.FirstWaypoint*. It starts with a duplicate map *Ids := State.Id_To_Next_Id*. It then searches this map for *First_Id* and its successor, then its predecessor, then further successors, removing elements from *Ids* as it goes. When no more successors can be found or a cycle is detected, it computes *State.Prefix* and *State.Cycle* and returns. Various return points are annoted with comments in Listing 2.

Listing 2: *Handle_MissionCommand* body.

```
procedure Handle_MissionCommand
 (State : in out Waypoint_Plan_Manager_State;
  MC : MissionCommand)
is
   First_Id : Pos64 := MC.FirstWaypoint;
   Id_List : Pos64_Vector;
   Ids : Pos64_Nat64_Map;
   function Successor (M : Pos64_Nat64_Map; K : Pos64)
     return Nat64 renames Element;
begin

   State.MC := MC;
   Extract_MissionCommand_Maps (State, MC);
   State.New_Command := True;
   State.Next_Segment_Id := First_Id;
   State.Next_First_Id := First_Id;
   Clear (State.Prefix);
   Clear (State.Cycle);
   Ids := State.Id_To_Next_Id;

   -- Look for First_Id.
   if Contains (Ids, First_Id) then
     if Successor (Ids, First_Id) = 0
       or else not
         Contains (Ids, Pos64 (Successor (Ids, First_Id)))
       or else Successor (Ids, First_Id) = First_Id
     then
       -- First_Id has no successors. Return.
       Append (State.Prefix, First_Id);
       return;
     else
       -- First_Id has a successor. Continue.
       Append (Id_List, First_Id);
       Append (Id_List, Successor (Ids, First_Id));
       Delete (Ids, First_Id);
     end if;
   else
     -- First_Id not found. Return with no segment.
     State.Next_Segment_Id := 0;
     State.Next_First_Id := 0;
     return;
   end if;

   pragma Assert (not Is_Empty (Id_List));
   pragma Assert (Length (Id_List) = 2);

   -- Look for a predecessor to First_Id.
   -- We already checked that it does not point to itself.
   for Id of Ids loop
     if Successor (Ids, Id) = First_Element (Id_List) then
       if Last_Element (Id_List) = Id then
         -- First_Id cycles with its precedessor. Return.
         State.Next_Segment_Id := Id;
         Reverse_Elements (Id_List);
         State.Cycle := Id_List;
         return;
       else
         -- First_Id has a precessor. Continue.
         State.Next_Segment_Id := Id;
         Prepend (Id_List, Id);
         Delete (Ids, Id);
         exit;
       end if;
     end if;
     pragma Loop_Invariant (not Is_Empty (Id_List));
     pragma Loop_Invariant (Length (Id_List) <= 3);
     pragma Loop_Invariant (Length (Ids) <= Max - 1);
   end loop;

   -- Search for successors until done.
   while Length (Ids) > 0 loop
     if Contains (Ids, Last_Element (Id_List)) then
       if Successor (Ids, Last_Element (Id_List)) = 0
         or else not Contains (State.Id_To_Next_Id,
           Pos64(Successor (Ids, Last_Element (Id_List))))
         or else Successor (Ids, Last_Element (Id_List)) =
                 Last_Element (Id_List)
       then
         -- Candidate successor is 0, unknown, or points
         -- to itself. Return with a prefix only.
         State.Prefix := Id_List;
         return;
       elsif Contains (Id_List,
               Successor (Ids, Last_Element (Id_List)))
       then
         -- Found a cycle. Compute prefix & cycle. Return.
         declare
           Index : Pos64_Vectors.Extended_Index;
           Next_Id : Pos64 :=
             Successor (Ids, Last_Element (Id_List));
         begin
           Index := Find_Index (Id_List, Next_Id);
           for I in First_Index (Id_List) .. Index-1 loop
             Append (State.Prefix, Element (Id_List, I));
             pragma Loop_Invariant
               (Integer (Length (State.Prefix)) =
                I - First_Index (Id_List) + 1);
           end loop;
           for I in Index .. Last_Index (Id_List) loop
             Append (State.Cycle, Element (Id_List, I));
             pragma Loop_Invariant
               (Integer (Length (State.Cycle)) =
                I - Index + 1);
           end loop;
           return;
```

```
            end;
          else
            -- Found a successor that's not a cycle.
            declare
              Id : Pos64 := Last_Element (Id_List);
            begin
              Append (Id_List, Successor (Ids, Id));
              Delete (Ids, Id);
            end;
          end if;
        else
          -- Can't find a successor. Return a prefix.
          State.Prefix := Id_List;
          return;
        end if;

        pragma Loop_Invariant (not Is_Empty (Id_List));
        pragma Loop_Invariant
          (Length (Id_List) <= Max - Length (Ids) + 1);
      end loop;

      -- No more successors. Return a prefix.
      State.Prefix := Id_List;

    end Handle_MissionCommand;
```

The last SPARK subprogram used by the WPM is *Produce_Segment*. It generates the next segment based on the current WPM *State* by updating *State.Segment* and sending it as a *MissionCommand*, modifying the last waypoint to point to itself. It also updates *State.New_Command*, *State.Next_Segment_Id*, and *State.Next_First_Id* to prepare for the next segment. The body for this procedure is shown in Listing 3.

Listing 3: *Produce_Segment* body.

```
    procedure Produce_Segment
     (State : in out Waypoint_Plan_Manager_State;
      Config : Waypoint_Plan_Manager_Configuration_Data;
      Mailbox : in out Waypoint_Plan_Manager_Mailbox)
    is
      Id : constant Pos64 := State.Next_Segment_Id;
      First_Id : constant Pos64 := State.Next_First_Id;
      Prefix : constant Pos64_Vector := State.Prefix;
      Cycle : constant Pos64_Vector := State.Cycle;
      Len : constant Positive :=
        Positive (Config.NumberWaypointsToServe);
      Overlap : constant Positive :=
        Positive (Config.NumberWaypointsOverlap);

      I : Natural := 1;
      C : Pos64_Vectors.Extended_Index;
      In_Prefix : Boolean;
    begin

      State.New_Command := False;
      State.Next_Segment_Id := 0;
      State.Next_First_Id := 0;
      Clear (State.Segment);

      C := Find_Index (Prefix, Id);
      In_Prefix :=
       (if C /= Pos64_Vectors.No_Index then True else False);

      while C in First_Index (Prefix) .. Last_Index (Prefix)
        and then I <= Len
      loop
        pragma Loop_Invariant
                (Natural (Length (State.Segment)) < I);
        Append (State.Segment, Element (Prefix, C));
        C := Iter_Next (Prefix, C);
        I := I + 1;
      end loop;

      C:= (if In_Prefix then First_Index (Cycle) else
                             Find_Index (Cycle, Id));

      while C in First_Index (Cycle) .. Last_Index (Cycle)
        and then I <= Len
      loop
        pragma Loop_Invariant
                (Natural (Length (State.Segment)) < I);
        Append (State.Segment, Element (Cycle, C));
        C := Iter_Next (Cycle, C);
        if not Iter_Has_Element (Cycle, C) then
          C := First_Index (Cycle);
        end if;
        I := I + 1;
      end loop;

      if Integer (Length (State.Segment)) > Overlap then
        State.Next_Segment_Id :=
          Element (State.Segment,
                   Last_Index (State.Segment) - Overlap + 1);
        State.Next_First_Id :=
          Element (State.Segment,
                   Last_Index (State.Segment) - Overlap + 2);
      end if;

      declare
        MC_Out : MissionCommand := State.MC;
        WP_List : WP_Seq;
        Id : Pos64;
        WP : Waypoint;
      begin
        MC_Out.FirstWaypoint := First_Id;
        for I in First_Index (State.Segment) ..
                 Last_Index (State.Segment)
        loop
          Id := Element (State.Segment, I);
          if Contains (State.Id_To_Waypoint, Id) then
            WP := Element (State.Id_To_Waypoint, Id);
            if I = Last_Index (State.Segment) then
              WP.NextWaypoint := WP.Number;
              ...
            end if;
            WP_List := Add (WP_List, WP);
          end if;
          pragma Loop_Invariant
            (Integer (Length (WP_List)) <=
             I - First_Index (State.Segment) + 1);
        end loop;
        MC_Out.WaypointList := WP_List;
        sendBroadcastMessage (Mailbox, MC_Out);
      end;

    end Produce_Segment;
```

3.3 Verification of the Waypoint Plan Manager in SPARK

Verification of the WPM in SPARK at the silver level, i.e. absence of runtime errors, is relatively straightforward. Preconditions are needed for *Handle_MissionCommand* to ensure that the length of *MC.WaypointList* is less than or equal to *Max* (i.e. the capacity of containers used in the WPM's *State*) and that *MC.FirstWaypoint* is positive. Preconditions are needed for *Produce_Segment* to ensure that the next segment's commanded first waypoint ID and starting ID are positive, that the number of waypoints to overlap per segment is at least 2, and that the maximum number of waypoints to serve is greater than the overlap and at most *Max*.

A few assertions and loop invariants are needed to establish that the sizes of all containers stay within their maximum bounds. Loop invariants are needed in loops that copy content from one container to another, including the body of *Extract_MissionCommand_Maps* (not shown). Assertions are needed in *Handle_MissionCommand* after the logic block that searches for *First_Id* to assert that *Id_List* has exactly 2 elements at that point.

It was generally easy to determine what assertions and loop invariants were needed, though it was somewhat challenging to figure out which were needed to bring the maximum size of type *Pos64_Vector* down to its actual maximum of *Max + 1*. In *Handle_Mission_Command*, *Id_List* is of this type, and the number of elements appended to it depends on the size of map *Ids*, so the two loop invariants relating the length of *Ids* to *Max* and *Id_List* were needed for this.

There is at least one improvement that could be made. At the end of *Produce_Segment*, where the list of waypoint ID numbers in *State.Segment* is used to retrieve *Waypoint* messages from *State.Id_To_Waypoint*, we must check that the corresponding waypoint ID exists in the keys of *State.Id_To_Waypoint*. This should not be necessary since all the IDs in *State.Segment* should be keys in *State.Id_To_Waypoint*, but the provers are not able to deduce this without additional assertions. If we attempted to prove gold-level properties of this service, such assertions would likely be naturally added, and this check could be removed.

Listing 4 shows relevant results from SPARK analysis. The code proves from a clean state in less than a minute on a standard laptop at level 2 (fast, all provers).

To give an idea of size, gnatmetric reports the body and spec of *Waypoint_Plan_Manager* have 248 code lines, 25 of which are preconditions, assertions, and loop invariants needed for SPARK verification. The whole service, which additionally includes packages *UxAS.Comms.LMCP_Net_Client.Service.Waypoint_Plan_Management* and *Waypoint_Plan_Manager_Communication*, has 629 code lines. The whole OpenUxAS project in SPARK/Ada has 12332 code lines.

Listing 4: SPARK analysis results.

```
-----------------------------------------
SPARK Analysis results   Total  Flow  Provers
-----------------------------------------
Data Dependencies            3     3  .
Flow Dependencies            .     .  .
  Initialization            10    10  .
Non-Aliasing                 .     .  .
Run-time Checks             89     .  89 (CVC4 94%,
                                       Trivial 6%)
Assertions                  26     .  26 (CVC4)
Functional Contracts        66     .  66 (CVC4 96%,
                                       Trivial 4%)
LSP Verification             .     .  .
Termination                  .     .  .
Concurrency                  .     .  .
-----------------------------------------
Total                  194  13 (7%)  181 (93%)
```

4 Conclusions

This paper describes implementation and formal verification of a service in the OpenUxAS mission-level multi-vehicle autonomy software framework in SPARK. Included is a framework for implementing services in SPARK/Ada that can interact with services in C++ over ZeroMQ using the LMCP message standard, a pattern that could be used in other multi-language message passing architectures. The WPM service presented in this paper is responsible for sanity checking lists of waypoint commands and segmenting them into shorter lists that can be sent to an autopilot. Currently, it is formally verified to be free of runtime errors. In the future, we would like to add contracts to relevant subprograms in the form of pre- and postconditions and verify them to the gold level. This will likely require adding a significant number of annotations to the code to guide the underlying provers toward a proof of the desired postconditions, far more than were needed in this effort to prove the absence of runtime errors.

Acknowledgments

Thanks to Pat Rogers, Joffrey Huguet, M. Anthony Aiello, and Claire Dross at AdaCore, who developed the core SPARK/Ada framework used to enable this work, and to Ben Hocking and Jonathan Rowanhill at Dependable Computing, who shared some code snippets for a simpler version of the WPM. Thanks to AFOSR, who funded this work under contract #20RQCOR096.

DISTRIBUTION STATEMENT A: Approved for public release. Distribution is unlimited. Case #AFRL-2022-2722.

References

[1] W. Dahm, "Report on technology horizons: A vision for Air Force science & technology during 2010–2030," Tech. Rep. AF/ST-TR-10-01-PR, United States Air Force, 2010.

[2] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?," *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[3] J. W. McCormick and P. C. Chapin, *Building high integrity applications with SPARK*. Cambridge University Press, 2015.

[4] J. Barnes, *Programming in Ada 2012*. Cambridge University Press, 2014.

[5] AdaCore, "LEARN.ADACORE.COM," 2022. https://learn.adacore.com.

[6] Y. Moy, "Climbing the software assurance ladder – practical formal verification for reliable software," *Electronic Communications of the EASST*, vol. 76, 2019.

[7] J.-C. Filliâtre and A. Paskevich, "Why3 – where programs meet provers," in *European Symposium on Programming (ESOP)*, pp. 125–128, Springer, 2013.

[8] S. Rasmussen, D. Kingston, and L. Humphrey, "A brief introduction to Unmanned Systems Autonomy Services (UxAS)," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 257–268, IEEE, 2018.

[9] AFRL/RQQ, "OpenUxAS," 2017. https://github.com/afrl-rq/OpenUxAS.

[10] L. V. Nguyen, B. Hoxha, T. T. Johnson, and G. Fainekos, "Mission planning for multiple vehicles with temporal specifications using UxAS," *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 67–72, 2018.

[11] T. Elliott, M. Alshiekh, L. R. Humphrey, L. Pike, and U. Topcu, "Salty – a domain specific language for GR (1) specifications and designs," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4545–4551, IEEE, 2019.

[12] J. Wei, M. L. Bolton, and L. Humphrey, "The level of measurement of trust in automation," *Theoretical Issues in Ergonomics Science*, vol. 22, no. 3, pp. 274–295, 2020.

[13] M. L. Bolton, E. Biltekoff, and L. Humphrey, "The level of measurement of subjective situation awareness and its dimensions in the situation awareness rating technique (SART)," *IEEE Transactions on Human-Machine Systems*, 2021.

[14] C. E. Tuncali, B. Hoxha, G. Ding, G. Fainekos, and S. Sankaranarayanan, "Experience report: Application of falsification methods on the UxAS system," in *NASA Formal Methods Symposium*, pp. 452–459, Springer, 2018.

[15] J. A. Davis, L. R. Humphrey, and D. B. Kingston, "When human intuition fails: Using formal methods to find an error in the 'proof' of a multi-agent protocol," in *International Conference on Computer Aided Verification (CAV)*, pp. 366–375, Springer, 2019.

[16] D. Greve, J. Davis, and L. Humphrey, "A mechanized proof of bounded convergence time for the Distributed Perimeter Surveillance System (DPSS) Algorithm A," in *17th ACL2 Workshop* (R. Sumners and C. Chau, eds.), Electronic Proceedings in Theoretical Computer Science (EPTCS) 359, pp. 33–47, 2022.

[17] M. A. Aiello, C. Dross, P. Rogers, L. Humphrey, and J. Hamil, "Practical application of SPARK to OpenUxAS," in *International Symposium on Formal Methods*, pp. 751–761, Springer, 2019.

[18] D. Kingston, S. Rasmussen, and L. Humphrey, "Automated UAV tasks for search and surveillance," in *International Conference on Control Applications (CCA)*, IEEE, 2016.

[19] P. Hintjens, *ZeroMQ*. O'Reilly Media Inc., 2013.

[20] iMatix Corporation, "Zyre," 2015. https://github.com/zeromq/zyre.

[21] AFRL/RQQ, "LmcpGen," 2017. https://github.com/afrl-rq/LmcpGen.

[22] M. Duquette, "The common mission automation services interface," in *Infotech@Aerospace*, p. 1542, AIAA, 2011.

[23] Z. Basnight, S. Rasmussen, A. Starr, M. Duquette, and K. Kalyanam, "Simulating cooperative control algorithms using MATLAB, Simulink, and AMASE," in *AIAA Modeling and Simulation Technologies Conference*, p. 8359, AIAA, 2012.

[24] AFRL/RQQ, "OpenAMASE," 2017. https://github.com/afrl-rq/OpenAMASE.

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. Julio Medina
Facultad de Ciencias
Universidad de Cantabria
Avda. de los Castros s/n
39005 Santander
Spain
Phone: +34-942-201477
Email: julio.medina@unican.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch

Ada-Europe Sponsors

27 Rue Rasson
B-1030 Brussels
Belgium
Contact:Ludovic Brenta
ludovic@ludovic-brenta.org

46 Rue d'Amsterdam
F-75009 Paris
France
Contact: Jamie Ayre
sales@adacore.com
www.adacore.com

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

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

Corso Sempione 68
20154 Milano
Italy
Contact: Massimo Bombino
massimo.bombino@vector.com
www.vector.com

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch


Ada
europe