The journal for the international Ada community

Ada User Journal

Volume 44
Number 3
September 2023

Produced by Ada-Europe

Information on Subscriptions and Advertisements

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

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

ADA USER JOURNAL

Volume 44
Number 3
September 2023

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

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

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

As we get closer to the end of 2023, I would like to refer again to the merger of the Ada User Journal with the Ada Letters, which will likely happen in 2024. Given that the two publications have been sharing content for the last couple of years, one of the main goals of the merger is to avoid this duplication. However, instead of keeping one of them and discarding the other, we plan to create a new publication, bringing together the expertise from both and meant to reach a wider audience. Specific details, like the new publication's name, formatting and design issues, and distribution methods, are still to be settled. I hope to bring you more information on this topic in the December issue.

Concerning the contents of this September issue, we conclude the publication of the articles from the AEiC 2023 Work-in-Progress session, and we include one article, the single one that was submitted to the AUJ for publication, from the AEiC 2023 Industrial Track. Seven AEiC 2023 WiP articles are included. We start with one authored by G. Jäger et al. from the University of Freiberg, in Germany, which describes the work being done to address the problem of remotely rescuing autonomous robots when safety-critical requirements are involved. Then, three articles related to AI follow. The first one, by B. Badjie et al., from the University of Lisbon, Portugal, looks at the problem of attacks on deep neural network models, describing an approach based on data denoising and reconstruction, combined with a defensive distillation methodology to improve the robustness against adversarial attacks. Then, D. Brown and G. Hawe, from Ulster University in the UK, present their work on the explainability of ML-based decisions, which is very important when dealing with safety-critical systems. In concrete, the article looks at several possible metrics to evaluate explainer models, and their trade-off. The third article in this AI-related set is authored by D. C. Schmidt et al. from Vanderbilt University, USA. The article delves into the discipline of prompt engineering, discussing how to code prompt patterns for correctly using Large Language Models (LLMs), namely for the development of more reliable software systems. We then provide an article authored by E. Sisinni et al. from the University of Brescia in Italy, which proposes an architecture for exploiting container-based systems to better support the requirements of IoT smart mobility applications. The next article, authored by T. Carvalho et al. from ISEP in Portugal, describes ongoing work to add real-time parallel programming extensions to the Rust language, which are not yet included in the existing parallel programming libraries. The last WiP article is authored by B. Djika Mezatio and G. Kouamou, from the University of Yaounde, in Cameroon, and by F. Singhoff and A. Plantec, from the University of Brest, France. It presents MONANO, a POSIX user-level library that allows monitoring real-time applications for detecting scheduling anomalies. The final article included in this issue is derived from a presentation at the AEiC 2023 Industrial Track. It is entitled “Safety of the Intended Functionality Concept Integration into a Validation Tool Suite” and presents work resulting from a collaboration between Austrian industry (Virtual Vehicle Research GmbH and AVL List GmbH) and academia (University of Graz). In concrete, the paper describes the experience in the integration of the SOTIF concept within the AVL SCENIUSTM tool suite, which covers the entire scenario-based validation process.

Last but not the least, the issue includes the News Digest section and the Calendar and Events section, respectively prepared by Alejandro R. Mosteo and Dirk Craeynest, their editors.

Antonio Casimiro
Lisboa
September 2023
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Alejandro R. Mosteo

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

Contents

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

Preface by the News Editor

Dear Reader,

Thanks to the efforts of a few Ada enthusiasts, it is now possible to peruse on-line the complete High Order Language Phase 1 reports for the original four candidate languages, of which Ada would eventually emerge. During this phase, language proposals addressed the IRONMAN requirements without providing a prototype implementation. Find the link on [1]; from my initial cursory read, there is plenty of good stuff in there!

If you are an Ada and macOS user and have still not been affected by the problem reported on [2], you probably want to be aware of it, although, hopefully, Apple will have fixed it by the time you read this digest.

Finally, on the practical side, I want to highlight the emerging consensus that when using `Text_IO` to read files, you should rely on the End_Error exception rather than on the End_Of_File subprogram. This is a rather unconventional conclusion, in which the exception becomes the ordinary; find the rationale in [3].

Sincerely,
Alejandro R. Mosteo.

[1] "Common HOL Phase 1 Reports", in Ada-related Resources.

[2] "MacOS: Best Not Upgrade to Xcode/CLT 15.0", in Ada and Operating Systems.

[3] "Get Character and Trailing New Lines", in Ada Practice.

Ada-related Events

October Ada Monthly Meetup 2023

From: Fernando Oleo Blanco <irvise_ml@irvise.xyz>
Subject: Re: Ada Monthly Meetup 2023
Date: Wed, 13 Sep 2023 21:30:10 +0200
Newsgroups: comp.lang.ada

I would like to announce the October Ada Monthly Meetup which will be taking place on the 7th of October at 13:00 UTC time (15:00 CET). As always, the meetup will take place over at Jitsi. Hopefully this time I will not have the same amount of technical issues…

If someone would like to propose a talk or a topic, feel free to do so! I will be talking about FOSDEM, but only for a couple of minutes, since the dates have already been announced (the same as every year, first weekend of February).

Here are the connection details from previous posts: The meetup will take place over at Jitsi, a conferencing software that runs on any modern browser. The link is [1]. The room name is "AdaMonthlyMeetup" and in case it asks for a password, it will be set to "AdaRules". I do not want to set up a password, but in case it is needed, it will be the one above without the quotes. The room name is generally not needed as the link should take you directly there, but I want to write it down just in case someone needs it.

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

From: Rod Kay <rodakay5@gmail.com>
Date: Sat, 23 Sep 2023 20:06:51 +1000

I might give a small talk on swig4ada, if that would be of interest.

If it's possible to request talks, I'd love to see an overview of 'Pragmarc' and 'Simple Components' by Jeffrey and Dmitry.

I used the neural net component of Pragmarc many years ago but have not kept up with Pragmarc's development. Also, I've been meaning to look into Simple Components for quite a while.

CfC 28th Ada-Europe Int. Conf. Reliable Software Technologies

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: CfC 28th Ada-Europe Int. Conf. Reliable Software Technologies
Date: Tue, 26 Sep 2023 12:06:18 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

[CfP is included in the Forthcoming Events Section —arm]

Ada-related Resources

[Delta counts are from July 28th to October 10th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 10 Oct 2023 18:22 CET
To: Ada User Journal readership

Ada groups on various social media:

- Reddit: 8_422 (+51) members [1]
- LinkedIn: 3_454 (+6) members [2]
- Stack Overflow: 2_365 (+20) questions [3]
- Gitter: 229 (-1) people [4]
- Telegram: 158 (-1) users [5]
- Ada-lang.io: 146 (+13) users [6]
- Libera.Chat: 72 (-1) concurrent users [7]

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

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

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

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

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

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

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

Repositories of Open Source Software

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: 10 Oct 2023 18:25 CET
To: Ada User Journal readership

GitHub: 1000* (+13) developers [1]

Rosetta Code: 940 (-1) examples [2]

38 (-3) developers [3]

Alire: 370 (+7) crates [4]

Sourceforge: 247 (+4) projects [5]

Open Hub: 214 (=) projects [6]

Codelabs: 57 (=) repositories [7]

Bitbucket: 37 (+6) repositories [8]

* This number is an unreliable lower bound due to GitHub search limitations.

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

[2] https://rosettacode.org/wiki/Category:Ada

[3] https://rosettacode.org/wiki/Category:Ada_User

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

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

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

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

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

Language Popularity Rankings

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

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

- TIOBE Index: 23 (=) 0.77% (=) [1]
- PYPL Index: 16 (=) 1.04% (-0.02%) [2]
- Stack Overflow Survey: 42 (=) 0.77% (=) [3]
- IEEE Spectrum (general): 36 (-1) Score: 0.0107 (new) [4]
- IEEE Spectrum (jobs): 29 (+4) Score: 0.0173 (new) [4]
- IEEE Spectrum (trending): 30 (+2) Score: 0.0122 (new) [4]

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

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

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

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

IRC Is Still Alive

From: Luke A. Guest <laguest@archeia.com>
Subject: Reminder that the IRC is still alive
Date: Wed, 16 Aug 2023 19:05:10 +0100
Newsgroups: comp.lang.ada

About twice a year we try to advertise the #ada channel on the Libera IRC network. The channel continues to be active and friendly. These days it averages about 63 users at a time, large enough to support lively and informative discussions but small enough so it's not a madhouse. The user numbers did suffer on the move when Freenode imploded.

Topics range all over the map, from building the latest GNAT to writing an OS in Ada to daily Ada programming issues to how to use PolyORB to use the Distributed Systems Annex. The stated topic is discussing Ada in the context of free and open-source software, but commercial users are equally welcome.

So fire up your favorite IRC client and come join us! The network is homed at irc.Libera.chat, but has servers all over the world. Visit www.Libera.chat on the web for details. Hope to see you soon!

Common HOL Phase 1 Reports

From: Luke A. Guest <laguest@archeia.com>
Subject: Common HOL Phase 1 Reports
Date: Wed, 30 Aug 2023 14:48:48 +0100
Newsgroups: comp.lang.ada

[This post refers to the Common High-Order Language program, started in 1975, of which Ada would be the outcome. DTIC stands for Defense Technical Information Center. —arm]

Edward Fish has managed to get the DTIC to scan in the other language's reports.

This has been a combined effort between a few of us on IRC to try to get the other two languages, blue and yellow released so we can see what could've happened.

This report contains all 4 language reports.

https://apps.dtic.mil/sti/trecms/pdf/ADB950587.pdf

From: Luke A. Guest <laguest@archeia.com>
Date: Wed, 30 Aug 2023 21:20:28 +0100

Now we have all 5 reference docs out in the open, yes 5. I want to ask the people who were there at the time, were there any analyses of the Tartan language? Design: https://apps.dtic.mil/sti/citations/ADA062815

From: Stéphane Rivière <stef@genesix.org>
Date: Thu, 31 Aug 2023 06:32:50 +0200

Fascinating (C) Spock.

Big up for this work!

US Government Looking into Memory Safe Programming

From: Ajdude <aj@ianozi.com>
Subject: US Government looking into memory safe programming
Date: Sun, 24 Sep 2023 22:28:56 -0000
Newsgroups: comp.lang.ada

The US Government is requesting information on adoption of memory safe programming languages and open-source software security. They're currently taking comments until October 9th. I think this is a good opportunity to help bring Ada back into the spotlight.

https://www.federalregister.gov/documents/2023/08/10/2023-17239/request-for-information-on-open-source-software-security-areas-of-long-term-focus-and-prioritization

From: Luke A. Guest <laguest@archeia.com>
Date: Mon, 25 Sep 2023 08:52:33 +0100

History is repeating itself. How long before they relax the requirements and idiots say "we can use C again, yay!"?

From: Stéphane Rivière <stef@genesix.org>
Date: Mon, 25 Sep 2023 11:59:57 +0200

> History is repeating itself.

+1

> How long before they relax the requirements and idiots say "we can use C again, yay!"?

By the time they discover Rust?

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 25 Sep 2023 12:38:54 +0200

> By the time they discover Rust?

Or when they realize that there is only one Rust compiler, and therefore that a single compiler virus could ruin the whole defense system.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Mon, 25 Sep 2023 17:55:08 +0200

> Or when they realize that there is only one Rust compiler, and therefore that a single compiler virus could ruin the whole defense system.

Maybe, given the emphasis on tools, verification and best practices, they might consider sub-languages, or profiles, of several existing languages.

It's not like memory-safety cannot be made available in languages other than Rust, I should think? Though, it seems to me that Rust has so much better market-aware development strategies than any other language since C, outside Microsoft's or Apple's areas of sales.

Also, I understand that Linux kernel development is steered towards Rust and LLVM. So, they have decided not to go back to the 80s, just pick some good bits and move on, possibly producing grust or crust while at it.

In order to pick well from Ada and the concepts embodied in it, imagine what parts of Ada should be thrown out, ignoring commercial enterprises living off legacy business? What changes to Ada are a good fit while aiming at memory safety, verification support, or light weight and safe parallel execution?

As you can see in [1], there is a suggestion to make money available to refactoring efforts.

[1] https://www.federalregister.gov/d/2023-17239/p-37

From: Luke A. Guest <laguest@archeia.com>
Date: Mon, 25 Sep 2023 17:21:57 +0100

> What changes to Ada are a good fit while aiming at memory safety, verification support, or light weight and safe parallel execution?

I started thinking about that here https://github.com/Lucretia/orenda.

From: Stéphane Rivière <stef@genesix.org>
Date: Tue, 26 Sep 2023 08:55:11 +0200

> Or when they realize that there is only one Rust compiler, and therefore that a single compiler virus could ruin the whole defense system.

Good point!

Still some doubts about their ability to reason that far ;)

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Tue, 26 Sep 2023 11:23:24 -0000

>Still some doubts about their ability to reason that far ;)

Whilst I have in the past refused to use lattice semiconductor hardware due to a CDN preventing secure compiler verification, whilst apparently none or few noticed.

I assume you mean trojaned compiler code inserted upstream to disable protections or ignore unsafe code?

Or do you mean utf-8 library code substitution aimed at a particular compiler?

Ada Advocacy Opportunity

From: Shark8 <onewingedshark@gmail.com>
Subject: Ada Advocacy Opportunity
Date: Wed, 27 Sep 2023 22:07:14 -0700
Newsgroups: comp.lang.ada

The federal government has issued an RFI (Request For Information) on the topic of increasing software security. 10-page max (excluding cover-page and appendix, if any).

Request for Information on Open-Source Software Security: Areas of Long-Term Focus and Prioritization

https://www.federalregister.gov/documents/2023/08/10/2023-17239/request-for-information-on-open-source-software-security-areas-of-long-term-focus-and-prioritization

Ada-related Tools

SPARK Reusable Components

From: Pragmada Software Engineering <pragmada@pragmada.x10hosting.com>
Subject: [Ann] SparkRC
Date: Tue, 1 Aug 2023 10:58:34 +0200
Newsgroups: comp.lang.ada

Those using SPARK may find this useful:

https://github.com/jrcarter/SparkRC

[From the website: SPARK Reusable Components – A few useful components to complement the Ada.Containers.Formal_* components, primarily to learn about proofs and functional correctness: queues, stacks, an O(logN)-searchable ordered structure, (...) maps. These can be fully proven to correctly implement their contracts and to be free of run-time errors. –arm]

Ada Binding to WolfSSL

From: Joakim Strandberg <joakimds@kth.se>
Subject: Announcing Ada binding to the wolfSSL library
Date: Thu, 3 Aug 2023 14:02:56 -0700
Newsgroups: comp.lang.ada

On the WolfSSL blog I saw the following announcement today:

Today we are happy to announce the availability of an Ada/SPARK binding that enables Ada applications to use post-quantum TLS 1.3 encryption through the WolfSSL embedded SSL/TLS library.

It opens the door to obtaining FIPS 140-3 and DO-178C certifications for Ada and Spark applications that use TLS for their encrypted communications and also makes them quantum-safe.

Check out the Ada/SPARK binding on GitHub here: https://github.com/wolfSSL/wolfssl/tree/master/wrapper/Ada

The Ada port is suitable for anything from IoT, embedded systems to Desktop and Cloud systems.

Contact us at facts@wolfssl.com, or call us at +1 425 245 8247 with any questions, comments, or suggestions.

URL to blog post: https://www.wolfssl.com/announcing-ada-binding-to-the-wolfssl-library/

LibAWS for Debian 12

From: philip...@gmail.com <philip.munts@gmail.com>
Subject: libaws for Debian 12 (Bookworm)
Date: Tue, 8 Aug 2023 09:55:53 -0700
Newsgroups: comp.lang.ada

Debian 12 no longer includes system packages for the AdaCore Ada Web Server Library. (Debian 11 had libaws20-dev et al).

I have managed to build AWS v23.0.0 for Debian 12 and published it in the nascent Munts Technologies Debian 12 Package Repository, available at:

http://repo.munts.com/debian12

The goop to build libaws*.deb is stored at: https://github.com/pmunts/libsimpleio/tree/master/libaws

SweetAda on NEORV32

From: Gabriele Galeotti <gabriele.galeotti.xyz@gmail.com>
Subject: SweetAda on NEORV32
Date: Thu, 10 Aug 2023 06:03:18 -0700
Newsgroups: comp.lang.ada

I've created a NEORV32 target platform in SweetAda (https://github.com/gabriele-galeotti).

NEORV32 (https://github.com/stnolting/neorv32) is a popular RISC-V SoC implementation in VHDL, suited for FPGAs.

The setup so far is blatantly primitive and runs under simulation by means of GHDL, outputting a welcome message inside the simulated UART console and continuously output the value of the mtime timer.

So far I have no FPGA hardware (besides the time) ready to create a real implementation, so if someone is using NEORV32 on real hardware, and is willing to test, it will be very interesting to know about a OK/KO flag feedback. The current setup needs only UART clocking parameters in the CTRL register, which I suppose it depends on the actual clock configuration. In the meantime I will continue to develop things inside the simulated GHDL environment.

WinRT v3

From: Alexg <agamper@bigpond.net.au>
Subject: [Ann] WinRt - version 3
Date: Sat, 26 Aug 2023 22:18:29 -0700
Newsgroups: comp.lang.ada

Dear Ada community

I have created a new git repo for the Ada binding to WinRT (now version 3).

This version is a cleaner implementation than the previous version and includes the following changes:

1) Wide strings are mapped to HStrings.

2) Async operations and actions are handled automatically.

3) code files now contain code at the Namespace level.

Git repo is located here
https://github.com/Alex-Gamper/Ada-WinRt3

GNAT Studio 24.0 for MacOS Ventura.

From: Blady <p.p11@orange.fr>
Subject: [ANN] GNAT Studio 24.0 for macOS Ventura.
Date: Wed, 6 Sep 2023 13:17:51 +0200
Newsgroups: comp.lang.ada

Here is a very preliminary version of GNAT Studio 24.0wa as a standalone app for macOS 13:

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2023-ventura

See readme for details.

Limitation: Ada Language Server has some latencies and doesn't respond when parsing source code with more 2000 lines. It may be due to some compilation options I missed.

There could be some other 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 to fix these limitations.

Simple Components v4.68

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Subject: ANN: Simple Components v4.68
Date: Sat, 30 Sep 2023 18:58:05 +0200
Newsgroups: comp.lang.ada

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

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

Changes (30 September 2023) to the version 4.67:

- Boolean types handling was added to the Python bindings;
- Python library search path for OSX fixed.

Ada and Operating Systems

Ada Support on Arch Linux

From: Rod Kay <rodakay5@gmail.com>
Subject: Ada Support on Archlinux
Date: Mon, 10 Jul 2023 04:22:50 +1000
Newsgroups: comp.lang.ada

Unfortunately, the attempt to have gprbuild and xmlada added into the official Archlinux package repositories has not been successful. Nonetheless, thanks to those who voted for these packages. The gist of the problem from Arch's Trusted Users (who manage adding new packages into the official repos) was "I don't use Ada, so I won't sponsor it.".

Given this situation, I've created a custom repository for the (50 odd) Arch Ada packages, currently served by a Linode. Details (and the current package list) can be found here ...

https://wiki.archlinux.org/title/Ada

If anyone has any suggestions for additional Ada projects to add to the package list, please let me know.

MacOS: Best Not Upgrade to Xcode/CLT 15.0

From: Simon Wright <simon@pushface.org>
Subject: macOS: best not upgrade to Xcode/CLT 15.0
Date: Wed, 20 Sep 2023 10:35:29 +0100
Newsgroups: comp.lang.ada

If you accept the Xcode/Command Line Tools upgrade to 15.0, you'll get crashes in the linker (`ld`). If you can't resist the upgrade, gnatmake or gprbuild with `-largs -Wl,-ld_classic`.

You probably won't be offered the upgrade unless you're already running Ventura.

MacOS Ventura 13.6 Update Problem

From: Moi <findlaybill@blueyonder.co.uk>
Subject: macOS Ventura 13.6 update problem
Date: Fri, 22 Sep 2023 21:02:17 +0100
Newsgroups: comp.lang.ada

Installing the macOS Ventura 13.6 security update clobbers GNAT. Specifically, the link stage fails:

```
> -macosx_version_min has been renamed to -macos_version_min
> 0  0x104de0f43  __assert_rtn + 64
> 1  0x104ce2f43 ld::AtomPlacement::findAtom(unsigned char, unsigned long long, ld::AtomPlacement::AtomLoc const*&, long long&) const + 1411
> 2  0x104cff431 ld::InputFiles::SliceParser::parseObjectFile(mach_o::Header const*) const + 19745
> 3  0x104d0fb71 ld::InputFiles::parseAllFiles(void (ld::AtomFile const*) block_pointer)::$_7::operator()(unsigned long, ld::FileInfo const&) const + 657
> 4  0x7ff80b631066 _dispatch_client_callout2 + 8
> 5  0x7ff80b642e09 _dispatch_apply_invoke + 213
> 6  0x7ff80b631033 _dispatch_client_callout + 8
> 7  0x7ff80b6410f6 _dispatch_root_queue_drain + 683
> 8  0x7ff80b641768 _dispatch_worker_thread2 + 170
> 9  0x7ff80b7cec0f  _pthread_wqthread + 257
> ld: Assertion failed: (resultIndex < sectData.atoms.size()), function findAtom, file Relocations.cpp, line 1336.
> collect2: error: ld returned 1 exit status
> gnatmake: *** link failed.
```

Simon's "magic formula", '-largs -Wl,-ld_classic' restores sanity. I guess the CLTs were updated without asking permission. 8-(

From: Simon Wright <simon@pushface.org>
Date: Sat, 23 Sep 2023 12:16:03 +0100

I managed to avoid this this morning (I've been resisting the attempted upgrade to CLT 15.0) by looking to see what was proposed, seeing that there were 2 upgrades (Ventura & CLT), and unchecking the CLT.

Iain Sandoe recommends [1] re-installing 14.3 (you should be able to download it from the developer.apple.com website, although you do need an apple ID to do that)

[1] https://github.com/iains/gcc-12-branch/issues/22#issuecomment-1730213294

From: Leo Brewin <leo.brewin@monash.edu>
Date: Sun, 24 Sep 2023 08:37:04 +1000

Hmm, this is odd. I'm having no problems. I'm running mac OS Ventura 13.6 (with the security update), CLT 15.0 and GNAT based on GCC 13.1.

From: Moi <findlaybill@blueyonder.co.uk>
Date: Sun, 24 Sep 2023 01:47:57 +0100

> Hmm, this is odd. I'm having no problems.

Ah! I'm still on GNAT 12.2.0.

I should have added that, although the link phase works, it produces this message, which is new:

> -macosx_version_min has been renamed to -macos_version_min

And Free Pascal Compiler version 3.2.2 [2021/05/16] for x86_64 also complains, thus;

> ld: warning: -multiply_defined is obsolete

But again, the linking succeeds.

From: Simon Wright <simon@pushface.org>
Date: Sun, 24 Sep 2023 12:55:56 +0100

> Hmm, this is odd. I'm having no problems. I'm running mac OS Ventura 13.6 (with the security update)

I was under the impression that 13.6 _was_ a security update!

Investigation so far shows that linking against the static Ada runtime (the default, -bargs -static) crashes, against the shared runtime (-bargs -shared) is OK.

With 12.2.0, 13.1.0, 14.0.0.

It occurs to me that it might be _any_ static library? ... later

From: Simon Wright <simon@pushface.org>
Date: Sun, 24 Sep 2023 16:38:36 +0100

> It occurs to me that it might be _any_ static library? ... later

Yes, so it is. Damn.

GNAT Linking and MacOS

From: Moi <findlaybill@blueyonder.co.uk>
Subject: GNAT linking and macOS
Date: Wed, 27 Sep 2023 20:30:17 +0100
Newsgroups: comp.lang.ada

I installed 14.0, Sonoma, on my M1 Mac last night.

The good news:

Using GNAT 12.2.0, it all just works, *so long as* I REMOVE '-largs -Wl,-ld_classic' from the linker options!

From: Simon Wright <simon@pushface.org>
Date: Thu, 28 Sep 2023 14:32:47 +0100

> Using GNAT 12.2.0, it all just works, *so long as* I REMOVE '-largs -Wl,-ld_classic' from the linker options!

Likewise, but that didn't work for me.

It turns out there's an environment variable DEFAULT_LINKER, which with the 15.0 CLT would be set to

```
export DEFAULT_LINKER=/Library/Developer/CommandLineTools/usr/bin/ld-classic
```

(the Xcode equivalent is much longer & more inscrutable)

I haven't tried this.

From: Simon Wright <simon@pushface.org>
Date: Thu, 28 Sep 2023 22:00:29 +0100

> Likewise, but that didn't work for me.

You can't reinstall the 14.3 CLT under Sonoma (it's "too old"). I reinstalled from a Time Machine backup, but if you don't have that set up I'd recommend taking a copy of /Library/Developer/CommandLineTools before updating to 15.0.

> It turns out there's an environment variable DEFAULT_LINKER [...]

> I haven't tried this.

This affects building the compiler, not using it.

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Date: Fri, 29 Sep 2023 19:30:27 -0700

[...] Are there any other workarounds to solve the inability to link? This does not only adversely affect Ada, but everything that uses a linker, BTW.

From: Simon Wright <simon@pushface.org>
Date: Sat, 30 Sep 2023 14:55:09 +0100

> Are there any other workarounds to solve the inability to link?

I have some evidence that the issue only arises with static libraries. Not much help.

We're hoping that the 15.1 release of Command Line Tools fixes this. In the meantime,

(1) using gnatmake, or gprbuild without changing the GPR:

```
$ gnatmake foo.adb -largs -Wl,-ld_classic
```

or

```
$ gprbuild -P foo -largs -Wl,-ld_classic
```

(2) modifying the GPR by adding a new package Linker:

```
package Linker is
   for Default_Switches ("ada") use
   ("-Wl,-ld_classic");
end Linker;
```

(3) if you already have a package Linker, modify it as above.

Ada Practice

Text vs Binary File Identification

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Subject: Using "pure" (?) Ada, how to determine whether a file is a "text" file, not a binary?
Date: Sat, 1 Jul 2023 10:15:25 -0700
Newsgroups: comp.lang.ada

Another very beginner question here...

Using "pure" (?) Ada, how to determine whether a file is a "text" file, not a binary?

Kind of like using the UNIX/Linux "file" command, but doesn't have to be comprehensive (yet). Something like the Perl "-T" feature.

On the other hand, if there already exists an Ada implementation of the UNIX "file" command as a library, could you point me to that?

As a side question, how does one read "binary" files in Ada?

A UNIX/Linux use case for the previous sentence is the concatenation of two (or more) "binary" files that were created using the UNIX/Linux "split" command.

So I'd be interested in emulating the UNIX "cat" command for "binary" files.

These are just personal experiments for learning how to do all kinds of Ada I/O...

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sat, 1 Jul 2023 22:39:27 +0200

> Using "pure" (?) Ada, how to determine whether a file is a "text" file, not a binary?

That depends on the definition of a text file. Under Unix and Windows, all files are sequences of bytes, and so may be considered sequences of Characters, and so text files.

If you can define what distinguishes text files from binary files, then it should be fairly easy to write Ada to distinguish them.

For example, if a text file is one in which all the characters, except line terminators, are graphic characters, then it should be clear how to determine whether a file meets that definition of a text file.

> As a side question, how does one read "binary" files in Ada?

Ada has Direct_IO, Sequential_IO, and Stream_IO for reading binary files. Which you would use and how to use it depends on what's in the file and what you need to do with it.

Memoization in Ada

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Subject: memoization in Ada? Hash ADT?
Date: Fri, 21 Jul 2023 20:50:04 -0700
Newsgroups: comp.lang.ada

I'm working on the Rosetta Code task:

"Stirling numbers of the second kind"

I have a working recursive solution written in Ada but I'd like to memoize it to cut down on the redundant and duplicative calls (similar to a recursive solution to calculating the Fibonacci sequence).

So I think I need a hash ADT (which I've used in Perl) but I've never used in Ada.

So I want to preserve the calculation of the Stirling2 for each N and K so I can do a lookup. If this were based on a single unsigned integer, an array would suffice. Maybe a 2d array would suffice?

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Date: Fri, 21 Jul 2023 22:30:44 -0700

I solved the specific problem using a 2d array for caching. This is not memoization, per se, but this works very well. The recursive calls are now very fast as there is a maximum of one calculation per recursive call.

So, any resources on how to write Ada programs that take advantage of memoization?

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Mon, 24 Jul 2023 14:18:25 -0700

> So, any resources on how to write Ada programs that take advantage of memoization?

Look here https://forum.ada-lang.io/ for discussions about Advent of Code puzzles. Some solutions use (and need, for completing in a reasonable time) memoization.

You find with HAC (https://hacadacompiler.sourceforge.io/) a set of solutions (search "memoiz*" or "cache"), mostly compiling with the HAC subset.

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Date: Tue, 25 Jul 2023 21:38:02 -0700

Thanks for the pointer to the forum.

Regarding HAC, isn't that Windows-only? I'm on a Mac (M1 chip). I'll look again at HAC.

From: Simon Wright <simon@pushface.org>
Date: Wed, 26 Jul 2023 08:50:01 +0100

> Regarding HAC, isn't that Windows-only? I'm on a Mac (M1 chip). I'll look again at HAC.

It worked well enough for me to find a failing test case (now fixed!)

From: Gautier Write-Only Address <gautier_niouzes@hotmail.com>
Date: Wed, 26 Jul 2023 14:36:15 -0700

> Regarding HAC, isn't that Windows-only?

Not at all :-)

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Date: Wed, 26 Jul 2023 15:18:51 -0700

> > Regarding HAC, isn't that Windows-only?

> Not at all :-)

Downloaded and ran demo. Will experiment further as time permits. Nice!

Rosetta Proper Divisors Fails to Compile

From: Kenneth Wolcott kennethwolcott@gmail.com
Subject: Rosetta Code task Proper divisors fails to compile
Date: Tue, 25 Jul 2023 21:49:49 -0700
Newsgroups: comp.lang.ada

Trying to understand (and use) a Rosetta Code task (Proper divisors)... https://rosettacode.org/wiki/Proper_divisors#Ada

it fails to compile

```
gnatmake -vh ./proper_divisors.adb

GNATMAKE 13.1.0
Copyright (C) 1992-2023, Free Software Foundation, Inc.
"proper_divisors.ali" being checked ...
  -> "proper_divisors.ali" missing.
gcc -c -I./ -I- ./proper_divisors.adb
generic_divisors.ads:11:08: error: (Ada 2005) cannot copy object of a limited type (RM-2005 6.5(5.5/2))
generic_divisors.ads:11:08: error: return by reference not permitted in Ada 2005
End of compilation
gnatmake: "./proper_divisors.adb" compilation error
```

Why does this work for the submitter of the Rosetta Code task and not for me?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Wed, 26 Jul 2023 10:36:12 +0200

For some reason your gnatmake seems to be defaulting to -gnat05 mode. This code has an expression function, which is Ada 12, so try adding -gnat12 to the command.

You also should not need to put "./" in front of the file name, though I don't see how that would make a difference.

From: Kenneth Wolcott <kennethwolcott@gmail.com>
Date: Wed, 26 Jul 2023 11:30:12 -0700

Thank you for your suggestion. Doesn't seem to have any effect.

(*SIGH*)

```
gnatmake -vh -gnat2012 proper_divisors.adb

GNATMAKE 13.1.0
Copyright (C) 1992-2023, Free Software Foundation, Inc.
"proper_divisors.ali" being checked ...
-> "proper_divisors.ali" missing.
gcc -c -gnat2012 proper_divisors.adb
generic_divisors.ads:11:08: error: (Ada 2005) cannot copy object of a limited type (RM-2005 6.5(5.5/2))
generic_divisors.ads:11:08: error: return by reference not permitted in Ada 2005
End of compilation
gnatmake: "proper_divisors.adb" compilation error
```

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Wed, 26 Jul 2023 21:57:37 +0200

> Thank you for your suggestion. Doesn't seem to have any effect.

Interesting. I get the same results with GNAT 12.

Looking more closely at the code, I think the error, while its msg is misleading, is correct. A function that returns a limited type can only return an aggregate, a function call, or an object declared by an extended return statement. The generic formal object None is none of these.

Changing the generic parameter to a function

```
with function None return Result_Type;
```

makes the code correct. You need to change the definition of Empty in Proper_Divisors

```
function Empty return Pos_Arr is
(1 .. 0 => <>);
```

and create a function to supply for the 2nd instantiation

```
function None return Natural is (0);<
...
package Divisor_Count is new
Generic_Divisors (Result_Type => Natural,
    None => None,
    One => Cnt, Add =>  "+");
```

and then it compiles and runs.

Another possibility is to make Result_Type simply private, though that is slightly less general. That may be how the OP got the code to compile and run. It might be that limited was added later because the OP saw that it could be, and never tested the change.

Using 'Image with Alire

From: Seth Workman <saworkman1@gmail.com>
Subject: Using 'Image with Alire
Date: Sun, 6 Aug 2023 14:17:40 -0700
Newsgroups: comp.lang.ada

I have only started learning about Ada recently and have discovered the 'Image attribute that can be used on all types starting in Ada 2022.

I am using Alire and added the following to include the `-gnat2022` switch.

```
   for Default_Switches ("Ada") use
   Learning_Config.Ada_Compiler_Switches
   & ("-gnat2022");
```

The Alire documentation warns about switches ~"In general, this should be avoided to preserve consistency in the ecosystem"

Is this the correct way about adding this switch or is there a way to use a toolchain that already has it by default?

From: Simon Wright <simon@pushface.org>
Date: Sun, 06 Aug 2023 22:58:12 +0100

> Is this the correct way about adding this switch or is there a way to use a toolchain that already has it by default?

This works fine, but in your alire.toml you could say

```
[build-switches]
"*".ada_version = "ada2022"
```

or

```
[build-switches]
"*".ada_version = ["-gnat2022"]
```

See "Release Information" (near the end) and "Build Profiles and Switches" in the documentation.

From: Seth Workman <saworkman1@gmail.com>
Date: Sun, 6 Aug 2023 15:17:07 -0700

I see now, I think using
"*".ada_version = "ada2022"
is better for this case.

Parallel Loops in GNAT

From: Jerry <list_email@icloud.com>
Subject: Parallel loops in GNAT?
Date: Fri, 11 Aug 2023 17:44:38 -0700
Newsgroups: comp.lang.ada

Does GNAT such as Simon's GCC 13.1.0 for macOS aarch64 allow parallel loops and blocks?

From: Simon Wright <simon@pushface.org>
Date: Sun, 13 Aug 2023 11:39:59 +0100

No, sorry, that's one of the advanced features that AdaCore aren't working on yet.

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

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 13 Aug 2023 14:44:02 +0200

ISO/IEC 8652:2023 is the first Ada standard to be approved without a working implementation. Probably a bad sign.

From: Jerry <list_email@icloud.com>
Date: Sun, 13 Aug 2023 15:03:20 -0700

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

Thanks, Simon. I saw that blog post but since it's nearly three years old, I was hopeful. My O(N^4) radar simulations will have to remain slow. :-(

Unifont Statically Compiled and Stack Size

From: Micah Waddoups <micah.waddoups@gmail.com>
Subject: Unifont static compiled and stack size...
Date: Sun, 13 Aug 2023 09:16:27 -0700
Newsgroups: comp.lang.ada

I tried to compile the Unifont hex file, converted with a script into variable Ada code, but it went on forever, gradually blowing up my memory. I used an .ads file and aimed for a static build, but I suspect I would have hit the stack size limit for executables if I succeeded

My request for insight is:

(A) How do you recommend I compile the Unifont into a form that is usable within my Ada program. (I am thinking compiling C and importing, since C is so basic it might just work, but even better would be Assembly and I don't know how to import a large data variable compiled in Assembly or if I can even compile that much data using Assembly... It should work, but the compiler might complain and I still have to figure out the importing part.)

(B) Do you think this has a chance of succeeding if I compile the font as a shared library? That doesn't affect initial stack limits, right?

Just to be clear, I am trying to import values 0 .. 16#FFFFF#, way beyond the two byte limit that so many libraries are functionally bound by in one bottle neck or another. I want to support something similar to 'kmscon', but with some shortcuts since I don't want to redo everything that others have done well. I just want to finish my library to a point of usefulness and focus on other projects. I felt compelled to create this library because every other library I looked at was broken in some way and even the most common font systems fail to support Unicode's full character range, making much of it useless. I figured I could create the exact effects that I am trying to with Unifont both in graphical Windows of various OSs, and on the Linux terminal if I rewrite enough of the low level code that I don't have to rely on the less complete existing libraries. Admittedly, I have too little time to work on it, and am so far behind other people's wonderful work that I will certainly have many holes and omitted functionality that should eventually be added later. My goal is to both make my programming projects possible and free certain features from too restricted licensing.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 14 Aug 2023 11:07:14 +0300

> My request for insight is: (A) How do you recommend I compile the Unifont into a form that is usable within my Ada program.

Could you show a little of the script-generated Ada code, just for us to understand the approach you have taken to represent the Unifont file?

While waiting for that, it may help the compiler if you disable the more advanced compiler optimizations, because some of them are super-linear in complexity and probably (but depending on the exact form of the Ada code) do not help for this case, anyway.

> (I am thinking compiling C [...] but even better would be Assembly [...]

I'm not familiar with the structure of the Unifont file, but if it is something like a table with rows and columns, it should be rather easy to translate it into a list of assembly-language constant-data definitions.

Assemblers are typically linear in complexity and should be able to handle large data definitions, assuming there is not a myriad of assembler labels that make references between different parts of the data structure.

Exporting a data object from assembly to Ada is simple and does not depend on the size of the object (but I admit I don't know how this is done for dynamically linked libraries). The only part that needs thought is how to define the Ada type of the object, but if the Unifont file is a row-column table, in other words a list of "row" records, that should be straight-forward too.

So I think the assembly-language solution is highly likely to work; its drawback, of course, is non-portability. But the constant-data definitions of most assembly languages are very similar to each other, so the assembler-generating script should be easy to port to different assembly languages.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 14 Aug 2023 10:31:43 +0200

> I'm not familiar with the structure of the Unifont file [...]

A comparable case. I have XPM to Ada translator (for having built-in images in GTK). It simply creates packages with declarations of initialized arrays. No stack issues.

Doing something like that for bitmap fonts is just as simple. The only minor issue is creating an index map: code point to the bitmap image name (array), because a flat array would blow out.

P.S. I always wanted static functions in Ada for the purpose of all static initializations of objects like maps etc.

From: Kevin Chadwick
<kc-usenet@chadwicks.me.uk>
Date: Mon, 14 Aug 2023 09:25:07 -0000

> a flat array would blow out.

What does blow out mean in this context?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 14 Aug 2023 11:39:45 +0200

> What does blow out mean in this context?

If you tried:

```
type Font_Type is array (Code_Point) of
   Bitmap_Ptr;
```

The range of code points is 0..16#10FFFF#. E.g. when I implemented Ada.Strings.Maps for Unicode, I could not use such arrays either as the native ASCII implementation does.

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Mon, 14 Aug 2023 12:06:35 +0200

> I tried to compile the Unifont hex file, converted with a script into variable Ada code, but it went on forever, gradually blowing up my memory. I used an .ads file and aimed for a static build, but I suspect I would have hit the stack size limit for executables if I succeeded

As I understand it, the file in question is for code points 0 .. 16#FFFF#, with a maximum of 32 bytes per code point. A straightforward representation of this is

```
package Unifont is
   type Byte is mod 2 ** 8 with Size => 8;
   type Line is array (1 .. 2) of Byte
      with Size => 16;
   type Bitmap is array (1 .. 16) of Line
      with Size => 256;
   function Width_8 (Map : in Bitmap)
   return Boolean is
      (for all L of Map => L (2) = 0);
   type Code_Point is mod 16#FFFF# + 1;
   type Font_Map is array (Code_Point)
      of Bitmap with Size => 2 ** 24;
Font : constant Font_Map :=
      (others => (others => (others => 0) ) );
end Unifont;
```

Font will occupy 2 MB.

We can test this with

```
with Ada.Text_IO;
with Unifont;
procedure Unifont_Test is
   -- Empty
begin -- Unifont_Test
   Ada.Text_IO.Put_Line (Item =>
   Unifont.Font (0) (1) (1)'Image);
end Unifont_Test;
```

and see what happens:

```
$ gnatmake -m -j0 -gnat12 -gnatan -gnato2 -
O2 -fstack-check unifont_test.adb
x86_64-linux-gnu-gcc-12 -c -gnat12 -gnatan -
gnato2 -O2 -fstack-check
unifont_test.adb
x86_64-linux-gnu-gcc-12 -c -gnat12 -gnatan -
gnato2 -O2 -fstack-check unifont.ads
x86_64-linux-gnu-gnatbind-12 -x
unifont_test.ali
x86_64-linux-gnu-gnatlink-12 unifont_test.ali
-O2 -fstack-check
$ ./unifont_test
0
```

so this representation seems to be workable. It should be trivial to write a program to read the file and produce the real array aggregate for Font.

From: Micah Waddoups
<micah.waddoups@gmail.com>
Date: Mon, 14 Aug 2023 08:10:01 -0700

Jeff, you missed a digit - it's five F's 16#FFFFF# because that is as high as Unifont goes in my copy of the hex file. [...]

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Mon, 14 Aug 2023 17:59:16 +0200

> Jeff, you missed a digit - it's five F's 16#FFFFF#

You're right. Sorry for misreading that. But increasing Code_Point to include 16#F_FFFF# still works for me.

Unicode defines code points up to 16#10_FFFF#, but perhaps those over 16#F_FFFF# are unused. Increasing Code_Point to include 16#10_FFFF# still works for me. That's 34 MB. It won't fit on the stack, but luckily library-level constants aren't allocated on the stack.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 14 Aug 2023 18:02:58 +0200

What are you going to do with this?

1. This is not a font usable in a GUI framework.
2. This is not a drawable in-memory image for a GUI framework either. Provided, you wanted to render obtained images manually. These images must be in a format supported by the corresponding engine.

E.g. GTK uses Pixbuf representation for drawable in-memory images. Which is

```
type Pixbuf_Image is array
   (Natural range 0..N*M-1) of GUChar;
pragma Convention (C, Pixbuf_Image);
```

containing 4 channels RGB + alpha, row-wise.

And, no, normally you cannot draw in an arbitrary OS window.

If you are so keen to use GNU Unifont, why do not you install it from its available formats like TrueType and be done with that? What is wrong with other fixed-size fonts?

Why do you want to render glyphs manually instead of using existing OS facilities and GUI libraries? You cannot get around these libraries without rewriting device drivers and who knows what else making the code highly non-portable.

From: Micah Waddoups
<micah.waddoups@gmail.com>
Date: Mon, 14 Aug 2023 21:48:03 -0700

> What are you going to do with this?

Dmitry, of course, you are correct. This was an attempt to take an image of the font source. [...]

The rendering is done with a small cache of per-need rendered Glyphs - each rendered glyph is at least eight times larger before any styling or transformation. Rendering all the glyphs at once takes up more memory than is needed and presupposes the destination format. So, in reality, more processor work is being done by doing it in stages, but less is being done in each stage, so there is more room for other processing at each stage. This is just the raw bitmap font that I want in the program without having to process and read the hex file every time it loads (that would be a very inefficient design).

In the first rendering, the glyphs that are actually used are rendered into a cache with only Alpha values. This is essentially gray-scale and is the value-map used for any transformations, glyph combining, and plotting on a per-line image plot map (which is another abstraction by line and index/column of just the first..last pixel boundaries). The plot map can very quickly be changed for insertions, deletions, changing text direction or flow, etc. and only at the final rendering of the View-able area is the Alpha transformed into full color (4-byte with alpha) to be sent to the GL, framebuffer, GTK, or other pixel-map handling system. Much like modern rendering systems, a lot of calculations happen behind the scenes at each key-press, only it is my attempt to combine it all into one library and scale it

into a project that can be plugged into whatever graphics system is being used. It is probably slower in response to input than GTK or any native text handling system because effects, layers of glyph-combinations, markups (squiggle line, underline, etc), and colorization all go into the font rendering before it hits the graphics buffer, but it feels to me more correct and avoids having to add post-rendering effects as a later stage as much as possible. Markups are done late, just before rotation (if any), but that is just as it must be, since they are effectively a rendered element of a different size than the individual glyphs they markup, yet still done before the rendered map is turned over to the graphics system.

The reason people rely on device drivers, native widgets, and less-portable combinations of libraries is that they incorporate the features and functionality desired into them, including how to handle input. I am attempting to take the lowest level of input (key-presses, clicks, touch, etc.) like a video game might, and outputting an image already rendered for display, thus replacing the normally convenient functionality provided by other systems. I am *not* trying to replace actual device drivers or rely on a particular operating system's device access scheme. That is why I am considering other well established libraries supporting GL or maybe SDL. Not all programs render text and input with the convenient systems, so this is nothing new. Also, since this scheme still tracks each character by line and index, even if I am only able to support text-mode terminal interface in one situation, that will only prevent using the full Unicode range of glyphs and combinations, not disable my ability to send/receive any given text with the display.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Tue, 15 Aug 2023 10:40:45 +0200

> P.S. I always wanted static functions in Ada for the purpose of all static initializations of objects like maps etc.

If data form the equivalent of a static Ada array, thus a mapping from an index type to a value type, could you approximate the static initialization of maps using expression functions?

Simplifying example:

```
package sttc is
   type Key is range 1 .. 7;
   type Value is new Character;
   type Cursor is private;

   function lookup (K: Key) return Cursor;
   function element (C: Cursor) return Value;

private
   type Cursor is new Key;
end sttc;
```

```
package body sttc is
   function lookup (K: Key) return Cursor is
      (Cursor (K));
   function element (C: Cursor) return Value
      is
      (case C is
       when 1 => 'M',
       when 2 => 'M',
       when 3 => 'X',
       when 4 => 'X',
       when 5 => 'I',
       when 6 => 'I',
      when 7 => 'I'
      );
end sttc;
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 16 Aug 2023 08:17:50 +0200

> could you approximate the static initialization of maps using expression functions?

In general case no. Initialization cannot be decomposed into functions. E.g. when it requires global [yet static] data, many places to set etc.

P.S. Expression functions are evil. I wonder why there is no expression gotos and labels? If you sell your soul to the devil, get the whole package! (:-))

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 17 Aug 2023 22:04:18 -0500

> P.S. Expression functions are evil. I wonder why there is no expression gotos and labels? If you sell your soul to the devil, get the whole package! (:-))

Ada only allows expressions to be evaluated at elaboration time (outside of generic instantiations), and expressions have a well-defined and simple control flow (even accounting for conditional expressions and quantified expressions, both of which implicitly appear in aggregates even in Ada 83 - allowing programmers to write them explicitly makes the code more readable than the horrible work-arounds commonly used pre-Ada 2012). Gotos and labels have arbitrary control flow, which can be much harder to analyze. (Janus/Ada converts the majority of code into an expression form for optimization - essentially most id statements become if expressions, and so on. It simply punts when the control flow is too complex to convert, so the unrestricted use of gotos effectively prevents most optimization as well as static analysis.)

If it was up to me, I would have left out declare expressions and quantified expressions, so the capabilities of expression functions would have been much more limited. But it seems valuable to be able to abstract an expression without changing the semantics (as requiring a separate body does).

GCC Support for Ada 2022

From: philip...@gmail.com <philip.munts@gmail.com>
Subject: Which GCC releases have how much support for Ada 2022?
Date: Mon, 14 Aug 2023 13:30:06 -0700
Newsgroups: comp.lang.ada

I am interested in cross-compilers for Linux boards such as the Raspberry Pi. I have successfully built cross-compilers for GNAT/GCC 12.3.1 on Debian 12 using the latest Arm GNU manifest and Linaro ABE, and Debian 12 has system packages for GNAT/GCC 12.2.0, native and cross, as well.

It isn't clear to me from the GCC release notes how complete the support for Ada 2022 is in GNAT/GCC 12.2.0 or 12.3.1. Is there a document or table somewhere that keeps track of that? And how does GCC 12 compare with GCC 13 WRT Ada 2022?

From: Micah Waddoups <micah.waddoups@gmail.com>
Date: Tue, 15 Aug 2023 14:12:18 -0700

[...] You may be able to find most of what you are looking for with the link in his answer:
https://blog.adacore.com/ada-202x-support-in-gnat

From: Maxim Reznik <reznikmm@gmail.com>
Date: Mon, 21 Aug 2023 13:04:30 -0700

> And how does GCC 12 compare with GCC 13 WRT Ada 2022?

This post could be helpful:
https://forum.ada-lang.io/t/gcc-13-1-released/374/3

In https://learn.adacore.com/courses/whats-new-in-ada-2022/index.html course there are GCC versions per feature. Unfortunately the feature list is not complete.

Parameterised 'Image Attributes

From: Rod Kay <rodakay5@gmail.com>
Subject: Parameterised 'Image Attributes
Date: Fri, 18 Aug 2023 17:18:29 +1000
Newsgroups: comp.lang.ada

There has been some recent discussion on #ada irc regarding formatted output.

Would it be possible/desirable to allow the 'Image attribute to have formatting parameters ? Something along the lines of ...

```
put_Line (some_Integer'Image
   (Width => 5, Padding => '0'));
```

... and similar 'Image attribute parameters for other types.

If the parameters have defaults, then there should not be any backwards compatibility issues (I think).

From: Luke A. Guest <laguest@archeia.com>
Date: Fri, 18 Aug 2023 09:25:44 +0100

I wanted them for ages, but there was a conversation ages ago where someone on here said attributes were for "debugging only," yet that's not what the ARM says.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Fri, 18 Aug 2023 11:53:44 -0700

TeleSoft's compiler (which I worked on) had 'Extended_Image and 'Extended_Value attributes that worked like that. I found them quite useful -- especially as an easy way to drop the leading space on Integer'Image.

One small problem was that we had different parameters for integer and enumeration types, which introduced an ambiguity for discrete formal types.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 19 Aug 2023 11:14:34 +0200

I wanted them for ages, but [...] someone on here said attributes were for "debugging only,"

The intent of the 'Image attribute is to have a quick representation, mainly for debugging purposes. If you want nice formatted output, use the Put procedure on String from Text_IO.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Aug 2023 12:03:09 +0200

> The intent of the 'Image attribute is [...] mainly for debugging purposes.

It seems that for the vast majority of Ada users this intent was wrong...

> If you want nice formatted output, use the Put procedure on String from Text_IO.

Put does not supersede 'Image. Put is I/O. 'Image is pure string formatting. Put is generic and requires instantiation of some package with some difficult-to-guess name. 'Image is built-in [statically] dispatching and generated automatically by the compiler.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 19 Aug 2023 13:56:14 +0200

> It seems that for the vast majority of Ada users this intent was wrong...

The vast majority of Ada users ignore a number of useful features provided by the language, and keep asking for improvements that are already there...

> Put does not supersede 'Image. [...]

Yes, Put has nothing to do with 'Image. Yes, Put requires instantiation. So what? Ada is more verbose, in favor of stricter typing. Ease of reading over ease of writing has always been a major design principle of Ada - although I confess it had a bad effect on its popularity, people want to write fast and ignore long term maintenance issues.

If you want formatting on an integer type (with or without IO), you instantiate Integer_IO. I don't find it hard to guess the name... Maybe you had something else in mind?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Aug 2023 15:01:36 +0200

> The vast majority of Ada users ignore a number of useful features [...]

Or these features are not that useful? Language users and designers have often different perspectives...

> Yes, put requires instantiation. So what? Ada is more verbose, in favor of stricter typing.

I don't see how instantiation is stricter typing. In fact instantiation introduces overloading (static ad-hoc polymorphism) which was always frowned upon at as less type safe than overriding.

> Ease of reading over ease of writing has always been a major design principle of Ada

I don't buy this either. It is

```
Put (X) vs. X'Image
```

equally readable and writable. If you refer to the instantiation noise or with/use clauses you would require to put somewhere far above in the package, that is not ease of reading. That is just meaningless noise.

> I don't find it hard to guess the name... Maybe you had something else in mind?

Yes, all other types that might require formatting. I doubt anybody, but a language lawyer could name the package appropriate for formatting a fixed-point type without looking into the RM. Which is absolutely unneeded as 'Image would be perfectly OK if it had the necessary parameters. All that generic text I/O packages are unnecessary as the stream I/O case perfectly illustrates. Ada 95 did stream I/O if not right, but far better making 'Read, 'Write etc attributes overridable. The problem of generic mess solved. We do not have and do not need any generics for stream I/O. Good riddance.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sat, 19 Aug 2023 17:27:25 +0200

> The intent of the 'Image attribute is to have a quick representation, mainly for debugging purposes.

There is a common problem across many types and problem domains of having a function that returns a string of an appropriate length representing a value of the type with desired formatting. Common examples include numeric values, dates, and times. The resulting string is usually combined with other information into a message that may be stored in memory for a while, though it is rare for it not to be output eventually. As an example, the message may be put on a protected queue for later output by a logging task.

Ada 83 tended not to include anything that the developer could implement; there was no math library or image functions for dates or times. The 'Image attribute was provided, but is unsuited for most such uses.

The use of the Text_IO generic sub-pkg Put procedures that output to strings is not convenient because they are procedures, not functions.

Later versions of Ada included more support for such needs, but not for numeric values.

The obvious solution is to have a library containing appropriate functions, which can be built around the Put procedures while still being functions. Such functions would need to be generic, unlike attribute functions which are automatically available for all types.

The conflict between this common need and the minimal functionality provided by 'Image results in such requests. It seems desirable for the language to provide such functions, and extending the 'Image functions seems like a reasonable way for it to do so, regardless of the original intentions for the attribute.

One library with such functions is the PragmAda Reusable Components (https://github.com/jrcarter/PragmARC). The package PragmARC.Images (https://github.com/jrcarter/PragmARC/blob/Ada-12/pragmarc-images.ads) provides such functions for integer and floating-point types. Function PragmARC.Images.Image is an instantiation for Standard.Integer.

PragmARC.Date_Handler (https://github.com/jrcarter/PragmARC/blob/Ada-12/pragmarc-date_handler.ads) provides image functions for dates and times; although the language now provides a function for a date-time image, Date_Handler continues to be useful as it provides for customized formats rather than the single format provided by Ada.Calendar.Formatting. Many users also find the semantics of the latter's time-zone parameter to be confusing.

ISO/IEC 8652:2023 provides a date-time image function the returns the image for the local time zone, but as there are no compilers* for this version of the language, I don't consider that relevant.

(*A compiler for a version of the language implements the entire core language of that version of the ARM.)

From: Moi <findlaybill@blueyonder.co.uk>
Date: Sat, 19 Aug 2023 17:49:41 +0100

> The intent of the 'Image attribute is to have a quick representation, mainly for debugging purposes.

My code uses 'Image heavily, because it is usually the neatest and the clearest way to format many strings that mingle words and numbers.

I sometimes have to pass the result of 'Image to a function that implements the kind of functionality people are asking for, and it would be even neater and clearer if I could get that with parameters to 'Image itself.

None of that output has anything to do with debugging.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sun, 20 Aug 2023 02:25:20 -0500

The profile of the Image attribute is:

```
X'Image
```

where X is an object of any type (or value of most types). And the old, unnecessary form was:

```
S'Image(X)
```

where S and X are of any type.

If one tries to add parameters to this, one gives up this nice form for a mis-mash of profiles for various classes of types. Moreover, the result no longer composes in the obvious way (necessary to have Image for records and arrays).

Additionally, one ends up with a magic mechanism that only the compiler can use. That *never* is a good idea. Especially as there now is a way to allow Image to support user-defined types. It would seem necessary to also support user-defined formatting parameters (else one has magic only applicable to a handful of language defined types).

Attributes do not allow named parameters outside a few special cases, and *never* allow reordering of parameters. Does that need to change, too?

Float input/output in particular is very large, especially when all of the formatting options are included. Do you really want to drag that into *every* Ada program, whether it uses it or not??

'Image is convenient for integer and enumeration output, and one can format them in the rare case where that is necessary. But it is useless for float output -- manual reformatting the output of 'Image would round the results incorrectly.

Ada has few built-in facilities because its primary purpose is to support the development of proper ADTs. Ease of writing is not a goal at all, and in most cases, the extra text is valuable to compilers and tools (even if it is not so valuable to human readers). If it was up to me, I would eliminate most of the shortcuts from Ada and require everything to be written out. (IDEs could/should do most of that for you anyway, so the extra text is not adding much effort.)

Ergo, I hope this idea is dead-on-arrival. I certainly won't be involved in it, that's for sure.

From: G.B. <bauhaus@notmyhomepage.invalid>
Date: Sun, 20 Aug 2023 09:53:11 +0200

> The conflict between this common need and the minimal functionality provided by 'Image results in such requests.

So, also

- See how other languages address formats (good bits, bad bits).
- Consider use cases.
- I/O is the program(mer)'s raison d'être. Can we easily Put something into a stream without the help of a suitable library?

Could there be a language defined type F whose purpose is to support the description of formats? Objects of type F would "configure" what 'Image does when computing a representation of a date, a number, ...

```
My_Length'Image (Arg => diameter,
   Format => ___);
```

Some use cases:

- I18n of number formats (cf ARM F.3), CHF 1'234'000.–
- Handle ubiquitous ISO formats of date-time (as mentioned below; also cf. ARM 9.6.1)
- reporting,
- integrate own output with output of other system components (a site-wide monitoring system searches outputs, say)
- fill in templates when these do not support formatting
- 'Input an object of type F at run-time, so that program's use of 'Image can be changed according to customer's local expectations.
- support the formalized exchange of "numerical" data in heterogeneous systems, using text streams.

These use cases are about the O of I/O. By symmetry, it would be nice to have implementations of Ada that support the I part of this kind of I/O, I think, with work to be split between implementers and programmers.

```
My_Length'Value (Arg => diameter,
   Format => ___);
```

Or perhaps multimethods that take a stream and a format when they need to write a value?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 20 Aug 2023 11:27:47 +0200

> Could there be a language defined type F whose purpose is to support the description of formats?

Not without multiple dispatch support and classes:

```
Type x Format [ x Target ]
```

Otherwise you get an untyped mess as in C:

```
printf ("%s", 123);
```

In the case of 'Image the dispatch is hard-wired. The compiler generates it according to one of built-in classes like 'integer type'. So yes it would be no problem to add parameters specific to each of the classes as well as common parameters like padding or alignment inside a field. But it will never ever happen.

You seem suggesting a class-wide parameter type instead:

```
type Format_Type is tagged record
   Width: Natural:= 0;
   Alignment: Alignment_Type:= Left;
   Padding: Character:= ' ';
end record;
type Integer_Format is new Format_Type
   with record
      Plus_Sign : Boolean := False;
      Base : Base_Type := 10;
   end record;
X'Image (Format => Format_Type'Class)
```

This still requires a change that will be outright rejected on highest philosophical grounds. (:-))

However with a Format_Type you do not need 'Image. You can simply use a binary operation, e.g.

```
function "/" (Value : Integer; Format :
   Integer_Format) return String;
```

So would do

```
Put_Line ("X=" & X / (Width=>10,
   Padding=>'0', Alignment=>Right));
```

instead of

```
Put_Line ("X=" & X'Image (Width=>10,
   Padding=>'0', Alignment=>Right));
```

Of course it must be generic, which kills all fun.

Ergo

1. Compiler magic is necessary because the language type system is too weak to express things like formatting.
2. No proposal however useful and reasonable will survive ARG because of #1.
3. Use a library that does the stuff. E.g.

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

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 20 Aug 2023 11:43:01 +0200

> Additionally, one ends up with a magic mechanism that only the compiler can use. That *never* is a good idea.

A better idea would be to improve the language to remove need in magic, but that is *never* a good idea either! (:-))

> Attributes do not allow named parameters outside a few special cases, and *never* allow reordering of parameters. Does that need to change, too?

Elementary! Attribute is just an alternative syntactic form of a subroutine call. There is no reason why attribute should be limited to look like FORTRAN IV! (:-))

> 'Image is [...] useless for float output [...]

Which is why Float 'Image must have parameters!

> Ada has few built-in facilities because it's primary purpose is to support the development of proper ADTs. Ease of writing is not a goal at all [...]

How is this related to the attribute 'Image lacking necessary parameters? Why is a generic function having such parameters OK, while 'Image with same parameters is not?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Aug 2023 18:11:19 -0500

Your #3 is the point of course. If a reasonable library can be written, you should use that. After all, the Ada philosophy is that it is suspicious to use any built-in types. Why then should it be less suspicious to use other things that are built-in??

The best approach for Ada going forward is to add things that make it easier to build good libraries (as in user-defined literals). And minimize magic.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Aug 2023 18:34:55 -0500

> A better idea would be to improve the language to remove need in magic, but that is *never* a good idea either! (:-))

No, I generally agree with this. We probably disagree on what would constitute an improvement, however. ;-)

> Elementary! Attribute is just an alternative syntactic form of a subroutine call. There is no reason why attribute should be limited to look like FORTRAN IV! (:-))

That turns out to be a bad idea. The reason people love attributes so much is that they don't have to worry about visibility -- they're always visible. That is not and cannot be true for subprograms.

For example, the reason that we don't allow user-defined attributes is that they would compromise portability. Since they're always visible, they could hide/make illegal attributes that are used in units (like generic units) that don't know anything about the additions. Moreover, not all attributes can be described as subprograms given Ada's current rules (reduction attributes have a type parameter; some of the annex 13 attributes have "any type" parameters, etc.)

It certainly would be a very bad thing for Janus/Ada, which would have to have its resolution and subprogram definition mechanisms redesigned. (All subprograms are materialized in the Janus/Ada symboltable, in particular for visibility management reasons, and that would not be possible for attributes. Resolution only works on materialized subprogram definitions.)

> Which is why Float 'Image must have parameters!

Which is why one shouldn't use Float'Image! ;-)

> Why generic function having such parameters is OK, while 'Image with same parameters is not?

It's perfectly OK to overload functions however one wants, because you can keep anything that is problem from being considered by avoiding "use" (and "with").

'Image is not appropriate for an attribute in the first place; attributes are supposed to be simple compile-time defined properties of a type. String conversion is not that.

My preference for making Ada easier to use for this sort of thing is to allow class-wide elementary types. Then one could have non-generic subprograms that operate on all integer and float types. (Fixed and enumerations would still require generics, although I suspect most people would simply convert fixed to float for output rather than worrying about an instantiation.) That would make a library simple to use, and few people would think that something built-in is needed.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Aug 2023 18:37:43 -0500

> Of course it must be generic, which kills all fun.

As noted in my other message, resurrecting the Ada 95 idea allowing class-wide types for elementary types would eliminate (or at least greatly reduce) this problem. I think that would be a more productive way to address this problem than hacking around with 'Image some more. (We've already proven that it is not a good way to define anything user-defined, thus the rather complex way to define such 'Image attributes.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 22 Aug 2023 09:38:38 +0200

> [...] allowing class-wide types for elementary types would eliminate (or at least greatly reduce) this problem.

Yes, but that would be a huge change.

> I think that would be a more productive way to address this problem than hacking around with 'Image some more.

One does not exclude another. If you allowed classes then there would be no reason not to have attributes [as] official primitive operations. E.g. an "imaginable" interface would provide "'Image" and the standard Integer would inherit from "imaginable"...

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 22 Aug 2023 10:13:47 +0200

> [...] not all attributes can be described as subprograms given Ada's current rules [...]

It is a primitive subprogram of some built-in class. The magic is not in the attribute, it is the class description. For magical classes overriding a primitive operation could look like

for <member-type>'<primitive-operation-name> use <subroutine-name>;

[...]

The problem is that whatever intention Ada designers had for attributes they also gave them the property of being a primitive operation where no user-defined class [is] allowed. This power steamrolls any "good" intentions.

Nobody loves the syntax T'Image (X) or X'Image! Give programmers X.Image and [<path-of-package-names-nobody-remembers>].Image (X) and they will forget about the attribute.

[...]

> Fixed and enumerations would still require generics

It would be interesting to play with the ways of constructing enumeration and fixed point classes. Both have static parameters, e.g. list of names in the case of enumeration. There might be a way to achieve static polymorphism without going full generic but also without turning the language into a C++ templates mess!

> That would make a library simple to use, and few people would think that something built-in is needed.

Absolutely. Ideally, everything must go into libraries.

From: Stephen Davies <joviangm@gmail.com>
Date: Wed, 23 Aug 2023 03:20:07 -0700

> Nobody loves the syntax T'Image (X) or X'Image!

I have no issue with the 'Image syntax.

Perhaps the formatting parameters could be restricted to T'Image(X) and not available for X'Image? Or, maybe the language should just add 'Trim_Image and 'Trim_Width and leave the advanced formatting to a library.

Actually, I think it might also be nice if Float'Trim_Image(X) returned a string that only used exponential notation for very large or very small values (which seems to be the default behaviour in Python). Different names would then be needed (Tidy_Image and Tidy_Width?).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 23 Aug 2023 18:16:10 +0200

> it might also be nice if Float'Trim_Image(X) returned a string that only used exponential notation for very large or very small values

To use the shortest representation for the given precision unless specified otherwise:

http://www.dmitry-kazakov.de/ada/strings_edit.htm#6

Ada 'Image attributes have "typographic quality" in plain contradiction to the claim being for debugging purposes. That is why the plus sign is always represented by a space and why floating-point representation is always selected even for exact zero and the way the exponent part is formatted. The typographic idea is to have *same looking* output. Note, even if the output is mathematically incorrect as in the case of floating-point numbers. 'Image considers precision and accuracy same, which is *always* wrong when dealing with floating-point numbers.

> Different names would then be needed (Tidy_Image and Tidy_Width?).

It takes several parameters to control the behavior in a reasonable way.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 28 Aug 2023 20:58:13 +0300

> There is no good reason why attributes should not have the same parameter syntax as subprograms and entry calls.

Yes in principle, but it is understandable that making this happen now could impact both the language definition and various implementations in non-trivial ways.

> Neither there is one why 'Image must be a non-overridable attribute.

In Ada 2022, 'Image is defined to call the new attribute 'Put_Image, which can be specified (ie. overridden) by the programmer for any type.

See http://www.ada-auth.org/standards/22rm/html/RM-4-10.html.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 28 Aug 2023 21:08:10 +0200

> it is understandable that making this happen now could impact both the language definition and various implementations

Compared to the useless and damaging sediments the language collects with each new release? (:-))

> See http://www.ada-auth.org/standards/22rm/html/RM-4-10.html.

Ah, thanks. I vaguely remembered that there was yet another ugly hack that does not really solve anything significant, but could not find it.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 6 Sep 2023 20:04:00 -0500

>Neither there is one why 'Image must be a non-overridable attribute

There actually is a good reason for this. Attributes have global visibility. So if you allowed overriding of attributes, then a with added or removed in a remote part of a program could silently change the behavior of code that has no knowledge of the change. That would be bad for "programming in the large". Note that Ada 95 was proven to have no such cases, and we've tried very hard to avoid them.

One could imagine adding rather severe restrictions to overriding of attributes to eliminate this problem (for instance, only allowing it for primitive operations of the type), but that would eliminate all real value of the feature (you can always use a primitive function and "use all" to get the same effect without any new features).

For 'Image specifically, the design of the attribute doesn't work well for composition (for Image for composite types), which is why Ada 2022 has a separate attribute that can be overridden similar to a stream attribute.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 7 Sep 2023 11:01:58 +0200

> [...] That would be bad for "programming in the large". [...]

Ah, but 'Image is for debugging only! (:-))

> One could imagine adding rather severe restrictions to overriding of attributes to eliminate this problem [...]

It must be a new type:

```
type My_Integer is new Integer;
for My_Integer'Image use Foo;
```

From: Rod Kay <rodakay5@gmail.com>
Date: Sat, 23 Sep 2023 20:00:26 +1000

I've been using 'Gnat.formatted_Output' which I've found quite useful. Unfortunately, it seems to be a little buggy with its formatting.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Mon, 25 Sep 2023 22:47:17 -0700

You can take a look at VSS's Virtual_String_Templates and Formatters, see
https://github.com/AdaCore/VSS/blob/master/source/text/vss-strings-templates.ads

https://github.com/AdaCore/VSS/blob/master/source/text/vss-strings-formatters.ads

and an example of its use

https://github.com/AdaCore/gnatdoc/blob/3e94448ac57270caf4b4502f208f78e1d51da2b2/source/gnatdoc-messages.adb#L130

UNAS by TRW

From: Chris Sparks <mrada442@gmail.com>
Subject: UNAS by TRW
Date: Tue, 22 Aug 2023 05:58:37 -0700
Newsgroups: comp.lang.ada

[In the following: Universal Network Architecture Services (UNAS) is a product from the American TRW Inc. corporation. –arm]

Does anyone know how to get the complete UNAS package from TRW? I use it at work and I see it has an open usage clause in the source headers. Since I am not allowed to download it from my work, maybe I can find a source elsewhere to get it?

Also are there any tutorials out there on how to use it? I am in the process of upgrading the Ada (83 to 05) in my current project and I am getting stuck on the plethora of calls being made by UNAS for which I don't even know how to set it up so it can run happily.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 24 Aug 2023 22:24:59 +0200

> maybe I can find a source elsewhere to get it?

Doubtful.

> Also are there any tutorials out there on how to use it?

"Out there", definitely not. In a few closed places that still use UNAS, perhaps but doubtful. In the one place that I know still uses UNAS, no.

> I am in the process of upgrading the Ada (83 to 05) in my current project [...]

Do I divine correctly that "your current project" is not "at work" If so I would suggest you consider PolyORB as a replacement*. UNAS is long dead, unmaintained and unmaintainable, mostly because it is proprietary software without anyone getting a license for it other than in their current application. Also, apart from a couple of people I know, nobody understands UNAS anymore. The company that made it has abandoned it, perhaps even gone bankrupt, so UNAS is mostly technical debt. Sorry for the bad "news".

* Modern multi-core computers with lots of memory might even make it feasible to avoid distributing the software over multiple computers in the first place. Maybe a monolithic application would do the job just fine, nowadays.

From: Chris Sparks <mrada442@gmail.com>
Date: Thu, 24 Aug 2023 16:36:00 -0700

I suspected as much. If I could only find something that would show me how to install it and be operational so I can finish my upgrade project. Going to a new software is something that would bring on risk, unless I could narrow down what exactly the UNAS is being used for. This effort I am working on is for the contract I am working on.

What would really help is documentation. Installation, operation so that I can tell that it is working.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Fri, 25 Aug 2023 04:13:54 +0200

> What would really help is documentation. Installation, operation so that I can tell that it is working.

If your customer has UNAS, they probably have documentation, or what passes as documentation.

BTW, UNAS is a framework for distributed applications i.e. multiple programs doing remote procedure calls and message passing over the network.

From: Chris Sparks <mrada442@gmail.com>
Date: Fri, 25 Aug 2023 17:41:09 -0700

Unfortunately they don't have any documentation as it was set up very long ago.

If I had it on my home PC I would have more time to look at it.

Project Euler 26

From: Csyh (Qaq) <schen309@asu.edu>
Subject: project euler 26
Date: Mon, 4 Sep 2023 02:19:51 -0700
Newsgroups: comp.lang.ada

I am new to Ada, I know is there a good way to start this program? Thanks

https://projecteuler.net/problem=26

[The problem is: Find the value of d<1000 for which 1/d contains the longest recurring cycle in its decimal fraction part. –arm]

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 4 Sep 2023 14:06:13 +0300

First invent/discover the method (algorithm) for solving the problem, without thinking about the programming language.

I don't think any language has built-in features that would lead to a direct solution, although some functional language with lazy evaluation could come close, because such languages can manipulate unbounded (potentially infinite) sequences of values. Such sequences can be handled in Ada, too, but with more effort -- they are not "built in" to Ada.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 4 Sep 2023 14:39:17 +0200

Infinite division does not require big numbers, which Ada 22 has, but I i would not use them anyway because the performance would be abysmal.

BTW, Ada is perfect for numeric algorithms no need to resort to functional mess... (:-))

The problem itself requires as you said mathematical analysis, because a naive method of comparing a partial division result with itself is obviously wrong. E.g. let you have 0.12341234... you could not conclude that the period is (1234) because it could actually be (123412345).

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Mon, 04 Sep 2023 17:01:04 +0100

> BTW, Ada is perfect for numeric algorithms no need to resort to functional mess... (:-))

Perfect? That's a bold claim!

Mind you, I don't think this problem is really a numerical one in that sense. It needs some simple integer arithmetic but then every language is perfect for that sort of arithmetic.

Using a functional mess (Haskell) a simple, native solution (i.e. using no modules) is only 9 lines long.

I don't want to start a language war. Ada is just more 'wordy' by deliberate design so a simple Ada solution is inevitably going to be longer in terms of lines. Rather my purpose in posting is to steer the OP away from thinking of this as a numerical problem in the classical sense. It really isn't.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 4 Sep 2023 21:20:56 +0200

[...]

> Using a functional mess (Haskell) a simple, native solution (i.e. using no modules) is only 9 lines long.

Apart from the fundamental inconsistency of functional paradigm: computing is about transition of states and nothing else; the imperative languages express solutions, i.e. an algorithm. Functional, and in general, declarative languages express puzzles.

They remind me of math examination tasks on studying a function. Here is a definition. Go figure out the properties and behavior...

Or, if you want, functional is like a chess composition: white to move and checkmate in 4 moves. Challenging, but Ada is about playing chess.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Mon, 04 Sep 2023 21:18:16 +0100

> Apart from the fundamental inconsistency of functional paradigm [...]

Rather than try to unpick that paragraph I'll just say that they can, none the less, give simple solutions to this sort of programming problem.

From: Francesc Rocher <francesc.rocher@gmail.com>
Date: Thu, 7 Sep 2023 00:31:09 -0700

> I am new to Ada, I know is there a good way to start this program?

Please take a look at my Euler tools repository, https://github.com/rocher/euler_tools (not the best math lib you'll find, I know).

I used this library tools to solve problem 26 here: https://github.com/rocher/alice-project_euler-rocher

Let me know what you think.

Equivalence between Named Anonymous Access

From: Blady <p.p11@orange.fr>
Subject: Equivalence between named access and anonymous access.
Date: Wed, 6 Sep 2023 16:37:08 +0200
Newsgroups: comp.lang.ada

I'm wondering about named access and anonymous access. In the following Ada code, are the writing of parameter P1 type of procedures PA and PB equivalent?

```
package C1 is
   type Inst is tagged null record;
   type Class is access all Inst'Class;
end C1;
```

```
with C1;
  package C2 is
    type Inst is tagged null record;
    type Class is access all Inst'Class;


    procedure PA (Self : Inst;
       P1 : C1.Class); -- named access
    procedure PB (Self : Inst; P1 : access
       C1.Inst'Class); -- anonymous access
  end C2;
```

Same with:

```
  function FA (Self : Inst) return C1.Class;
  -- named access
  function FB (Self : Inst) return access
     C1.Inst'Class; -- anonymous access
```

Are FA and FB writing equivalent?

If not, why?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Wed, 6 Sep 2023 17:54:42 +0200

They are not equivalent from the access checks point of view:

```
   declare
     Y : C2.Inst;
     X : aliased C1.Inst;
   begin
     C2.PA (Y, X'Access);
     -- Non-local pointer error
     C2.PB (Y, X'Access); -- Fine
   end;
```

Furthermore, tagged anonymous access is controlling (dispatches) when not class-wide.

From: Gautier Write-Only Address
<gautier_niouzes@hotmail.com>
Date: Wed, 6 Sep 2023 13:55:02 -0700

> In the following Ada code, are the writing of parameter P1 type of procedures PA and PB equivalent?

They are not equivalent because the anonymous access opens more possibilities (example below), but you are certainly aware of that.

So I guess you have another question in mind...

```
with C1, C2;
procedure test is
  x2 : C2.Inst;
  type My_Reference_1 is access all
     C1.Inst'Class;
  r1 : My_Reference_1;
begin
  x2.PB (r1);
  x2.PA (r1);
  -- ^ expected type "Class" defined at
  -- c1.ads:3 found type "My_Reference_1"
  -- defined at line 6
end;
```

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 7 Sep 2023 02:20:02 +0200

> I'm wondering about named access and anonymous access.

The rules for using access-to-object types are:

1. Don't use access types
2. If you think you should use access types, see rule 1.
3. If you still think you should use access types, don't use anonymous access types
4. If you still think you should use anonymous access types, don't develop software

The semantics of named access types are well defined and easily understood. The semantics of anonymous access types are defined in ARM 3.10.2, of which the AARM says
"Subclause 3.10.2, home of the accessibility rules, is informally known as the 'Heart of Darkness' amongst the maintainers of Ada. Woe unto all who enter here (well, at least unto anyone that needs to understand any of these rules)."

The ARG freely admits that no one understands 3.10.2, which means that what you get when you use anonymous access types is whatever the compiler writer thinks it says. This may differ between compilers and between different versions of the same compiler, and from what you think it says.

So no sane person uses them.

From: Blady <p.p11@orange.fr>
Date: Thu, 7 Sep 2023 18:06:15 +0200

Thanks Dmitry, also Gautier and Jeff for your previous answers.

Well, I was questioning myself about the choice between named access and anonymous access in the old Ada port of Java library, for instance:

```
   type Typ;
   type Ref is access all Typ'Class;
   type Typ(LayoutManager2_I :
     Java.Awt.LayoutManager2.Ref;
     Serializable_I : Java.Io.Serializable.Ref)
    is new Java.Lang.Object.Typ
     with null record;
   ---------------------------------------
   -- Constructor Declarations --
   ---------------------------------------
   function New_BorderLayout
      (This : Ref := null) return Ref;
   function New_BorderLayout
      (P1_Int : Java.Int;
       P2_Int : Java.Int;
       This : Ref := null) return Ref;
   --------------------------------
   -- Method Declarations --
   --------------------------------
   procedure AddLayoutComponent
     (This : access Typ;
      P1_Component : access
         Standard.Java.Awt.
         Component.Typ'Class;
      P2_Object : access
          Standard.Java.Lang.
          Object.Typ'Class);
   function GetLayoutComponent
     (This : access Typ;
      P1_Object : access
          Standard.Java.Lang.
          Object.Typ'Class)
      return access Java.Awt.Component.
        Typ'Class;
```

Why choose named access for New_BorderLayout and anonymous access for AddLayoutComponent or GetLayoutComponent for the type of parameters P1_xxx and the return type?

Why not all named or all anonymous?

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

Date: Thu, 7 Sep 2023 18:18:11 +0200

> Why choosing named access for New_BorderLayout and anonymous access for AddLayoutComponent

It's very poor design to have access types in the visible part of a non-private pkg spec.

From: Blady <p.p11@orange.fr>
Date: Thu, 7 Sep 2023 21:10:12 +0200

> It's very poor design to have access types in the visible part of a non-private pkg spec.

I got your point :-)

But, in this specific case, I was wondering why not writing all with named access or all with anonymous access?

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Thu, 7 Sep 2023 22:23:37 +0200

> Well, I was questioning myself about the choice between named access and anonymous access in the old Ada port of Java library [...]

Contravariance is unsafe. I gather that Typ is tagged. If you ever derive from it, it will "inherit" the broken construction function, because the function is class-wide. The safe choice here is anonymous access. The compiler will require to override the construction function. That is for the return value. The case for the argument depends. Again anonymous access type is handier but if you going to copy/store references, then named types are better.

> Why not all named or all anonymous?

My rough rule is like this:

Do not expose access types if you can.

If you successfully hide them either completely or by declaring them private, then named they go.

If you expose access types, then anonymous access is usually a better choice because it is easier to use, especially when access is merely to work around language limitations on argument/result passing (unconstrained object, access rules nightmare) AKA

closures. Then it is much safer in a hierarchy of types and it is more use-clause friendly.

Is Gnoga Dead?

From: Reinert <reinkor@gmail.com>
Subject: Is gnoga dead?
Date: Sat, 9 Sep 2023 02:02:46 -0700
Newsgroups: comp.lang.ada

I asked google bard:
"Is gnoga currently maintained?"

And it answered:

"No, Gnoga (Ada) is also no longer maintained. It was a fork of the original Gnoga project that was created in 2012. The last commit to the GitHub repository was in 2016.

Gnoga (Ada) was an attempt to improve the performance and scalability of Gnoga. It used a different statistical approach to NLG and was implemented in Ada.

There are a number of other NLG engines available, such as GPT-3 and LaMDA, which are more powerful and up-to-date than Gnoga (Ada)."

Is this correct?

From: Blady <p.p11@orange.fr>
Date: Sat, 9 Sep 2023 11:53:36 +0200

Do you mean this one:
https://sourceforge.net/projects/gnoga?

If so, the answer is yes [it is maintained –arm].

Could you please provide the link where you found the above quoted text?

From: Reinert <reinkor@gmail.com>
Date: Sat, 9 Sep 2023 04:47:41 -0700

I just logged into my google account and asked "bard" (https://bard.google.com/?hl=en).

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sat, 9 Sep 2023 13:59:00 +0200

> I asked google bard:

> [...]

> Is this correct?

The presence of the word "also" in the first sentence should be enough to tell you that this is nonsense.

If you are asking about an "NLG engine" named gnoga, then maybe this is correct. If you are asking about the Ada web-application framework Gnoga that Blady maintains, then it's not correct.

From: Reinert <reinkor@gmail.com>
Date: Sat, 9 Sep 2023 09:46:26 -0700

Good to hear.

Yes, did mean the Ada web-application framework.

From: Stéphane Rivière <stef@genesix.org>
Date: Sat, 9 Sep 2023 19:55:56 +0200

Gnoga is not dead, being maintained by Pascal, Gautier and other individuals and companies.

Expect a pleasant surprise (imho) before the end of the year (with full web demos). v22 manual abstract:

1 About v22 framework
1.1 Ready to use in production
v22 is a general purpose, KISS oriented, modular Ada framework for GNU/Linux Debian/Ubuntu service, console and web programs.

v22 is composed of many packages in charge of UTF-8 strings, program and OS functions, HTTP(s)/WS(s) web framework, integrated cURL, console handling and text files, advanced network, MySQL and SQLite high level binding, logging and configuration files handling.

Although based on the v20 library, the v22 framework represents a major step forward in the following areas:

- UTF-8 compatibility;
- Simplified string processing (only one UTF-8 String type is used);
- Internationalization;
- New and extended database API;
- Extended database access to MySQL, in addition to SQLite, with schema on-the-fly update at table, index, and column level;
- Improved concurrent access and performance for SQLite;
- New LGPLv3 licensing instead of GPLv3;
- New FSF GNAT GCC Linux ready-to-use development environment for v22 (not tied anymore to GPLv2 license);
- And much more.

1.2 Cooperative and open

v22's native dependencies are Gnoga, Simple_Components, UXStrings and Zanyblue.

v22 is both a high-level framework and an extension to the lower level components cited above. v22 has been designed to:

- Use unmodified components;
- Not "reinvent the wheel". Component functions are to be used first;
- Offer higher-level functions or functions that do not exist in the components.

.../...

In short:

- UXStrings is used throughout v22. The v22.Uxs package extends UXStrings functionality. The v22.Sql package extends the functionality of Gnoga.Server.Database. The v22.Gui graphics framework is based on Gnoga.Gui;
- v22's architecture allows it to be open to additional packages, depending on the software development required.

From: Reinert <reinkor@gmail.com>
Date: Sat, 9 Sep 2023 23:51:12 -0700

Sounds like I can somehow trust that gnoga will be around for many years to come.

So my special issue: I work on making my (cancer) cellular behavior analysis program (https://korsnesbiocomputing.no) as a "cloud service". It's all programmed in Ada using GLOBE_3D. I am considering using guacamole apache. It's intensive about handling images. So what are the arguments for and against using guacamole (as compared to for example guacamole apache)?

From: Stéphane Rivière <stef@genesix.org>
Date: Mon, 11 Sep 2023 09:52:45 +0200

> Sounds like I can somehow trust that gnoga will be around many years to come.

I think so.

> So my special issue: I work on making my (cancer) cellular behavior analysis program (https://korsnesbiocomputing.no) as a "cloud service".

Very interesting indeed.

> [...] So what are the arguments for and against using guacamole (as compared to for example guacamole apache)?

No idea. My first concerns could be scaling.

From: Reinert <reinkor@gmail.com>
Date: Mon, 11 Sep 2023 23:00:45 -0700

Did mean ..... *gnoga* (as compared to, for example, guacamole apache) :-)

From: Stéphane Rivière <stef@genesix.org>
Date: Tue, 12 Sep 2023 09:16:52 +0200

> Did mean ..... *gnoga* (as compared to for example guacamole apache) :-)

No, sorry, I was thinking more of Guacamole. Not really fond of a remote desktop vs a true Web app... But maybe I'm wrong...

Aggregate with Derived Types

From: Blady <p.p11@orange.fr>
Subject: Aggregate with derived types.
Date: Thu, 14 Sep 2023 16:02:39 +0200
Newsgroups: comp.lang.ada

I want to extend a container type like Vectors, I've written:

```
type My_Float_List2 is new
   My_Float_Lists.Vector with null record;
```

But the initialization gives an error line 6:

```
1. with Ada.Containers.Vectors;
2. with Ada.Text_IO;
3. procedure
 test_20230914_derived_agg is
4.    package My_Float_Lists is new
Ada.Containers.Vectors (Positive, Float);
5.    subtype My_Float_List1 is
My_Float_Lists.Vector;
6.    type My_Float_List2 is new
My_Float_Lists.Vector with null record;
7.    ML1 : My_Float_List1 := [-3.1, -6.7,
3.3, -3.14, 0.0];
8.    ML2 : My_Float_List2 := ([-3.1, -6.7,
3.3, -3.14, 0.0] with null record);  |
>>> error: no unique type for this
aggregate
9. begin
10.    Ada.Text_IO.Put_Line
(ML1.Element (3)'Image);
11.    Ada.Text_IO.Put_Line
(ML2.Element (3)'Image);
12. end test_20230914_derived_agg;
```

The RM says:

4.3.2 Extension Aggregates

1 [An extension_aggregate specifies a value for a type that is a record extension by specifying a value or subtype for an ancestor of the type, followed by associations for any components not determined by the ancestor_part.]

Language Design Principles

1.a The model underlying this syntax is that a record extension can also be viewed as a regular record type with an ancestor "prefix".

The record_component_association_list corresponds to exactly what would be needed if there were no ancestor/prefix type. The ancestor_part determines the value of the ancestor/prefix.

Syntax
2 extension_aggregate ::=
(ancestor_part with
record_component_association_list)
3 ancestor_part ::= expression |
subtype_mark

It is not so clear for me what a unique type could be? Any clue?

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 14 Sep 2023 17:31:47 +0200

IIUC, you have to qualify the value:

```
(My_Float_List1'[-3.1, -6.7, 3.3, -3.14, 0.0]
   with null record)
```

or

```
(My_Float_Lists.Vector'[-3.1, -6.7, 3.3,
    -3.14, 0.0] with null record)
```

(not tested)

From: Blady <p.p11@orange.fr>
Date: Thu, 14 Sep 2023 22:00:19 +0200

Thanks Jeff, both proposals are compiled ok by GNAT.

I wonder why the float list aggregate isn't inferred by the compiler and need some help with a qualification.

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 14 Sep 2023 23:37:47 +0200

I'm not sure. But can't you simply write

```
ML2 : My_Float_List2 := [-3.1, -6.7, 3.3,
   -3.14, 0.0];
```

? I presume that My_Float_List2 inherits its aggregate definition from My_Float_List1.

From: Blady <p.p11@orange.fr>
Date: Fri, 15 Sep 2023 09:27:57 +0200

Unfortunately not directly:

```
10.    ML2c : My_Float_List2 := [-3.1, -6.7,
3.3, -3.14, 0.0];                    |
>>> error: type of aggregate has private
ancestor "Vector"
>>> error: must use extension aggregate
```

Shouldn't it inherit them?

Indeed you have it if you defined a private extension with explicit aspects:

```
package PA is
   type My_Float_List3 is new
      My_Float_Lists.Vector with private
      with
         Constant_Indexing =>
            Constant_Reference,
         Variable_Indexing => Reference,
         Default_Iterator => Iterate,
         Iterator_Element  => Float,
         Aggregate =>
            (Empty => Empty,
             Add_Unnamed => Append,
             New_Indexed => New_Vector,
             Assign_Indexed =>
                Replace_Element);
   function Constant_Reference
     (Container : aliased My_Float_List3;
      Index : Positive) return
        My_Float_Lists.
        Constant_Reference_Type is
       (My_Float_Lists.Constant_Reference
          (My_Float_Lists.Vector
             (Container), Index));
   function Reference (Container : aliased
      in out My_Float_List3;
      Index : Positive) return
         My_Float_Lists.Reference_Type is
     (My_Float_Lists.Reference
        (My_Float_Lists.Vector (Container),
         Index));
private
   type My_Float_List3 is new
      My_Float_Lists.Vector with
      null record;
end PA;
ML3 : PA.My_Float_List3 := [-3.1, -6.7,
   3.3, -3.14, 0.0];
```

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Sat, 16 Sep 2023 01:39:57 -0500

> I wonder why the float list aggregate isn't inferred by the compiler and need some help with a qualification.

The language rule is that the ancestor_part of an extension aggregate is expected to be of "any tagged type" (see 4.3.2(4/2)). An aggregate needs to have a single specific type, and "any tagged type" is not that.

The reason that the ancestor is "any tagged type" is that the type of the ancestor determines the extension components needed along with other legality rules. One could imagine a language where all of these things are decided simultaneously, but people worried that the complexity would make it difficult/impossible to implement. So aggregates are essentially black boxes whose type has to be determinable from the outside, and similar rules exist for parts inside the aggregate.

Project Euler 29

From: Csyh (Qaq) <schen309@asu.edu>
Subject: project euler 29
Date: Fri, 15 Sep 2023 02:03:16 -0700
Newsgroups: comp.lang.ada

Now this time, I am facing trouble for problem #29.

[How many *distinct* terms are in the sequence (for a in 2 .. 100 => (for b in 2 .. 100 => a**b))? –arm]

As I know integer type is for 32 bits. but for this problem as me to find out the 2 ** 100 and even 100 ** 100.

I used Python to get the answer correctly in 5 minutes.

```
context = []
for a in range(2,101):
    for b in range(2,101):
        context.append(a**b)
len(list(set(context)))
```

I know the algorithm is easy, but I am pretty interested in how to calculate a large [?] like it. And thanks for the help from problem 26, your discussions come to me every working hour.

For this problem I want to know how to know is there an easy way to store a large number like 100 ** 100, and how do you make a similar function like "set(context)" to delete the duplicate value in a vector.

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 15 Sep 2023 11:50:42 +0200

> for this problem I want to know how to know is there an easy way to store a large number like 100 ** 100, and how do U make a similar function like "set(context)" to delete the duplicated value in a vector.

You will need an unbounded-integer pkg. If you want to write portable code in a standard language, then you can write Ada 12 using a library such as PragmARC.Unbounded_Numbers. Integers

(https://github.com/jrcarter/PragmARC/blob/Ada-12/pragmarc-unbounded_numbers-integers.ads). This will compile with both GNAT and ObjectAda.

If you want to write non-portable code in a non-standard, Ada-like language, then you can use the GNAT language, which is mostly Ada 12 with some Ada 23 features, one of which is the Ada-23 standard package Ada.Numerics.Big_Numbers.Big_Integers (http://www.ada-auth.org/standards/22aarm/html/AA-A-5-6.html). This can only be compiled with GNAT. Note that, unlike PragmARC.Unbounded_Numbers.Integers, GNAT's implementation of Ada.Numerics.Big_Numbers.Big_Integers is not truly unbounded. I don't know if it will hold 101 ** 101 without modification.

You can store the results directly in a set from the standard library to avoid duplicate values. If I understand your Python (probably not), you would want to output the result of Length for the resulting set.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Fri, 15 Sep 2023 16:42:38 +0100

> I know the algorithm is easy [...]

Most of the Project Euler problems have solutions that are not always the obvious one (though sometimes the obvious one is the best). You can, of course, just use a big number type (or write your own!) but this problem can be solved without having to use any large numbers at all.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 15 Sep 2023 18:34:21 +0200

> As I know integer type is for 32 bits [...]

I missed this the first time.

No, you don't know that Integer is 32 bits. ARM 3.5.4 (21) [http://www.ada-auth.org/standards/aarm12_w_tc1/html/AA-3-5-4.html] requires "In an implementation, the range of Integer shall include the range –2**15+1 .. +2**15–1."

There are compilers for which Integer is less than 32 bits, so assuming otherwise is not portable. I know a lot of people don't care about portability, but I've also seen projects that spent large sums porting code that they thought didn't have to be portable. The cost of writing portable code is usually much smaller than the cost of porting non-portable code.

Of course, you can always declare your own integer type with whatever range is appropriate for your problem, though the compiler doesn't always have to accept it. I don't know of any compiler that doesn't accept 32-bit integer declarations, nor any targeting 64-bit platforms that doesn't accept 64-bit integers. But you're unlikely to find a compiler that will accept range 2 .. 101 ** 101

In King (https://github.com/jrcarter/King) the compiler must accept all integer type declarations.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Fri, 15 Sep 2023 11:04:27 -0700

> I don't know if it will hold 101 ** 101 without modification.

It only has to hold 100 ** 100. The Python code in the parent uses the expression `range(2,101)`. Python's range() function yields a range that includes the first bound and excludes the second bound.

From: Francesc Rocher <francesc.rocher@gmail.com>
Date: Sat, 16 Sep 2023 03:07:06 -0700

Please take a look at this solution:

https://github.com/rocher/alice-project_euler-rocher/blob/main/src/0001-0100/p0029_distinct_powers.adb

It's not using any big numbers library.

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 17 Sep 2023 15:54:12 -0700

> Also, do you have a different approach to solve this 29th problem?

I see two natural approaches: 1) use bignums--it didn't occur to me to not use them until this discussion. 2) Notice that a**b == c**d exactly when the two sides have the same prime factorization, and the factors of a**b are just the factors of a repeated b times, so you can count up the distinct tuples of factors.

Method #2 is efficient (since a,b,c,d are all < 100) and doesn't use bignums, but it is a fair amount of code to write unless you have convenient libraries at hand for factorization and can easily count sets of distinct tuples. I guess there are fancier approaches possible too, that avoid searching 100**2 combinations, but 100**2 is just 10000 which is small.

Certainly both are easier to do if your language or libraries has convenient features for dealing with variable sized objects like bignums, or sets of tuples. The bignum approach is less efficient but it is much easier to code. The Python expression

```
len(set(a**b for a in range(2,101) for b in range(2,101)))
```

takes around 25 msec to compute on my old slow laptop.

I will look at your Ada solution!

From: Paul Rubin <no.email@nospam.invalid>
Date: Sun, 17 Sep 2023 17:09:38 -0700

>> Also, do you have a different approach to solve this 29th problem?

> Yes, but it's not in Ada. I implemented an equality test for a^b == c^d.

Oh interesting, based on a comment in Francesc's code, I think I see a method to do it without the auxiliary array, at a small increase in runtime cost. Basically given a and b, you can find their prime factors and easily enumerate the combinations x,y with a**b==x**y and 1 <= x,y <= 100. You can label each "equivalence class" by the (a,b) with the smallest possible a.

So you just loop through 1 <= a,b <= 100 and count only the a,b pairs where a is the smallest a for its equivalence class. I might see if I can code this, which should also let me describe it more concisely.

From: Ben Bacarisse <ben.usenet@bsb.me.uk>
Date: Mon, 18 Sep 2023 01:16:19 +0100

> So you just loop through 1 <= a,b <= 100 and count only the a,b pairs where a is the smallest a for its equivalence class.

This is likely to be fast which is why I wanted to compile Francesc's to try it out. Mind you, a naive a^b == c^d test gives pretty good performance for the kind of range requested.

Get Character and Trailing New Lines

From: Blady <p.p11@orange.fr>
Subject: Weird behavior of Get character with trailing new lines.
Date: Fri, 22 Sep 2023 21:30:15 +0200
Newsgroups: comp.lang.ada

I'm reading a text file with Get character from Text_IO with a while loop controlled by End_Of_File.

% cat test_20230922_get_char.adb

```
with Ada.Text_IO; use Ada.Text_IO;
procedure test_20230922_get_char is
   procedure Get is
      F : File_Type;
      Ch : Character;
   begin
      Open (F, In_File,
         "test_20230922_get_char.adb");
      while not End_Of_File(F) loop
        Get (F, Ch);
        Put (Ch);
      end loop;
      Close (F);
      Put_Line ("File read with get.");
   end;
begin
Get;
end;
```

All will be well, unfortunately not!

Despite the End_Of_File, I got an END_ERROR exception when there are

several trailing new lines at the end of the text:

```
[...] Execution of
../bin/test_20230922_get_char terminated by
unhandled exception
raised ADA.IO_EXCEPTIONS.END_ERROR
: a-textio.adb:517
```

The code is compiled with GNAT, does it comply with the standard?

A.10.7 Input-Output of Characters and Strings

For an item of type Character the following procedures are provided:

```
procedure Get(File : in File_Type;
   Item : out Character);
procedure Get(Item : out Character);
```

After skipping any line terminators and any page terminators, reads the next character from the specified input file and returns the value of this character in the out parameter Item. The exception End_Error is propagated if an attempt is made to skip a file terminator.

This seems to be the case, then how to avoid the exception?

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Fri, 22 Sep 2023 22:52:21 +0300

In Text_IO, a line terminator is not an ordinary character, so you must handle it separately, for example like this:

```
while not End_Of_File(F) loop
   if End_Of_Line(F) then
      New_Line;
      Skip_Line(F);
   else
      Get (F, Ch);
      Put (Ch);
   end if;
```

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 22 Sep 2023 22:05:55 +0200

As you have quoted, Get (Character) skips line terminators. End_Of_File returns True if there is a single line terminator before the file terminator, but False if there are multiple line terminators before the file terminator. So you either have to explicitly skip line terminators, or handle End_Error.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 23 Sep 2023 09:02:37 +0200

And this works only if the input file is "well formed", i.e. if it has line terminators as the compiler expects them to be (f.e., you will be in trouble if the last line has no LF). That's why I never check End_Of_File, but handle the End_Error exception. It always works.

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Sat, 23 Sep 2023 11:39:25 +0300

Hm. The code I suggested, which handles line terminators separately, does work without raising End_Error even if the last line has no line terminator, at least in the context of the OP's program.

> That's why I never check End_Of_File, but handle the End_Error exception. It always works.

True, but it may not be convenient for the overall logic of the program that reads the file. That program often wants to do something with the contents, after reading the whole file, and having to enter that part of the program through an exception does complicate the code a little.

On the other hand, past posts on this issue say that using End_Error instead of the End_Of_File function is faster, probably because the Text_IO code that implements Get cannot know that the program has already checked for End_Of_File, so Get has to check for that case anyway, redundantly.

My usual method for reading text files is to use Text_IO.Get_Line, and (I admit) usually with End_Error termination.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sat, 23 Sep 2023 11:25:05 +0200

> [...] having to enter that part of the program through an exception does complicate the code a little.

It rather simplifies the code. You exit the loop and do whatever is necessary there.

Testing for the file end is unreliable and non-portable. Many types of files simply do not support that test. In other cases the test is not file immutable with the side effects that can change the program logic.

It is well advised to never ever use it.

From: Blady <p.p11@orange.fr>
Date: Mon, 25 Sep 2023 21:55:56 +0200

Thanks all for your helpful answers.It actually helps.

Especially, I was not aware of the particular behavior of End_Of_File with a single line terminator before the file terminator.

In my case, I prefer to reserve exceptions for exceptional situations :-) so I've taken the code from Niklas' example.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Tue, 26 Sep 2023 00:53:53 -0500

> And this works only if the input file is "well formed"

Agreed. And if the file might contain a page terminator, things get even worse because you would have to mess around with End_of_Page in order to avoid hitting a combination that still will raise End_Error. It's not worth the mental energy to avoid it, especially in a program that will be used by others. (I've sometimes used the simplest possible way to writing a "quick&dirty" program for my own use; for such programs I skip the error handling as I figure I can figure out what I did wrong by looking at the exception raised. But that's often a bad idea even in that case as such programs have a tendency to get reused years later and then the intended usage often isn't clear.)

'Valid Attribute and Input Operations

From: Maciej Sobczak
<see.my.homepage@gmail.com>
Subject: Valid attribute and input operations
Date: Sat, 23 Sep 2023 13:22:09 -0700
Newsgroups: comp.lang.ada

I am in the middle of a heated debate with Richard Riehle on LinkedIn, where we cannot get to terms with regard to the exact semantics of X'Valid in the context of input operations performed by standard Get procedure.

In short, consider the following example:

```
with Ada.Text_IO;
with Ada.Integer_Text_IO;
procedure Is_Valid_Test is
  X : Integer range 0..200;
begin
  Ada.Text_IO.Put("Get an Integer: ");
  Ada.Integer_Text_IO.Get(X);
  if X'Valid then
    Ada.Text_IO.Put_Line
      ("The Input is Valid ");
  else
    Ada.Text_IO.Put_Line
      ("The Input is not Valid ");
  end if;
end Is_Valid_Test;
```

When the input is 500, what should be the behavior of this program?

There are two interpretations:

1. Get is an input operation and can create invalid representations (as stated in 13.9.2, p.7). Then, the X'Valid test that follows Get(X) can be used to safely recognize whether the value is in the range or not. The program should print the second string (from the else branch), but should not raise any exceptions for this input (500).
2. Get is not an input operation in the meaning referred to in 13.9.2/7, or is an input, but only for type Integer (and it cannot create invalid integer representations on typical computers anyway). The X variable is an actual parameter that has a subtype that is different from the formal parameter and is subject to conversions when the Get subprogram exits normally (6.4.1/17,17a). This conversion should raise Constraint_Error for this input (500).

I have checked the above program on several on-line compilers, all of them behave according to interpretation 2 above.

Richard claims to get behavior 1 on his compiler.

What is your take on this? Any language lawyers?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sat, 23 Sep 2023 23:48:49 +0200

> What is your take on this? Any language lawyers?

The important thing is the definition of Ada.Text_IO.Integer_IO.Get [ARM A.10.8(7-10)]:

"... skips any leading blanks, line terminators, or page terminators, then reads a plus sign if present or (for a signed type only) a minus sign if present, then reads the longest possible sequence of characters matching the syntax of a numeric literal without a point. ...

"Returns, in the parameter Item, the value of type Num that corresponds to the sequence input.

"The exception Data_Error is propagated if the sequence of characters read does not form a legal integer literal or if the value obtained is not of the subtype Num."

So a call to Get can only return a valid value of type Num (Integer for your case) or raise Data_Error.

If Get is reading "500" then that certainly represents a valid value of type Integer, and Get should copy that back to the actual parameter.

If you are using Ada (a language with run-time checks), then a check should be made that the value is in the range of the actual parameter's subtype, here Integer range 0 .. 200. That should fail and Constraint_Error should be raised.

However, if you are not using Ada because that check has been suppressed, then the actual parameter will be left with the invalid value 500 and Constraint_Error will not be raised.

If I build your program with checks enabled, I get Constraint_Error. If I build it with checks suppressed, I get the not-valid message (GNAT 12.3).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 26 Sep 2023 01:13:53 -0500

I believe Jeffrey's analysis is correct.

Note that there are some special cases for validity that are intended to make it easier to write code like that you have. But they only make sense for base subtypes (and the type you have is not that). Moreover, they are not foolproof -- execution is not erroneous in these cases, but they still are a bounded error, and it is always correct for a bounded error to be detected and raise Program_Error.

This can happen in practice, too. For instance, for Janus/Ada, enumeration types with specified representations operate internally on the position numbers, and thus reading an enumeration variable will convert the representation to a position number with a table lookup. If the lookup fails, Program_Error is raised, and that happens before the value ever can be assigned to a named variable (and thus before any possible test of validity). I believe that we identified other similar cases back in the day. Probably one of them is the signalling NaN. Some bit patterns for float values represent signalling NaNs, which trap instantly if read. That's at the hardware level on most processors, so the only hope is to handle the resulting exception. It's too late by the time you get to 'Valid.

Moral: to make truly bulletproof code, you have to handle exceptions AND use 'Valid. You probably can skip the exceptions if everything is typed with integer basetypes, but if any other kinds of types are involved, they are necessary.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Tue, 26 Sep 2023 10:22:14 +0300

> ... for Janus/Ada, enumeration types with specified representations operate internally on the position numbers

Hm, that's interesting. Is that also the representation for record components of such an enumerated type?

For example, if I have:

```
type Command is (Off, On) with Size
   => 4;
for Command use (Off => 2, On => 5);
type Two_Commands is record
  C1, C2: Command;
end record
with Pack, Size => 8;
TwoC : Two_Commands :=
   (C1 => On, C2 => Off);
```

will the record components (in memory) have the values C1 = 1 and C2 = 0 (position numbers) or C1 = 5, C2 = 2 (specified representation)?

if they are represented by position numbers in the record, many if not most of my embedded Ada programs would fail if compiled with Janus/Ada, because the record values stored in I/O control registers or accessed via DMA would be wrong.

Damn, I thought those programs were not so compiler-dependent :-(

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 27 Sep 2023 22:27:41 -0500

No, the specified representation is always used when storing to memory (with the single exception of loop parameters, which cannot have address clauses or other representation specifications). I think even enum parameters are written in the representation. However, any time an enumeration value is read into a register it is converted to a position number. Usually, such values are used in indexing, comparing, or an attribute like 'Pos or 'Succ, all of which are defined to work on position numbers. But if you simply assign the value out again, it will get converted both ways. We do have an optimization to remove pairs of TOREP/DEREP, but not the reverse since Program_Error is a possibility from DEREP. (Well, unless unsafe optimizations are on, but I don't recommend using those for the obvious reasons.)

Should Light Runtimes Get More Consideration?

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Subject: Should light runtimes get more consideration?
Date: Tue, 26 Sep 2023 11:44:02 -0000
Newsgroups: comp.lang.ada

I created the issue below a little while ago. Today I wonder whether Ada 2022s 'Image attribute on records use of Unbounded strings is for good reason. Is it an oversight that Bounded String would work with the new light runtime or String with all runtimes including the older zero footprint runtimes?

Perhaps it was decided that a light runtime would not use this feature? And I can certainly avoid it. However I use a light runtime with 100s of kilobytes or RAM and many gigabytes of flash.

Ada is a much nicer language than Rust which uses unsafe all over for embedded but one thing that is interesting is that I believe all Rust code can be run easily on any target. Should Ada aspire to that?

On the other hand, micros are becoming multiprocessors bringing more demand for tasking (protected types are not compatible with a light runtime) but personally I believe multi chip single core designs are far better than multicore and not only due to the impossibility of side channel attacks like Spectre.

https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues/67

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 27 Sep 2023 22:48:16 -0500

As noted on the ARG Github, you confused the Unbounded version of Text_Buffers with an unbounded string (completely unrelated things), and

moreover, failed to notice that the language provides multiple ways to use a Bounded Text_Buffer instead. So the language addresses this particular concern.

I don't know if GNAT implements all of those ways (in particular, the restriction Max_Image_Length), but that is hardly the fault of the language!

For anyone else interested in this particular discussion, I recommend reading and following up on the ARG Githib issue rather than here (https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues/67).

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Thu, 28 Sep 2023 09:46:23 -0000

> Bounded Text_Buffer instead. So the language addresses this particular concern.

> I don't know if GNAT implements all of those ways [...]

I see. I guess the error message could suggest those options, too. Perhaps after the 2022 GNAT support work is completed.

That buffer support is pretty neat but my main concern, which GNAT may (it may not) address more than the current language by providing a cortex runtime, is that such demanding runtimes are brilliant but I am not sure if even Ravenscar is scalable to so many microchips such as Rust is trying to support. That isn't a huge issue but barriers to entry like having to work out your own exception replacement might be turning users away. Which is unfortunate when Ada is the best language out there by a significant margin for embedded development or frankly any protocol or hardware register use.

Of course others will rightly argue Ada is the best due to many of the more complex runtime features but that doesn't help with the issue of ease of adoption on an unsupported microchip that I have raised above.

From: Simon Wright <simon@pushface.org>
Date: Thu, 28 Sep 2023 14:25:18 +0100

When I started on Cortex GNAT RTS [1], a large part of the motivation (aside from the fun element) was that AdaCore's bare-board RTSs were GPL'd (they still are). Not that I cared about that, but other people did.

I took the approach of AdaCore's SFP (small footprint) profile, now renamed to light tasking, which implemented Ravenscar tasking but not exception propagation or finalization.

The tasking part wasn't too hard, though I think exception handling and finalization might have made things more difficult.

Basing the tasking on FreeRTOS saved a lot of grief (there are a couple of areas when the resulting semantic isn't _quite_ Ada's).

I did some work on finalization, not merged.

Exception handling, without finalization, seemed a daunting prospect, specially since the last project I worked on before retirement regarded an unhandled exception as requiring a reboot (and ditching any missiles in flight).

The current implementation has about 17 files (1 .h, 1 .s, 9 .ads, 4 .adb) to customise to the chip (setting up interrupt vectors, the clock, and memory). There are about 200 Ada sources that are common.

AdaCore currently has 68 RTS packages in the Alire gnat_arm_elf toolchain. 18 of these are 'embedded' packages (full Ada, but with Jorvik tasking). I'd be surprised if they had a higher proportion of chip dependency than my efforts. Most if not all of the exception handling will be chip-independent. I'm not sure how many of the 90 or so Ada sources in the STM32F4 gnarl/ directory are actually chip-dependent, I get the impression it's not high.

So, unless you're going to use some target that AdaCore haven't released support for, your best bet must be to either use or at worst start from the available RTS packages.

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

From: Drpi <314@drpi.fr>
Date: Thu, 28 Sep 2023 19:51:57 +0200

> I'm not sure how many of the 90 or so Ada sources in the STM32F4 gnarl/ directory are actually chip-dependent, I get the impression it's not high.

Right, not high.

I've created 2 of them based on one of the AdaCore RTS. I can't say it has been easy since you first have to understand how it works (and things change at each new release). One important point is that some critical parameters are hard coded in the source code. Like the core frequency. You MUST use a fixed clock frequency to get correct time management (delays, ...). This is why in their last version, you run a script to generate part of the RTS source code (frequency and other parameters are injected in the source code). When you change the core frequency you have to regenerate the RTS binary.

I created the RTS to evaluate the potential use of Ada on embedded targets. I have never used them except for testing. The main reason is that AdaCore RTS are made for specific use (avionics, spatial...). The code using these RTS must be provable (or as provable as possible). This induces big limitations. Tasking is very limited. For example you can't use timeouts. Never. They propose a workaround but it is complex and not equivalent to a real timeout management. I'd like to have a full Ada RTS for embedded targets, like on desktop. I don't need to certify/prove my hardware/software. Some people say micro-controllers are too limited for this. That's true for some of them. I use micro-controllers with megabytes of FLASH memory and hundreds of kilobytes of RAM. Is this not enough?

From: Simon Wright <simon@pushface.org>
Date: Thu, 28 Sep 2023 21:53:14 +0100

> I'd like to have a full Ada RTS for embedded targets, like on desktop.

Have you considered using something like a Raspberry Pi?

From: Drpi <314@drpi.fr>
Date: Thu, 28 Sep 2023 23:18:15 +0200

> Have you considered using something like a Raspberry Pi?

A RaspberryPi is a computer (based on a microprocessor with an OS), not a microcontroller. It consumes a lot of electrical power. The OS (linux) is not real time. It uses a lot of board space. The processor is a proprietary black box...

From: Chris Townley <news@cct-net.co.uk>
Date: Fri, 29 Sep 2023 00:51:11 +0100

> A RaspberryPi is a computer [...]

Plenty use the Raspberry Pi as a microcontroller.

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Fri, 29 Sep 2023 09:59:37 -0000

>Plenty use the Raspberry Pi as a microcontroller.

I think Simons point was that ARM/Linux has a working full runtime. I guess bare Raspberry Pi would not and I guess it would be a rather large module or board or single board computer depending on the model.

WRT energy use. I use a low power run feature on the STM32L4 which means the system clock speed can change at any time. That seems to be incompatible with any runtime that I have seen except the minimal light-cortex-m4 one. I assume working with clocks is more scalable than working with runtimes but I do not know for sure.

From: Chris Townley <news@cct-net.co.uk>
Date: Fri, 29 Sep 2023 11:42:08 +0100

> I think Simons point was that Arm/Linux has a working full runtime. [...]

Agreed, but in addition to the mainline Pis there is the Zero, and the Pico, which has a 'RP2040' made by Raspberry Pi and is a dual-core ARM Cortex M0+ processor, with a flexible clock running up to 133MHz.

From: Drpi <314@drpi.fr>
Date: Fri, 29 Sep 2023 15:42:17 +0200

> WRT energy use. I use a low power run feature on the STM32L4 which means the system clock speed can change at any time. [...]

The fact that the clock speed is hard coded is a design choice. It simplifies the time management. It makes the runtime more "hard real time" compliant since there are less computations to be done at execution.

From: Drpi <314@drpi.fr>
Date: Fri, 29 Sep 2023 15:44:31 +0200

> [...] which has a 'RP2040' made by Raspberry Pi and is a dual-core ARM Cortex M0+ processor, with a flexible clock running up to 133MHz

A runtime for the RP2040 already exists. It is based on the AdaCore ARM runtimes so it has the same limitations.

Conference Calendar

Dirk Craeynest
KU Leuven, Belgium. Email: Dirk.Craeynest@cs.kuleuven.be

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with ☺ denote events with close relation to Ada.

The information in this section is extracted from the on-line *Conferences and events for the international Ada community* at http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

The COVID-19 pandemic had a catastrophic impact on conferences world-wide. In general the situation seems to improve further, and only a few events are still planned to be held "virtually" or in "hybrid" mode. Where available, the status of events is indicated with the following markers: "(v)" = event is held online, (h)" = event is held in a hybrid form (i.e. partially online).

2023

October 02-04 — 25th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2023), Jersey City, New Jersey, USA. Topics include: design and development of distributed systems with a focus on systems that are able to provide guarantees on their structure, performance, and/or security in the face of an adverse operational environment; distributed and concurrent computing (foundations, fault-tolerance and scalability); distributed, concurrent, and fault-tolerant algorithms; synchronization protocols; formal methods, validation, verification, and synthesis; secure software and secure programming methodologies; formal methods, semantics and verification of secure systems; fault tolerance, reliability, availability of distributed secure systems; etc.

October 03-06 — 23rd **International Conference on Runtime Verification** (RV'2023), Thessaloniki, Greece. Topics include: monitoring and analysis of runtime behaviour of software and hardware systems; program instrumentation; logging, recording, and replay; combination of static and dynamic analysis; monitoring techniques for concurrent and distributed systems; fault localization, containment, resilience, recovery and repair; etc.

October 09-12 — 34th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2023), Florence, Italy. Topics include: development, analysis methods and models throughout the software development lifecycle; dependability attributes (i.e., security, safety, maintainability, survivability, resilience, robustness) impacting software reliability; reliability threats, i.e. faults (defects, bugs, etc.), errors, failures; reliability means (fault prevention, fault removal, fault tolerance, fault forecasting); reliability of AI-based systems; reliability of open-source software; reliability of model-based and auto-generated software; normative/regulatory/ethical spaces about software reliability; societal aspects of software reliability; etc.

☺ October 10-12 — 5th **International Conference on Reliability, Safety and Security of Railway Systems** (RSSRail'2023), Berlin, Germany. Topics include: safety in development processes and safety management; combined approaches to safety and security; system and software safety analysis; formal modelling and verification techniques; system reliability; validation according to the standards; tool and model integration, tool chain; domain-specific languages and modelling frameworks; model reuse for reliability, safety and security; etc.

October 16-18 — 8th **International Conference on Engineering Computer Based Systems** (ECBS'2023), Västerås, Sweden. Topics include: any topics that involve digital computing machines, embedded and cyber-physical systems, formal methods, software engineering for AI-based systems, parallel and distributed systems, software testing, verification and validation, model-based design and development, industrial applications and experimental evaluation, etc.

☺ October 17 — **High Integrity Software Conference** (HISC'2023), Bristol, UK. Topics include: advanced software development for high-integrity and high-assurance systems, including programming languages, AI-assisted software development, verifiable code generation; verification of novel, high-integrity and high-assurance systems; assurance of high-integrity, high-assurance systems; infrastructure & ecosystem for high-integrity software.

October 18-20 (h) 16th **International Conference on Verification and Evaluation of Computer and Communication Systems** (VECoS'2023), Marrakech, Morocco. Topics include: analysis of computer and communication systems, where functional and extra-functional properties are inter-related; cross-fertilization between various formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems.

October 19-20 (v) 19th **International Conference on Formal Aspects of Component Software** (FACS'2023), Internet. Topics include: applications of formal methods in all aspects of software components and services; formal methods, models, and languages for software-intensive systems, components and services: formal aspects of concrete software-intensive systems, including real-time/safety-critical systems, hybrid and cyber physical systems, components that use artificial intelligence, ...; tools supporting formal methods for components and services; case studies and experience reports over the above topics; special track on formal methods at large; etc.

☺ October 21-25 32nd **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2023), Vienna, Austria. Topics include: parallel architectures; compilers and tools for parallel computer systems; applications and experimental systems studies of parallel processing; computational models for concurrent execution; support for correctness in hardware and software; reconfigurable parallel computing; parallel programming languages, algorithms, and applications; middleware and run time system support for parallel computing; etc.

October 22-24 30th **Static Analysis Symposium** (SAS'2023), Cascais (Lisbon), Portugal. Co-located with SPLASH'2023. Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.

October 22-26 23nd IEEE **International Conference on Software Quality, Reliability and Security** (QRS'2023), Chiang Mai, Thailand. Topics include: reliability, security, availability, and safety of software systems; software testing, verification, and validation; program debugging and comprehension; fault tolerance for software reliability improvement; modeling, prediction, simulation, and evaluation; metrics, measurements, and analysis; software vulnerabilities; formal methods; operating system security and reliability; benchmark, tools, industrial applications, and empirical studies; etc.

☺ October 22-27 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2023), Lisbon, Portugal. Topics include: all aspects of software construction and delivery, at the intersection of programming, languages, and software engineering.

- Oct 22 9th **International Workshop on Formal Techniques for Safety-Critical Systems** (FTSCS'2023). Topics include: case studies and experience reports on the use of formal methods for analyzing safety-critical systems, including avionics, automotive, medical, railway, and other kinds of safety-critical and QoS-critical systems; methods, techniques and tools to support automated analysis, certification, debugging, etc., of safety/QoS-critical systems; code generation from validated models; etc.
- Oct 22-27 16th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2023). Topics include: software language engineering rather than engineering a specific software language; software language design and implementation; software language validation (verification and formal methods for languages, testing techniques for languages, simulation techniques for languages); software language integration and composition; software language maintenance (software language reuse, language evolution, language families and variability, language and software product lines); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.
- ☺ 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.

October 23 12th **Workshop on Programming Languages and Operating Systems** (PLOS'2023), Koblenz, Germany. Topics include: domain-specific and type-safe languages for the OS; the design of language-specific unikernels; language-based approaches to crosscutting system concerns, such as security and run-

time performance; PL support for system verification, testing, and debugging; the use of OS abstractions and techniques in language runtimes; verification and static analysis of OS components; critical evaluations of new programming language ideas in support of OS construction; experience reports on applying new language techniques in commercial OS settings; etc.

October 24-27 (h) 28th IEEE **Pacific Rim International Symposium on Dependable Computing** (PRDC'2023), Singapore. Topics include: software and hardware reliability, resilience, safety, security, testing, verification, and validation; dependability measurement, modeling, evaluation, and tools; architecture and system design for dependability; reliability analysis of complex systems; dependability issues in computing systems (e.g. high performance computing, real-time systems, cyber-physical systems, ...); emerging technologies (autonomous systems including autonomous vehicles, human machine teaming, smart devices/internet of things); etc.

October 24-27 21st **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2023), Singapore. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware and software systems; program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent systems; analysis and verification of deep learning systems; verification in industrial practice; applications and case studies; etc.

November 06-10 21st **International Conference on Software Engineering and Formal Methods** (SEFM'2023), Eindhoven, the Netherlands. Topics include: software development methods (formal modelling, specification, and design; software evolution, maintenance, re-engineering, and reuse), design principles (programming languages; abstraction and refinement; ...), software testing, validation, and verification, security and safety (security, privacy, and trust; safety-critical, fault-tolerant, and secure systems; software certification), applications and technology transfer (real-time, hybrid, and cyber-physical systems; intelligent systems and machine learning; education; ...), case studies, best practices, and experience reports.

November 13-15 18th **International Conference on integrated Formal Methods** (iFM'2023), Leiden, the Netherlands. Topics include: recent research advances in the development of integrated approaches to formal modelling and analysis; all aspects of the design of integrated techniques, including language design, verification and validation, automated tool support and the use of such techniques in software engineering practice.

Nov 14 **VerifyThis Collaborative Long-Term Challenge**. Topics include: demonstrate practical value of formal methods, evaluate current tools on specifying and verifying requirements of realistic software systems, exchange on the state-of-the-art and future directions.

Nov 13-17 (h) 18th **International Conference on Software Engineering Advances** (ICSEA'2023), Valencia, Spain. Topics include: trends and achievements; advances in fundamentals for software development; advanced mechanisms for software development; advanced design tools for developing software; software performance; software security, privacy, safeness; advances in software testing; specialized software advanced applications; open source software; agile and Lean approaches in software engineering; software deployment and maintenance; software engineering techniques, metrics, and formalisms; software economics, adoption, and education; etc.

November 26-29 21st **Asian Symposium on Programming Languages and Systems** (APLAS'2023), Taipei, Taiwan. Topics include: programming paradigms and styles; methods and tools to specify and reason about programs and languages (programming techniques, domain-specific languages, proof assistants, type systems, static and dynamic program analysis, language-based security, model checking, testing, ...); programming language foundations; methods and tools for implementation (compilers, virtual machines, refactoring, intermediate languages, run-time environments, garbage collection and memory management, build systems, ...); concurrency and distribution (parallel programming, distributed computing, verification and testing of concurrent and distributed systems, ...); applications and emerging topics (programming languages and PL methods in education, security, privacy, artificial intelligence and machine learning, ...; case studies in program analysis and verification). Deadline for early registration: October 25, 2023.

December 03-09 31st ACM **Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering** (ESEC/FSE'2023), San Francisco, California, USA. Deadline for submissions: October 20, 2023 (student travel support applications).

December 04-07 30th **Asia-Pacific Software Engineering Conference** (APSEC'2023), Seoul, South Korea. Topics include: requirements and design (component-based software engineering; software architecture, modeling, and design; middleware, frameworks, and APIs; software product-line engineering; ...); testing and analysis (testing, verification, and validation; program analysis; program repairs; ...); formal aspects of software engineering (formal methods, model-driven and domain-specific engineering); software comprehension and traceability; dependability, safety, and reliability; software maintenance and evolution (refactoring, reverse engineering, software reuse, debugging and fault localization, ...); software repository mining; etc.

December 04-08 20th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2023), Lima, Peru. Topics include: semantics of programming languages; theories of concurrency; theories of distributed computing; models of objects and components; timed, hybrid, embedded, and cyber-physical systems; security; static analysis; software verification; software testing; model checking and theorem proving; applications and experiences; etc.

December 05-08 43rd IEEE **Real-Time Systems Symposium** (RTSS'2023), Taipei, Taiwan.

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

2024

January 15-16 25th **International Conference on Verification, Model Checking, and Abstract Interpretation** (VMCAI'2024), London, UK. Co-located with POPL'2024. Topics include: program verification, model checking, abstract interpretation, static analysis, type systems, program certification, detection of bugs and security vulnerabilities, hybrid and cyber-physical systems, concurrent and distributed systems, analysis of numerical properties, analysis of smart contracts, etc., case studies on all of the above topics.

☺ January 17-19 51st ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2024), London, UK. Topics include: fundamental principles and important innovations in the design, definition, analysis, transformation, implementation, and verification of programming languages, programming systems, and programming abstractions. Deadline for submissions: October 15, 2023 (tutorials).

Jan 15-16 **International Conference on Certified Programs and Proofs** (CPP'2024). Topics include: new languages and tools for certified programming; program analysis, program verification, and program synthesis; program logics, type systems, and semantics for certified code; verification of correctness and security properties; etc.

January 17-19 19th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2024), Munich, Germany. Topics include: computer architecture, programming models, compilers and operating systems for general-purpose, embedded and cyber-physical systems. Areas include safety-critical dependencies, cybersecurity, energy efficiency and machine learning.

February 07-09 18th **International Working Conference on Variability Modelling of Software-Intensive Systems** (VaMoS'2019), Bern, Switzerland. Topics include: variability across the software lifecycle, test and verification of variable systems, evolution of variability-intensive systems, runtime variability, variability mining, reverse-engineering of variability, economic aspects of variability, variability and quality requirements, industrial development of variable systems, experience reports from managing variability in practice, etc. Deadline for submissions: October 10, 2023 (abstracts), November 17, 2023 (papers), October 27, 2023 (New and Controversial Ideas Track papers).

March 02-06 IEEE/ACM **International Symposium on Code Generation and Optimization** (CGO'2024), Edinburgh, UK.

March 12-15 31st IEEE **International Conference on Software Analysis, Evolution, and Reengineering** (SANER'2024), Rovaniemi, Finland. Topics include: 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. Deadline for submissions: October 13, 2023 (research abstracts), October 29, 2023 (research

papers), November 6, 2023 (registered reports), November 10, 2023 (Reproducibility Studies and Negative Results (RENE) abstracts), November 17, 2023 (RENE papers), November 13, 2023 (industrial abstracts, short papers and posters abstracts, Early Research Achievement (ERA) abstracts), November 20, 2023 (industrial papers, short papers and posters papers, ERA papers, tool demo papers), January 14, 2024 (Journal First papers).

March 20-22 32nd **Euromicro International Conference on Parallel, Distributed and Network-Based Processing** (PDP'2024), Dublin, Ireland. Topics include: embedded parallel systems, dependability, survivability, programming languages, compilers, middleware, runtime, and systems software, performance prediction and analysis, simulation and modelling of parallel/distributed systems, etc.

April 06-11 27th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2024), Luxembourg City, Luxembourg. 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: October 12, 2023 (papers), November 9, 2023 (TACAS artifact submissions), January 4, 2024 (ESOP, FASE, FoSSaCS artifact submissions).

April 08-11 30th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2024), Winterthur, Switzerland. Theme: "Out of the Lab, into the Wild!" Deadline for submissions: October 27, 2023 (workshops), November 10, 2023 (research papers), February 9, 2024 (workshop submissions, education and training track, posters, tools, doctoral symposium).

April 08-12 36th ACM **Symposium on Applied Computing** (SAC'2024), Avila, Spain.

☺ Apr 08-12 **Track on Programming Languages** (PL'2024). Topics include: technical ideas and experiences relating to implementation and application of programming languages, such as compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, etc. Deadline for submissions: October 13, 2019 (regular papers, SRC research abstracts).

Apr 08-12 **Software Verification and Testing Track** (SVT'2024). Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, model checking, correct by construction development, model-based testing, software testing, static and dynamic analysis, abstract interpretation, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging, verification and validation of large scale software systems, real world applications and case studies applying software testing and verification, etc.

April 14-20 46th **International Conference on Software Engineering** (ICSE'2024), Lisbon, Portugal. Deadline for submissions: October 30, 2023 (TCSE award nominations).

April 14-15 12th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2024). 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 (such as, autonomous, cyber-physical, intelligent, and IoT systems); formal methods for certification; guidelines to use formal methods in practice; usability of formal methods; etc. Deadline for submissions: December 1, 2023 (abstracts), December 8, 2023 (papers).

April 24-25 16th **Software Quality Days** (SWQD'2024), Vienna, Austria. Theme: "Software Quality as a Foundation for Security". Topics include: all topics related to software and systems quality, such as methods and tools for constructive and analytical quality assurance; testing of software and software-intensive systems; process improvement for development and testing; automation in quality assurance and testing; domain specific quality issues such as embedded, medical, automotive systems; continuous integration, deployment, and delivery; project and risk management; secure coding, software engineering and system design; detection and prevention of vulnerabilities and security threats; etc. Deadline for submissions: November 17, 2023.

May 07-11 15th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2024), London, UK. Deadline for submissions: October 13, 2023 (workshops), October 20, 2023 (SPEC Kaivalya Dixit Distinguished Dissertation award nominations), October 27, 2023 (research track abstracts), November 03, 2023 (research track papers, industry track papers), January 12, 2024 (artifacts), January 26, 2024 (emerging research track, posters, demos, tutorials), February 9, 2024 (data challenge).

May 13-16 17th **Cyber-Physical Systems and Internet of Things Week** (CPS-IoT Week'2024), Hong Kong. Event includes: 5 top conferences, HSCC, ICCPS, IoTDI, IPSN, and RTAS, as well as poster and demo sessions, workshops, tutorials, competitions, industrial exhibitions, and PhD forums. Deadline for submissions: October 13, 2023 (papers), November 7, 2023 (workshops, tutorials, competition proposals).

☺ May 13-16 29th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2024). Topics include: time-sensitive applications; real-time and embedded operating systems; application profiling, WCET analysis, compilers, tools, benchmarks and case studies; modelling languages, modelling methods, model learning, model validation and calibration; scheduling and resource allocation; verification and validation methodologies; etc. Deadline for submissions: October 31, 2023 (papers).

May 13-16 15th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2024). Topics include: safety and resilience for CPS; software platforms and systems for CPS; specification languages and requirements; design, optimization, and synthesis; testing, verification, certification; security, trust, and privacy in CPS; tools, testbeds, demonstrations and deployments; etc. Deadline for submissions: October 24, 2023 (abstracts), October 31, 2023 (papers).

May 27-31 38th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2024), San Francisco, California, USA. Topics include: applications to solve problems using parallel and distributed computing concepts; programming models, compilers, and runtime systems (ranging from the design of parallel programming models and paradigms to languages and compilers supporting these models and paradigms, to runtime and middleware solutions); system software; existing and emerging architectures; experiments and performance-oriented studies in the practice of parallel and distributed computing; etc. Deadline for submissions: October 5, 2023 (papers).

May 27-31 17th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2024), Toronto, Canada. 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/cyber-physical systems, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc. Deadline for submissions: October 18, 2023 (abstracts), October 25, 2023 (papers).

June 04-06 16th **NASA Formal Methods Symposium** (NFM'2024), Moffett Field, California, USA. Topics include: identifying challenges and providing solutions towards achieving assurance for critical systems; formal techniques for software and system assurance for applications in space, aviation, robotics, and other NASA-relevant safety-critical systems. Deadline for submissions: December 1, 2023 (abstracts), December 8, 2023 (papers).

June 10-12 21st **International Conference on Software and Systems Reuse** (ICSR'2024), Limassol, Cyprus. Theme: "Sustainable Software Reuse". Topics include: new and innovative research results and industrial experience reports dealing with all aspects of software reuse within the context of the modern software development landscape, such as technical aspects of reuse (model-driven development, variability management and software product lines, domain-specific languages, new language abstractions for software reuse, software composition and modularization, technical debt and software reuse, ...), software reuse in industry and in emerging domains (reuse success stories, reuse failures and lessons learned, reuse obstacles and success factors, return on investment studies, ...). Deadline for submissions: January 22, 2024 (abstracts), January 29, 2024 (full papers).

☺ June 11-12 12th **European Congress on Embedded Real Time Systems** (ERTS'2024), Toulouse, France. Topics include: all aspects of critical embedded real-time systems, such as model-based system and safety engineering, product line engineering, programming languages, verification methods, software development frameworks, dependability, safety, cyber security, quality of service, fault tolerance, maintainability, certification, etc. Deadline for submissions: October 15, 2023 (regular abstracts, short papers), April 3, 2024 (regular papers), May 5, 2024 (final short and regular papers).

♦ June 11-14 28th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC'2024), Barcelona, Spain. Organized by Ada-Europe and Barcelona Supercomputing Center. In cooperation with the Ada Resource Association (ARA), ACM SIGAda, ACM SIGBED, and ACM SIGPLAN. Deadline for submissions: January 15, 2024 (journal-track papers), February 26, 2024 (industrial track and work-in-progress papers, tutorial and workshop proposals). #AEiC2024 #AdaEurope #AdaProgramming

☺ September 16-20 38th **European Conference on Object-Oriented Programming** (ECOOP'2024), Vienna, Austria. Topics include: programming languages, software development, systems and applications.

☺ October 20-25 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2024), Pasadena, California, USA.

☺ Oct 20-25 **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2024). Topics include: all practical and theoretical investigations of programming languages, systems and environments, targeting any stage of software development, including requirements, modelling, 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: October 20, 2023 (round 1), April 5, 2024 (round 2).

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

BARCELONA
AEiC 2024

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

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

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

Industrial track Chairs
Luciana Provenzano
luciana.provenzano@mdu.se
Mälardalen University, Sweden

Michael Pressler
Michael.Pressler@de.bosch.com
Robert Bosch GmbH, Germany

Work-In Progress-track Chairs
Alejandro R. Mosteo
amosteo@unizar.es
CUD Zaragoza, Spain

Ruben Martins
rubenm@andrew.cmu.edu
Carnegie Mellon University, USA

Tutorial Chair
Maria A. Serrano
maria.serrano@nearbycomputing.com
NearbyComputing, Spain

Workshop Chair
Sergio Saez
ssaez@disca.upv.es
Universitat Politècnica de València, Spain

Exhibition & Sponsorship Chair
Ahlan Marriott
ahlan@Ada-Switzerland.ch
White Elephant GmbH, Switzerland

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

Webmaster
Hai Nam Tran
hai-nam.tran@univ-brest.fr
University of Brest, France

Local Chair
Nuria Sirvent
nuria.sirvent@bsc.es
Barcelona Supercomputing Center, Spain

The 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) will take place in Barcelona, Spain.

AEiC 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 include keynotes, Q&A sessions, and social events where practitioners and researchers from industry, academia, and government organizations active in the promotion and development of reliable software technologies.

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

- *Formal and model-based engineering of critical systems*
- *High-Integrity Systems and Reliability*
- *AI for High-Integrity Systems Engineering*
- *Real-Time Systems*
- *Ada Language*
- *Applications in relevant domains*

The conference comprises different tracks and co-located events:

- *Journal track* papers present research advances supported by solid theoretical foundation and thorough evaluation.
- *Industrial track* contributions highlight industrial open challenges and/or the practitioners' side of a relevant case study or industrial project.
- *Work-in-progress track* papers illustrate novel research ideas that are still at an initial stage, between conception and first prototype.
- *Tutorials* guide attenders through a hands-on familiarization with innovative developments or with useful features related to reliable software.
- *Workshops* provide discussion forums on themes related to the conference topics.
- *Vendors presentations and exhibitions* allow for companies to showcase their latest products.

Important Dates

15 January 2024	Deadline for submission of journal-track papers
26 February 2024	Deadline for submission of industrial-track papers, work-in-progress papers, tutorials, and workshop proposals
22 March 2024	First round notification for journal-track papers, and notification of acceptance for all other types of submissions
11-14 June 2024	Conference

Call for journal track submissions

Following a journal-first model, this edition of the conference 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 a Special Issue of Elsevier JSA – the *Journal of Systems Architecture* (Q1 ranked, CiteScore 8.5, impact factor 4.5). Accordingly, the conference is listed as "Journal Published" in the latest update of the CORE Conference Ranking released in August 2023. Contributions must be submitted by **15 January 2024**. Submissions should be made online at https://www.editorialmanager.com/jsa/, selecting the "Ada-Europe AEiC 2024" option (submission page open from 15 November 2023) as article type of the paper. General information for submitting to the JSA can be found at the Journal of Systems Architecture website.

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. The abstract of the accepted contributions will be included in the conference booklet.

The Ada-Europe organization will waive the Open Access fees for the first four accepted papers (whose authors do not already enjoy Open Access agreements). Subsequent papers will follow JSA regular publishing track. Prospective authors may direct all enquiries regarding this track to the corresponding chairs, Bjorn Andersson (baandersson@sei.cmu.edu) and Luis Miguel Pinho (lmp@isep.ipp.pt).

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 **26 February 2024**, via EasyChair at https://easychair.org/my/conference?conf=aeic2024, 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 Luciana Provenzano (luciana.provenzano@mdu.se) and Michael Pressler (Michael.Pressler@de.bosch.com).

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 **26 February 2024**, via EasyChair at https://easychair.org/my/conference?conf=aeic2024, 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 Alejandro R. Mosteo (amosteo@unizar.es) and Ruben Martins (rubenm@andrew.cmu.edu).

Call for tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic for 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 at any time but no later than the **26 February 2024** to the respective chair Maria A. Serrano (maria.serrano@nearbycomputing.com), with subject line: "[AEiC 2024: tutorial proposal]". Once submitted, each workshop proposal will be evaluated by the conference organizers as soon as possible. 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 at any time but no later than the **26 February 2024** to the respective chair Sergio Saez (ssaez@disca.upv.es), with subject line: "[AEiC 2024: workshop proposal]". Once submitted, each workshop proposal will be evaluated by the conference organizers as soon as possible.

Call for exhibitors and sponsors

The conference will include a vendor and technology exhibition with the option of a 20 minutes presentation as part of the conference program. Interested providers should direct inquiries to the Exhibition & Sponsorship Chair Ahlan Marriot (ahlan@ada-switzerland.ch).

Venue

The conference will take place in Barcelona, Spain. Barcelona is a major cultural, economic, and financial centre, known for its architecture, culture, and Mediterranean atmosphere, a hub for technology and innovation. There's plenty to see and visit in Barcelona, so plan in advance!

Organized by:

In cooperation with:

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/

VR-Based Teleoperation of Autonomous Vehicles for Operation Recovery

Georg Jäger, Gero Licht, Norman Seyffer, Stefan Reitmann
Freiberg University of Mining & Technology, Institute for Computer Science, Akademiestraße 6, 09599 Freiberg, Germany; email: {georg.jaeger, gero.licht, norman.seyffer, stefan.reitmann}@informatik.tu-freiberg.de

Abstract

While research will enable the deployment of autonomous systems in harsh and inaccessible environments, their operation may be interrupted due to unforeseen situations. A possibility to recover operation nonetheless is to employ teleoperation. However, what requirements and criteria need to be fulfilled by such a system when deployed in safety-critical operation scenarios? How can a timely and safe operation recovery be ensured? The present work aims to report our progress in developing a research platform for addressing these and similar questions.

1 Introduction

Autonomous systems' potential to alleviate people from manual and repetitive tasks has fueled their development. Use cases span from environmental monitoring to individualized delivery systems and beyond.

For such deployments, however, safety of these systems becomes a mandatory prerequisite. Several standards and norms exist, e.g. based on the IEC 61511 [5]. Their overarching idea is to provide guarantees at design time that the system will be safe at run time.

However, providing such guarantees becomes increasingly difficult. Firstly, because of the system's complexity, which is (secondly) increasing due to the open operation environments that render the task of defining a system's Operational Design Domain (ODD) with sufficient coverage challenging as well [8]. The latter entails that the system under design will be faced with unforeseen situations at run time. In these cases, the system is commonly brought into a predetermined fail-safe state (if applicable) and interrupted in its mission execution.

But how long can a system remain in its fail-safe state? This is use-case dependent. For instance, for an autonomous delivery system operating on sidewalks, it is assumed that stopping its movement in case of a failure is safe. Contrarily, this assumption does not hold in case the failure occurs while the system is crossing a road. There, stopping the movement can be considered safe only for a limited period. Operation recovery is therefore mandatory to maintain the system's safety. As we presume, however, that an unforeseen situation occurred, we consequently need to assume that the system will not be able to recover on its own. A possible option to recover operation nonetheless is to teleoperate the system remotely. This places the human operator and the corresponding teleoperation system in the center of the overall system's safety argumentation. Therefore, a central question is: **what requirements do teleoperation systems need to fulfill to be allowed as a safety functionality in autonomous systems?**

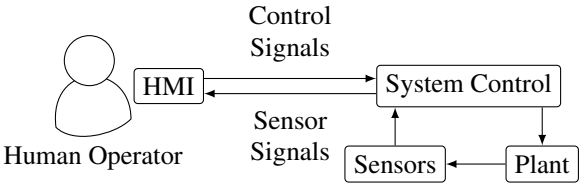

Figure 1: General Structure of a Teleoperation Scenario for a Single Autonomous System.

To analyze this question, the next section will briefly review the state of the art in teleoperation systems and current discussions on appropriate Human Machine Interface (HMI) solutions. Section 3 will present the system currently under design along with two use cases we are focusing on - Autonomous Surface Vehicles (ASVs) (robotic boats) for environmental monitoring and medium-sized Autonomous Ground Vehicles (AGVs) for urban scenarios, in this case, delivery systems. As the latter is a work-in-progress, we point out research opportunities related to the use cases in section 4 before we conclude the article.

2 State of the Art on Teleoperation Systems

Teleoperation systems are a long-standing field of research that has led to a rich literature that we want to briefly present in this section.

The general architecture of teleoperation systems is shown in Figure 1. It consists of three main components: a human operator presented with an HMI, a bidirectional data connection, and the remotely controlled system with its sensors and separate control architecture.

Commonly, a remotely located human operator is presented with the current state of the controlled system as perceived by its sensors through an HMI. It allows the operator to determine appropriate control commands sent to and executed by the controlled system.

For this loop to be successful, the HMI has to enable the operator to correctly understand and assess the state and context of

the remote system. To that end, existing concepts leverage the visual focus of operators and present them with video streams of onboard cameras, e.g. as in [2]. However, to support a better sense of orientation and scale those streams can be processed to present enriched views as deployed by Schimpe et al. [11].

Next to the data, the presentation itself is central to enable the operator to safely control the remote system. For that, Hosseini and Lienkamp [4] propose using Head-Mounted Displays (HMDs) to enhance situation awareness and task performance. Lager and Topp [7] confirm that users working with VR-GUIs perform better in detecting collision or grounding situations of surface vehicles than users working with conventional GUIs or even 3D-GUIs. The users rated the VR-GUI to be the easiest tool to use and at the same time the best expert tool.

As before, the visual focus of human perception is exploited by allowing operators to use their natural behavior (e.g. turning their head) to create an immersive experience increasing situational awareness. For that, the style with which the information is presented has to be designed carefully, especially for such VR-based systems. Inappropriate design can increase the risk of *motion sickness* of the operator. It emerges from disagreeing sensational stimuli and can cause, for instance, disorientation and nausea [1].

A source of such disagreeing sensational stimuli, other than the HMD, can be delays and limited bandwidth of the transmission link. Although not using HMD, the authors of [10] show that operators are capable of navigating successfully with low-resolution data but are challenged when presented with delayed data. In their use case of operating a quadrotor, they identified $442\,\mathrm{ms}$ of delay as the break-even point for reducing the resolution and switching to simplifying data abstractions.

The importance of delays was underlined in [11] and [2] where low-level control signals (steering angle and velocity) are sent to a passenger vehicle. Measuring delays of about 100-120 ms, it was deemed sufficient to maneuver the car at low to intermediate speeds of no more than $35\,\mathrm{km\,h^{-1}}$. While this latency was satisfactory for the most part, [2] encountered delays peaking at $>500\,\mathrm{ms}$ during their test drives on a 3G Network. The authors conclude that such delays are acceptable for rural roads but not for crowded areas of challenging terrains where they become a threat to the system's safety.

In the endeavor of identifying the set of relevant hazards for teleoperation systems in automotive scenarios, [3] consider not only the human operator or the delays of the data transmission, but also consider the controlled system itself. However, these are focused on automotive systems and therefore do not consider the contextual situations of, for instance, environmental monitoring systems or autonomous delivery systems.

Thus, while a rich set of approaches and analyses regarding the performance of teleoperation systems exists, only a few works address the challenges of safety argumentation.

Figure 2: Clearpath Kingfisher ASV with 360° Camera.

3 VR-based Teleoperation for Operation Recovery

To enable analyzing safety challenges and argumentations on conceptual and practical abstraction levels, we are currently developing a VR-based teleoperation system. We will firstly introduce the state of the current prototype in the next subsection before we elaborate on the planned future work.

3.1 Teleoperation for Robotic Boats

The prototype presented here is developed for robotic boats - in particular for a Clearpath Kingfisher (cf. Figure 2). It was used within the ESF project *RoboBoatAssist* (2022) for the assistance of robotic boats.

The boats were designed to autonomously monitor and record water parameters such as bathymetry data [9]. However, due to critical situations occurring within the shore area, manual control is necessary. For that, a Virtual Reality (VR)-based assistance system using HMD was developed. The central source of information is a video stream produced by a 360° camera (*Theta Z1*) mounted on the boat, cf. Figure 2. Using Robot Operating System (ROS) and the *GStreamer* framework, the images are streamed from the boat to the operator using a HMD. It is a standalone solution working without a separate computer in between. The Z1 produces spherical videos in 4K (3840 x 1920, 29.97 FPS), which can be streamed to the **Unity**[1] game engine with a delay $t_{offset} = t_{HMD} - t_{captured} < 0.1\,\mathrm{s}$ via $2.4\,\mathrm{GHz}$ WiFi network.

The operator wearing the HMD is located within a virtual environment showing the video stream mapped to a sphere around his virtual position in the Unity scene (see fig. 3). It allows exporting applications for HMDs in general and enables using the HTC Focus 3 in this case. This specific model has inside-out tracking making it applicable for outdoor environments as no additional equipment or setup is required for tracking the operator's movement.

For connecting Unity to the ROS network, the Transmission Control Protocol (TCP) endpoint and connector packages developed by Unity Technologies[2, 3] are employed. As this

[1] https://unity.com/

[2] https://github.com/Unity-Technologies/ROS-TCP-Connector

[3] https://github.com/Unity-Technologies/ROS-TCP-Endpoint

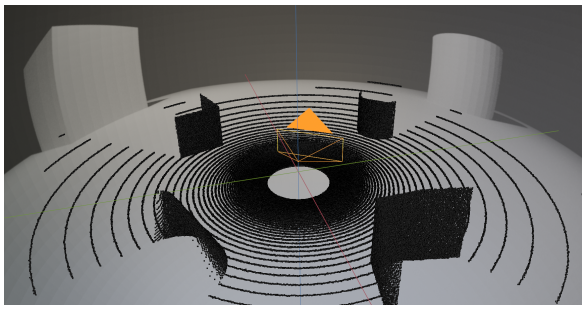

Figure 3: Combining image and spatial data: the operator is placed in the middle $(0, 0, 0)$ of the virtual scene using a virtual, VR-based camera (orange). Sensor data is visualized within the scene using a sky sphere ($360°$ image as texture) and a particle system (point cloud from a rotating LiDAR).

Figure 4: The research platform *Clearpath Husky* extended to support the use case of autonomous delivery systems.

means that the VR system itself is not a ROS node but communicates using generic TCP connections, adapting it to different releases (ROS version 1 or 2) is straightforward and planned for future work.

3.2 Teleoperation for Autonomous Delivery Systems

Within the research project *RoboTraces*[4], we plan to extend our prototypical VR-based teleoperation system to land-based AGVs, specifically the *Clearpath Husky*. For that, we focus on the use case of autonomous delivery systems deployed in open environments where defining the ODD becomes challenging and requiring human control in unforeseen situations.

Simultaneously, the scenarios in which a human operator is required to take over control of the autonomous system increase in complexity and thereby challenge situational awareness. To mitigate the effects, we are planning to integrate additional sensory information in the virtual environment presented to the operator.

In a first step, the available spatial data will be exploited. The system features four ZED2i stereo cameras covering a $360°$ view around the system while producing not only RGB but also depth information. Moreover, two solid-state Light Detection and Ranging (LiDAR) mounted in the front and back provide additional spatial data. In both cases, *particle systems* can be used for visualization in the Unity application, cf. Figure 3).

In a second step, near-field ultra-sonic distance sensors will be used to give feedback about obstacles becoming dangerously close to the system. The view will be colored red indicating that immediate actions are needed to prevent a collision.

Finally, in a third step, processed and abstracted data is to be presented. It is planned to simplify the presentation of detected obstacles and their categories to not only support the operator in understanding the system's environment, but also the system's perception of the same.

4 Research Opportunities

The planned extensions and the presented use cases shall enable researching requirements to be allocated to such a teleoperation system when used in safety-critical scenarios, such as operation recovery. The corresponding research opportunities can be associated with the operator and its HMI, the transmission of data, and the system under control. We will briefly discuss these opportunities in the next subsections.

4.1 Human Operator

Since teleoperation will act as a fallback option for operation recovery, one central aspect to ensure the system's safety is the human operator - which therefore raises safety-related questions. In [3] and related work, for instance, different hazards associated with the operator are identified. They can be categorized into (in-)appropriate training, mental and physical well-being, and situational awareness.

Since the chain of argumentation for safety encompasses the human operator, training is essential to reduce risks related to inappropriate control actions. Similarly, mental and physical well-being needs to be supported by the HMI where *motion sickness* (cf. Section 2) becomes significant when using HMDs.

Most central for the operator to determine safe control actions, however, is its situational awareness. Its provision by an employed teleoperation system turns therefore into a safety question. From that line of thought, we identified the following, non-exhaustive list of research questions:

- Is there a set of functionalities to guarantee situational awareness in teleoperation systems?
- How should non-visual information about the remotely controlled system and its surroundings be displayed?
- Do we need an environmental model of the remote system within the HMI?

With respect to safety, determining a minimal set of functionalities to be provided by teleoperation systems to enable situational awareness by trained operators could serve as a baseline.

Supposedly, this set of functionalities is use-case dependent - with increasing complexity of the same, the teleoperation system may be obligated to provide a better presentation of information. With our VR-based approach, visual and spatial information can be displayed intuitively to the operator.

[4]This research was partially founded by German Ministry for Digital and Transport under the grant no. 19F1117B

However, other relevant parameters (e.g. wind strength and direction for ASVs or unsafe distances of obstacles for AGVs) need to be displayed as well.

Moreover, depending on the situation, the operator could be supported in interpreting the displayed information by highlighting them appropriately. For that, having an environmental model of the remote system within the HMI could form a basis.

4.2 Transmission of Data

Next to the HMI of the operator, its connectivity and data transmission from/to the remotely controlled system has to be taken into consideration. Here security becomes a central concern. Since the operator will control the system remotely, the system has to have a constant digital connection. This connection, however, exposes an attack surface such that adversaries may intrude and threaten to (i) take over control of the controlled system to perform malicious (and therefore unsafe) control actions, (ii) compromise the integrity of transmitted data, which reduces or nullifies the situational awareness of the operator, (iii) or to compromise confidentiality such that private data as obtained by RGB cameras is accessible by unauthorized personal.

Although these challenges are common to the field of security, solutions to them require taking the safety of the system into account. For instance, employed encryption can not introduce delays ([10]), incomplete data transmissions or connection loss. All of these may also be introduced by general connectivity issues:

- How to ensure integrity, confidentiality, and availability with respect to the cybersecurity of the teleoperated system while not impacting the system's performance negatively?
- How to handle incomplete data transmission?
- How to ensure safety in case of connection loss?

A loss in either of the three security attributes can have severe consequences - not only but also with respect to safety. Next to the actual threat of a cyberattacker compromising any of the attributes, implementing a solution to the problem likely requires overhead in processing and data transmission. This may result in increased processing power consumption and reduces energy efficiency.

Moreover, data transmission can also fail because the network becomes unavailable. From the operator's perspective, however, deciding on an appropriate control action is most central, leaving the decision on how to handle incomplete data transmission to the system. Here, one has to take situational awareness of the operator into account again: is it favorable to either fully apply a control signal or not at all and thus introducing transactional behavior to prevent divergence between the operator's perception of the remote situation and its reality? Or could it be possible to detect partially transmitted data to provide corresponding feedback to the operator?

If delays and incomplete data transmission increase, the worst-case scenario may occur: connection loss during a teleoperation session. In this case, the operator can not interact with the system anymore - thus transferring the responsibility of safety back to the autonomous system.

4.3 Capabilities of the Remote System

While the challenge of connection loss originates in the transmission link between the teleoperated system and the human operator, addressing it involves the system itself. Since the autonomous system is forced back to maintain its safety autonomously, the question arises:

- Which levels of autonomy does the remotely controlled system need to maintain?

As the teleoperation system itself is a fallback option for autonomous behavior already, a connection loss could result in a complete system failure. Considering that an operator could have connected to the system before, the situation could have improved and thus may allow the robot to obtain a fail-safe state at least. For that, however, it needs to be discussed which autonomous functionalities to enable (automatically or specifically by the operator) once a connection was established. Then, in case of a connection loss, maintaining the system's safety through a reduced set of autonomous functionalities could be possible - and is therefore opening another research opportunity.

5 Conclusions

In this article, we present the work in progress to implementing a VR-based teleoperation system for operation recovery of autonomous systems. We focus on the use cases of ASVs (robotic boats) and AGVs (autonomous delivery systems).

While the current system allows visualizing video streams of an onboard camera through an HMD, the system is to be extended for presenting spatial and abstracted data as well.

With this platform, we aim at researching questions concerning the safety of teleoperated systems. An initial list of questions is presented in the previous section but needs to be extended and supported by a structured analysis. In that endeavor, we aim for applying the System-Theoretic Process Analysiss (STPAs). STPA differs from traditional hazard analysis techniques such as Fault Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA) as it does not assume random component failures to be the origin of a chain of events eventually leading to a system failure. In contrast, STPA considers system failures as a consequence of inappropriate and/or unsafe control actions and asks for reasons that might have caused them. Thus, it considers the system and its components as control systems interacting with their environment and with each other. The resulting perspective allows to identify hazards in complex software systems and is therefore well suited for the analysis of teleoperation systems [6].

Appendix

The following abbreviations are used in this manuscript:

STPA System-Theoretic Process Analysis 207
TCP Transmission Control Protocol 205
VR Virtual Reality 205

References

[1] Eunhee Chang, Hyun Taek Kim, and Byounghyun Yoo. "Virtual Reality Sickness: A Review of Causes and Measurements". In: *International Journal of Human–Computer Interaction* (2020).

[2] Sebastian Gnatzig et al. "A System Design for Teleoperated Road Vehicles". In: *International Conference on Informatics in Control, Automation and Robotics*. 2013. URL: https://api.semanticscholar.org/CorpusID:46122663.

[3] Simon Hoffmann and Dr. Frank Diermeyer. *Systems-Theoretic Safety Assessment of Teleoperated Road Vehicles*. 2021. URL: https://arxiv.org/abs/2104.06795.

[4] Amin Hosseini and Markus Lienkamp. "Enhancing telepresence during the teleoperation of road vehicles using HMD-based mixed reality". In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. 2016, pp. 1366–1373. DOI: 10.1109/IVS.2016.7535568.

[5] International Electrotechnical Commission. *Functional safety - Safety instrumented systems for the process industry sector*. Standard. Geneva, CH: International Electrotechnical Commission, 2020.

[6] Takuto Ishimatsu et al. "Hazard analysis of complex spacecraft using systems-theoretic process analysis". In: *Journal of spacecraft and rockets* 51.2 (2014), pp. 509–522.

[7] Mårten Lager and Elin A. Topp. "Remote Supervision of an Autonomous Surface Vehicle using Virtual Reality". In: *IFAC-PapersOnLine* (2019). 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019. ISSN: 2405-8963.

[8] Alexander Poddey et al. "On the validation of complex systems operating in open contexts". In: *arXiv preprint arXiv:1902.10517* (2019).

[9] Sebastian Pose et al. "AI-Prepared Autonomous Freshwater Monitoring and Sea Ground Detection by an Autonomous Surface Vehicle". In: *Remote Sensing* (2023).

[10] Maik Riestock et al. "User study on remotely controlled uavs with focus on interfaces and data link quality". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017.

[11] Andreas Schimpe et al. "Open Source Software for Teleoperated Driving". In: *2022 International Conference on Connected Vehicle and Expo (ICCVE)*. 2022, pp. 1–6. DOI: 10.1109/ICCVE52871.2022.9742859.

Denoising Autoencoder-Based Defensive Distillation as an Adversarial Robustness Algorithm Against Data Poisoning Attacks

Bakary Badjie, José Cecílio, António Casimiro
LASIGE, Departamento de Informática, Faculdade de Ciências da Universidade Lisboa, Lisboa; email: {bbadjie, jmcecilio, casim}@ciencias.ulisboa.pt

Abstract

Deep neural networks (DNNs) have demonstrated promising performances in handling complex real-world scenarios, surpassing human intelligence. Despite their exciting performances, $DNNs$ are not robust against adversarial attacks. They are specifically vulnerable to data poisoning attacks where attackers meddle with the initial training data, despite the multiple defensive methods available, such as defensive distillation. However, defensive distillation has shown promising results in robustifying image classification deep learning (DL) models against adversarial attacks at the inference level, but they remain vulnerable to data poisoning attacks. This work incorporates a data denoising and reconstruction framework with a defensive distillation methodology to defend against such attacks. We leverage a denoising autoencoder (DAE) to develop a data reconstruction and filtering pipeline with a well-designed reconstruction threshold. We added carefully created adversarial examples to the initial training data to assess the proposed method's performance. Our experimental findings demonstrate that the proposed methodology significantly reduced the vulnerability of the defensive distillation framework to a data poison attack.

Keywords: Deep Neural Network, Denoising Autoencoder, Defensive Distillation, Adversarial attacks and Robustness, Data Poisoning

1 Introduction

The necessity of $DNNs'$ resilience against adversarial attacks has shown a growing concern as their use in real-world safety-critical systems has increased exponentially. Adversarial attacks [1] are attempts to trick $DNNs$ by making subtle alterations to the input data x. The alterations are usually designed to be imperceptible to humans but can detrimentally impact the $DNN's$ ability to make accurate decisions. Several defense strategies have been proposed to overcome this problem, including defensive distillation, which has successfully defended $DNNs$ from input perturbations ϵ in run-time settings [2]. Nevertheless, one of the drawbacks of the defensive distillation framework is that it remains susceptible to data poisoning attacks, in which adversaries aim to impair the model's performance by either maliciously altering the existing training data or inserting erroneous data entries into it. These harmful data entries could be carefully crafted to be close to the model's decision boundary, bypassing the countermeasures offered by defensive distillation. This study presents a novel method for robustifying a distilled network against data poisoning attacks by integrating a denoising autoencoder (DAE) [3] in the defensive distillation pipeline. Defensive distillation involves training two $DNNs$, the instructor model and the student model, such that the knowledge of the former is transferred into the latter, making it robust to adversarial examples and previously unseen input [1, 4]. A DAE is a type of DNN trained in an unsupervised setting to learn a latent representation of input data x, enabling it to reconstruct distorted data back to its original form. In the training phase, it is exposed to a perturbed version of x and learns to reconstruct it while filtering out noise. The aim is minimizing the reconstruction loss between the x and reconstructed input x_r as much as possible [3].

This paper is motivated by the fact that the instructor model is not immune to data poisoning adversarial attacks. However, the student model has more latitude to reject variations in the input space X because it leverages the "distilled" version of the training data, where training data is labeled with the soft labels or probability vector $F(X)$ obtained by the increase in the softmax temperature parameter T of the instructor model [2]. Thus, minimizing the instructor model's susceptibility to data poisoning attacks is pivotal for developing a reliable and robust distilled DNN. To achieve this, we designed a DAE to remove noise and reconstruct poisonous samples within the training dataset. The defensive distillation method already offers a strong resilience foundation in the test phase; combined with a DAE, the resilience against data poisoning adversarial attacks is significantly strengthened in the training phase. Moreover, our strategy considers several adversarial attack aspects, such as the attacker's access to the training data and trial-and-error techniques to access the model gradient, making it a more effective defense mechanism against such attacks.

We used the fast gradient sign method (FGSM) [5] and the iterative-fast gradient sign method (I-FGSM) [6] to evaluate the effectiveness of the proposed algorithm. We used

these adversarial examples generation algorithms to perturb a proportion of the training data.

The results in Figure 2 show that the proposed approach enables the detection and reconstruction of the majority of poisonous inputs in the training data and significantly improves the robustness of the instructor network against data poisoning attacks. The perturbations used by the $FGSM$ and $I-FGSM$ algorithms are so small that the resulting poisonous data within the training set is imperceptible to humans. The proposed approach offers a more potent protection mechanism against adversarial poisoning attacks by combining defensive distillation with a DAE. This enables the distillation algorithm to significantly mask or lower the gradient around the training data and widen the search space that attackers need to explore in order to craft adversarial examples in the input space X. The proposed approach significantly reduces the limitations and susceptibility of the defensive distillation algorithm to data poisoning attacks.

Knowledge distillation was first proposed in [4]. The author aimed to lower the computational resources required to deploy large-scale $DNNs$ on resource-constrained devices such as smartphones. Therefore, they extrapolate the probability vector or class knowledge produced by the instructor network and use it to train small networks, reducing the network's scalability without compromising accuracy and allowing deployment on resource-constrained devices.

Statistically and experimentally, Papernot et al. [2] further explore this idea as a preventive measure against adversarial inputs. Using class knowledge from the instructor network and distilling it to the student network, the authors minimize the amplitude of the student network's gradients that attackers required access to generate adversarial examples in the input space X. The authors do not necessarily transfer knowledge from large-scale networks to smaller ones; they instead use two networks of the same architecture. Moreover, they demonstrate that models developed via defensive distillation are less susceptible to adversarial inputs in runtime settings. Additionally, according to Goldblum et al., [7], while effective neural networks (NN) may be developed by transferring information from the teacher model to the student model, they may still be exposed to strong adversarial attacks in runtime scenarios. To address this limitation, they developed an Adversarially Robust Distillation (ARD), which involves creating small NNs and distilling their robustness in a larger network. The authors say this strategy performs better than the conventional defensive distillation benchmark. Previous works where the defensive distillation technique was leveraged include [8, 9, 10]. Although these strategies successfully prevent evasion attacks (run-time attacks), none of the previous authors have considered addressing adversarial poison attacks in the context of knowledge distillation. The end goal of this paper is motivated by this limitation, which makes the defensive distillation algorithm vulnerable to data poisoning attacks. As a result, we used the DAE as a filter in the knowledge distillation pipeline. Yue et al. [11] demonstrated that $DAEs$ are highly helpful in spotting and reconstructing contaminated images in the training data manifold. They apply this approach to identify and mitigate adversarial poison attacks in the federated learning setting. Other works in which DAE is used as a filter against poison data include [12, 13].

2 Mitigation of adversarial poison attacks

2.1 Defensive Distillation

As mentioned in the previous section, the technique used in defensive distillation involves training two $DNNs$ that are similar in structure; the instructor network $F\phi_1$ and the student network $F\phi_2$. In this work, a standard NN training procedure is used to train $F\phi_1$ using the original dataset, with an increase in the softmax temperature T to 5 degrees.

The softmax layer normalizes an output vector or logits $Z(X)$ of $F\phi_1$ final hidden layer into a probability vector $F(X)$ more closely aligned with the data manifold's uniform statistical distribution. This assigns a probability value to each class of the dataset for each input x. A specific neuron in the softmax layer that corresponds to a class indexed by $i \in 0..N-1$, where $(N = numberofclasses)$ calculates element i of the probability vector given by;

$$F(X) = \left[\frac{e^{Z_i \frac{(X)}{T}}}{\sum_{l=0}^{N-1} e^{Z_l \frac{(X)}{T}}} \right]_{i \in 0..N-1} \quad (1)$$

The $F(X)$ is used as a label for the inputs in the training data to train $F\phi_2$ with the same temperature value as used in $F\phi_2$. T is usually reset to its default value of 1 during testing so that the distilled network can produce more discrete output. This also discourages overly confidence in the distilled network's output and improves its generalization to new inputs.

The motivation for knowledge distillation stems from the fact that the semantic characteristics of the data that $F\phi_1$ learn are encoded in both its class probability vectors and learned weight parameters. In addition to only transferring a sample's correct class, extrapolating class knowledge from these vectors and utilizing it to label inputs in the training data for training $F\phi_2$ with the same or different architecture will supplement it with extra information about each class. Another advantage of using the class probability vector as a label to train $F\phi_2$ is that it multiplies its features that must be altered to create a successful attack. Attackers would need to modify many features in the input space X to be able to drive network $F\phi_2$ into incorrect predictions, making it considerably more challenging to construct adversarial inputs. Also, it reduces variations in X and masks or lessens the model's gradient, which an attacker is required to exploit in order to find adversarial examples around x. However, it is worth noting that standard defensive distillation is still vulnerable to data poison attacks, which motivates us to incorporate it with an DAE, and the distilled network mostly losses precision.

2.2 Denoising Autoencoder

The goal of the DAE is to learn a compressed representation of x, correct any abnormalities in the data, and reconstruct it back to its original, undistorted state with the help of the latent vector h. The design comprises an encoder network f_e and a decoder network f_d, represented as a composition of $i-th$ encoding layer $f_e^{(i)}$ and decoding layer $f_d^{(i)}$, respectively. The

first layer of f_e receives the erroneous form of data x^* from the input space X and translates it to a lower-dimensional latent vector $h^{(i)}$. The subsequent layer maps $h^{(i)}$ from the previous layer to generate more compressed latent features. This process continues until the last layer, which represents the final lower-dimensional latent representation of the input h. Conversely, the last layer of f_d takes in h and maps it to the next layer in a backward pass direction until the first layer, which maps the reconstructed features to the original input space, producing the reconstructed output image x_r.

During training, the DAE learns to minimize the reconstruction error between the initial unperturbed input data x and output reconstructed data x_r. This is accomplished by the network's ability to learn the mapping function resilient to adversarial perturbation ϵ and rebuild the x^* to their original form without compromising the most significant intrinsic statistical properties of x_r. The latent representation of the $i-th$ layer can be expressed as;

$$h^{(i)} = f_e^{(i)}(h^{(i-1)}) \wedge h^{(0)} = x \tag{2}$$

where;

$h^{(i)}$ = latent representation at the $i-th$ encoder layer.
i = the layer index of the encoder
$f_e^{(i)}$ = composition of individual encoder functions

While the final lower-dimensional latent representation of the input is expressed as;

$$h = f_{e1}(f_{e1}(f_{ei}(x_i))) \tag{3}$$

where;

h = latent representation vector
$f_{e1}(f_{e1}(f_{ei}$ = first decoder layer, second up to the last layer.
x = input of the encoder

The reconstructed output at each $i-th$ layer is expressed as;

$$x_r^{(i)} = f_d^{(i)}(x_r^{(i-1)}) \tag{4}$$

where;

$x_r^{(i)}$ = the reconstructed output at the $i-th$ decoder layer
$f_e^{(i)}$ = composition of individual decoder functions

The final reconstructed output of the decoder network is computed as;

$$x_r = f_{di}(f_{d2}(f_{d1}(h_i))) \tag{5}$$

where;

h = latent representation vector
$f_{di}(f_{e2}(f_{e3}$ = last decoder layer, up to the first layer.
x = input of the encoder

2.3 Adversarial Attacks and Robustness

Adversarial attacks are types of malicious data modification that seek to deceive ML models in their decision-making [1]. A wide range of ML-based applications are susceptible to adversarial attacks. In contrast, adversarial robustness describes the model's capacity to maintain its expected performance in the presence of malicious interruptions or adversarial attacks [14]. A model is deemed resilient if it correctly classifies $x \in X$ within or outside a range of perturbation sets defined in X. Robustness is a crucial characteristic for ML models since it guarantees their dependability in real-world applications where the x may be noisy or purposefully altered to cause misclassification [15]. In our experiment, we used the following parameters to generate adversarial perturbations for $FGSM$ and $IFGSM$; $epsilon_fgsm = 0.01, epsilon_ifgsm = 0.01, alpha = 0.01, num_iterations = 10$. The epsilon ϵ determines the magnitude of the perturbation needed to trigger the model's sensitivity to changes in the image's pixel values. The alpha α parameter controls the step size of the perturbations in each iteration of the $IFGSM$ attack. $num_iterations = 10$ indicates that the $IFGSM$ will be applied at every 10 iteration.

3 Autoencoder-based Defensive Distillation Approach

Following the methodology used in [10], the student model in our study is built to be uncertain about its prediction when a new input x_n is statistically different from the training set (i.e., discouraging overconfidence in its classification). We used the Kullback–Leibler (KL) divergence [16] to quantify the statistical differences between the x_n and the ground truth. If their statistical difference is more significant than the $P-value$, the network becomes uncertain about its prediction and returns a null value. This ambiguity is evaluated during "dropout inference," a technique in which the entire network is used for prediction while turning off the dropout layer. This approach allowed the model to offer uncertainty estimates for the predictions and estimate Bayesian inference. Analyzing the instructor model's predictions yields the uncertainty measurements needed to train the student model.

As done in [2], the $F(X)$ generated by the teacher model is not always sufficient to reduce the gradient surrounding the training data and handle variation in X, which would prevent sophisticated attacks from successfully compromising the student model. Relying only on the probability vector will make the student model classify inputs based on traditional softmax probability estimation without considering the uncertainty of its prediction.

While developing the DAE model, we used the "$OPTUNA$ library," a Pytorch-based hyperparameter optimization toolkit, to select the appropriate hyperparameters. The $OPTUNA$ uses clever methods like Bayesian optimization (BO) to search the hyperparameter dimension effectively. With a few trials, it selects the best set of hyperparameters suitable to improve the model's performance. We generate adversarial inputs from a portion of the Germany Traffic Sign Recognition Benchmark (GTSRB) dataset while the remaining portion is used as clean data. Mean squared error (MSE) loss is used to

evaluate the reconstruction error between X and x, where the goal is to minimize the statistical difference between the x^* and x. Furthermore, we use the random weight initialization strategy to avoid issues with vanishing or exploding gradients and promote quicker and more efficient convergence during training. In the training phase, both X and x^* are fed to the DAE as inputs to enable it to learn the statistical correlations between them and also to learn the noisy pattern in the input. During backpropagation, the loss function is minimized while the weights are updated using a stochastic gradient descent (SGD) optimization algorithm. The effectiveness of the trained DAE is evaluated on a different test dataset containing both clean and distorted images. A specified reconstruction threshold is set, which serves as a yardstick or decision line to verify the integrity of each data point after reconstruction. The images whose reconstruction error is above the threshold are termed adversarial inputs and are subsequently discarded before reaching the teacher model.

4 Results

Many works in the literature that use DAE randomly select the reconstruction threshold, which can lead to inconsistencies in the evaluation process and introduce additional constraints in interpreting the level of similarity between the input and reconstructed image, making it difficult to understand the behavior of the network. There is also a risk of selecting a sub-optimal threshold value that does not optimize the network's performance.

To avoid this scenario, in this work, an initial reconstruction threshold of 0.015 was chosen randomly at the beginning of the DAE network's training process. This made updating the reconstruction threshold to 0.003 easier during the evaluation phase. We used the "$inferthreshold$" function as a decision threshold to update the initial threshold. The "$inferthreshold$" function calculates the values inferred from the percentage of instances identified as adversarial in the evaluation dataset. This function is to be found in the "$alibi-detect$" outlier/adversarial detection library. Figures 1 and 2 show how the DAE network performed on the evaluation datasets, which comprised instances of only adversarial inputs and a combination of both adversarial and clean inputs, respectively.

Upon reconstruction, images with reconstruction errors greater than the updated threshold value (0.003) are labeled as adversarial inputs since the DAE network cannot restore them to their original state. Reconstructed-clean images are those with reconstruction errors below the updated threshold value. The identified adversarial inputs are subsequently eliminated before passing the dataset to the instructor model in the distillation stage. As shown in Figure 2, the green dotted circles indicate the reconstructed clean images, whereas the red dotted ones represent the adversarial images that the DAE network could not reconstruct. According to our experimental results, the average reconstruction error produced by the DAE network using the evaluation dataset was 0.008190, indicating that our designed DAE network functioned satisfactorily. Figure 3 shows the average reconstruction and validation loss at each epoch.

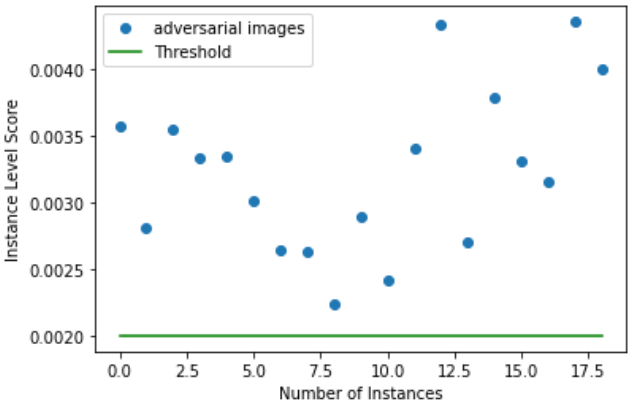

Figure 1: Illustration of the DAE's performance on only adversarial inputs during reconstruction

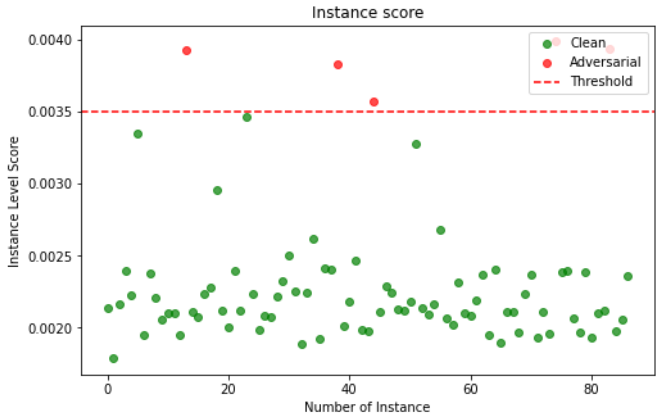

Figure 2: Illustration of the DAE's performance on the evaluation dataset during reconstruction.

The result also demonstrates that $IFGSM$ devises more robust adversarial examples than $FGSM$ because the latter only takes into account the gradient of the loss function with respect to the target $x \in X$ only once and then makes a single step along the path of the gradient around x to create an adversarial example, whereas the former takes into account the gradient of the loss function with respect to the target $x \in X$ at every iteration and keeps updating the perturbation in X in the direction of the gradient's sign around the target $x \in X$ which results in generating more robust adversarial examples.

The reconstructed images are sent to the instructor model, trained with a softmax temperature of $T = 5$. This produces soft labels to annotate the new dataset, which we then use to train the student model (distilled network). The instructor model's total accuracy throughout training is 99.89%, with an average loss of 0.11%. The student model also performed well by correctly classifying adversarial inputs in the test dataset with an accuracy rate of 76.09%. Although this precision might not be particularly excellent in other attack scenarios, the design is incredibly robust to $IFGSM$ and $FGSM$ adversarial attacks.

5 Conclusion

The use of a DAE network in combination with defensive distillation has proven to be an effective method in enhancing

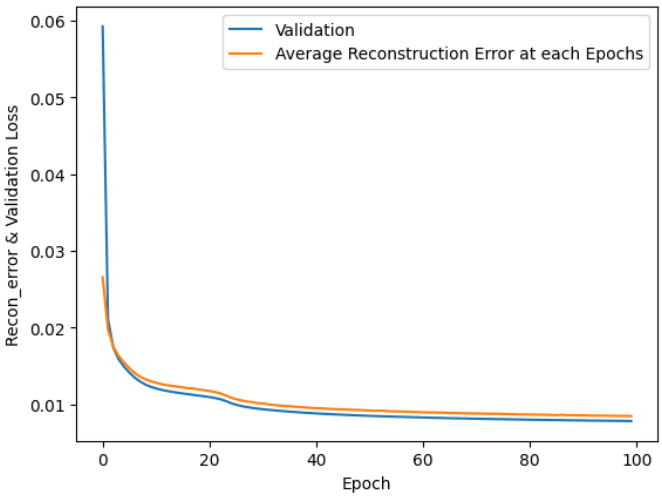

Figure 3: Illustration of the DAE's performance on the evaluation dataset during reconstruction.

the robustness of a distilled network against both poisoning and evasion adversarial attacks. It was also shown that the "alibi-detect" library helped the DAE network detect and get rid of adversarial inputs in the training dataset. By updating the reconstruction threshold during the evaluation phase, the DAE network was able to accurately identify inputs as either adversarial or clean. The use of $IFGSM$ resulted generating in stronger adversarial examples than $FGSM$, highlighting the importance of considering the gradient of the loss function with respect to the $x \in X$ at every iteration. The student model, trained with soft labels produced by the teacher model, demonstrated excellent robustness against adversarial attacks. Based on the results, the proposed approach could be a valuable addition to the arsenal of techniques available to improve the reliability in ML-driven systems.

6 Acknowledgments

This work was supported by the LASIGE Research Unit (ref. UIDB/00408/2020 and ref. UIDP/00408/2020), and by the European Union's Horizon 2020 research and innovation programme under grant agreement No 957197 (VEDLIoT project).

References

[1] Y. Chen, M. Zhang, J. Li, and X. Kuang, "Adversarial attacks and defenses in image classification: A practical perspective," in *2022 7th International Conference on Image, Vision and Computing (ICIVC)*, pp. 424–430, IEEE, 2022.

[2] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE symposium on security and privacy (SP)*, pp. 582–597, IEEE, 2016.

[3] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *arXiv preprint arXiv:2003.05991*, 2020.

[4] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *stat*, vol. 1050, p. 9, 2015.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[6] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

[7] M. Goldblum, L. Fowl, S. Feizi, and T. Goldstein, "Adversarially robust distillation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3996–4003, 2020.

[8] M. Kuzlu, F. O. Catak, U. Cali, E. Catak, and O. Guler, "Adversarial security mitigations of mmwave beamforming prediction models using defensive distillation and adversarial retraining," *International Journal of Information Security*, pp. 1–14, 2022.

[9] E.-C. Chen and C.-R. Lee, "Ltd: Low temperature distillation for robust adversarial training," *arXiv preprint arXiv:2111.02331*, 2021.

[10] N. Papernot and P. McDaniel, "Extending defensive distillation," *arXiv preprint arXiv:1705.05264*, 2017.

[11] C. Yue, X. Zhu, Z. Liu, X. He, Z. Zhang, and W. Zhao, "A denoising autoencoder approach for poisoning attack detection in federated learning," *IEEE Access*, vol. 9, pp. 43027–43036, 2021.

[12] A. Kascenas, N. Pugeault, and A. Q. O'Neil, "Denoising autoencoders for unsupervised anomaly detection in brain mri," in *International Conference on Medical Imaging with Deep Learning*, pp. 653–664, PMLR, 2022.

[13] M. Tripathi, "Facial image denoising using autoencoder and unet," *Heritage and Sustainable Development*, vol. 3, no. 2, p. 89, 2021.

[14] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim, "An analysis of adversarial attacks and defenses on autonomous driving models," in *2020 IEEE international conference on pervasive computing and communications (PerCom)*, pp. 1–10, IEEE, 2020.

[15] F. Wang, C. Zhang, P. Xu, and W. Ruan, "Deep learning and its adversarial robustness: A brief introduction," in *HANDBOOK ON COMPUTER LEARNING AND INTELLIGENCE: Volume 2: Deep Learning, Intelligent Control and Evolutionary Computation*, pp. 547–584, World Scientific, 2022.

[16] F. Raiber and O. Kurland, "Kullback-leibler divergence revisited," in *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pp. 117–124, 2017.

Exploring Trade-offs in Explainable AI

Dene Brown, Glenn Hawe

Ulster University, 2-24 York Street, Belfast BT15 1AP; email: dene.brown@sysada.co.uk, gi.hawe@ulster.ac.uk

Abstract

Machine Learning (ML) models are increasingly used in systems that involve physical human interaction or decision-making systems that impact human health and safety. Ensuring that these systems are safe and reliable is an important topic of current AI research.

For many ML models it is unclear how a prediction (output) is arrived at from the provided features (input). Critical systems cannot blindly trust the predictions of such "black box" models, but instead need additional reassurance via insight into the model's reasoning. A range of methods exist within the field of Explainable AI (XAI) to make the reasoning of black box ML models more understandable and transparent.

The explanations provided by XAI methods may be evaluated in a number of (competing) ways. In this paper, we investigate the trade-off between selected metrics for an XAI method called UnRAvEL, which is similar to the popular LIME approach. Our results show that by weighting the terms within the acquisition function used in UnRAvEL, different trade-offs can be achieved.

Keywords—Machine Learning, Explainable AI, Gaussian Process, LIME, UnRAvEL.

1 Introduction

There is increasing demand for machine learning (ML) models to be used within critical systems. One of the issues holding this back is the understanding of the decision making, and the proof of reliability of these models. Current high performing ML models are commonly complex black box models. With these models it is difficult to understand how the outputs of these are achieved, and the importance of each feature in determining the output.

Figure 1 shows a typical model, which takes as input a vector X of input features (x_1, x_2, …, x_n) and generates a prediction f(X) that depends on X. But as we have little or no insight into the reasoning of each prediction, the model is described as being "black box".

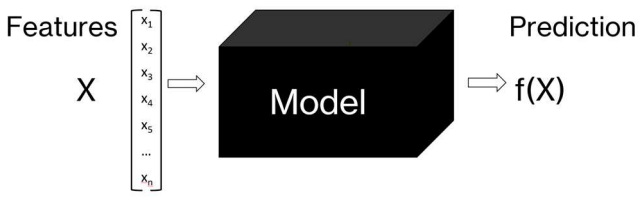

Figure 1. A typical black box model with opaque reasoning

Explainable AI (XAI) is a discipline that is attempting to provide insight into the reasoning of black box models [1]. One common XAI approach is to identify the contribution of each feature in the determination of a model's prediction.

One of main XAI approaches is LIME (Local Interpretable Model-agnostic Explanations) [2]. In the original LIME paper there is a simple experiment to demonstrate why XAI is needed.

Figure 2. An image from the LIME [2] paper. The left image is the image presented as test data for the model, it is of a husky with a background of snow. The image on the right shows the areas of the image identified by LIME that were the most important in the prediction

In the experiment, Figure 2, a model is built to distinguish between images of wolves and images of huskies. The model is built, and it achieves a good performance. However, it has been deliberately trained to be a flawed model. All the training images of wolves presented to the model at training have snow in the background, and all images of huskies do not have snow. The model is then presented with the test images, and all test images with snow in the background are predicted to be a wolf, and those with no snow in the background are predicted to be a husky, regardless of which species the image contains.

This experiment demonstrates why XAI is needed if ML models are going to be accepted as being reliable. Even if a model scores well in testing we are not certain it will be reliable 'in the field', as it may rely on dubious reasoning which the test data failed to expose.

The central idea behind LIME is to provide an interpretation for the prediction of a point via a local "explainer" model constructed around the point. Explainer models are easier to interpret and provide relative scores for the importance of input features in the determination of the output.

There have been numerous extensions to LIME [3][4][5] attempting to improve the explanations provided through better sampling points for the explainer model. One such extension is UnRAvEL (Uncertainty driven Robust Active Learning Based Locally Faithful Explanations) [6] which

uses a novel Gaussian-Process based acquisition function called FUR (Faithful Uncertainty Reduction) to select points for the explainer model.

In early XAI work there was an emphasis for explainable models to provide feature importance scores to potential users and domain experts. This is intended to give the experts confidence that the black box model is making predictions for the right reasons, and to persuade them to trust the model outcomes.

Expanding the subjective concepts of confidence and trust to that of reliability will be a challenging task. One potential route is in the use of recognised metrics to give objective performance measurements of explainable models. A recent review [7] of XAI metrics has identified 12 metrics that consistently appear in papers on XAI. In the review they are given the name the "Co-12" properties (because the properties all have names beginning with "co").

In this paper we aim to demonstrate that we can use UnRAvEL to generate a set of explanations that represent a trade-off in specific Co-12 properties. In particular, we propose that by weighting the two terms within the FUR acquisition function used within UnRAvEL, different points will be sampled for the explainer model and thus different explanations provided. By controlling the weights, we can control (and ultimately optimize) the trade-off in Co-12 properties for an explanation.

The remainder of this paper is structured as follows. In Section 2 we introduce LIME and UnRAvEL. In Section 3 we summarize the Co-12 properties used to evaluate explanations. In Section 4 we present our methodology for generating different explanations using UnRAvEL. In Section 5 we present the results of our experiments and in Section 6 we summarize our findings.

2 Local Explainable Models

LIME

Figure 1 showed a black box model with its prediction f(X) being a function of the input features. LIME uses a local linear model to determine the importance of the input features. The prediction function f(X) for a linear model is given in Equation 1.

$$\boldsymbol{f(X) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n}$$

Equation 1

In a linear model each input feature is multiplied by a weight to calculate the prediction. This is a simple model to interpret, as the magnitude of a feature's weight determines its importance in the prediction. LIME uses linear models to locally interpret single points.

Figure 3 shows a 2-D feature space with features x_1 and x_2. A black box model in this feature space makes a prediction of either pink or blue. It appears to be complex model whose output is not easily interpretable.

The principle behind LIME is to focus on a single point, which is identified by the black square in Figure 3. In Figure 4 we zoom in on this point and create additional points

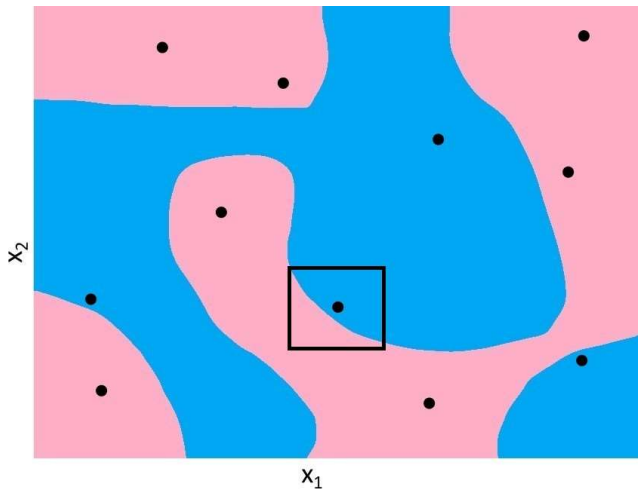

Figure 3. Two-Dimensional feature space with complex model prediction

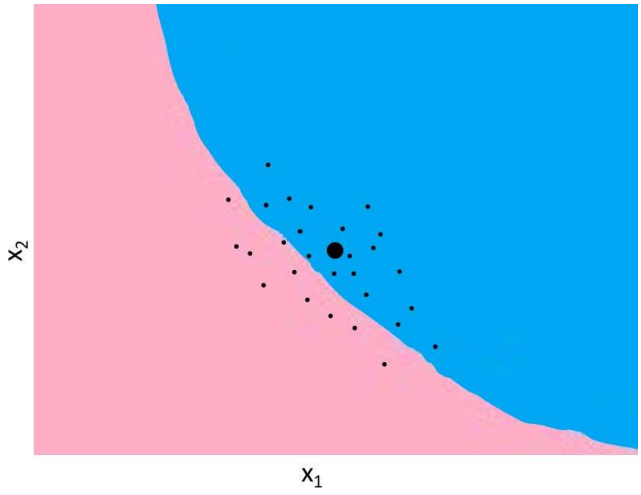

Figure 4. LIME creates new sample points around the point being explained

around the point being explained. These points are generated from perturbations of the original point, and their predictions are given by the points being input to the black box model.

Using these additional points and their predictions a linear model is created to model behaviour around the point.

From Equation 1 we can see that the larger the weight for a feature the greater effect it has on the outcome. Therefore, weights in the local linear model give the importances of the features in the locality.

Figure 5 shows the predictions and feature importances for two points from the diabetes dataset use in this project. The feature importances were generated by a LIME model to explain those points. The importance corresponds to the weights of the linear model. Weights in a linear model can be positive (orange) or negative (blue).

UnRAvEL

UnRAvEL uses a Gaussian Process [8] to create a local interpretable model. Figure 6, Figure 7 and Figure 8 show the Gaussian Process (GP) modelling a 1-D Forrester function over a number if iterations. The Original function

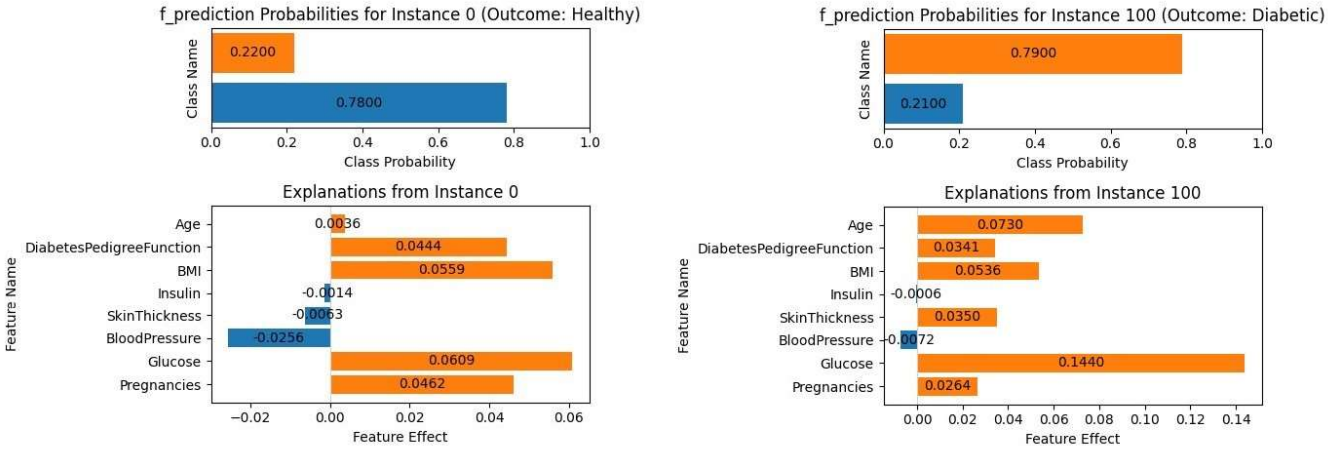

Figure 5. Predictions and feature importances for two instances of the diabetes dataset

is the blue line, the GP model is the red line, the orange lines represent the uncertainty in the GP prediction. In these figures the GP only models the function in the range 0.1 to 0.7, it does not select points outside this range.

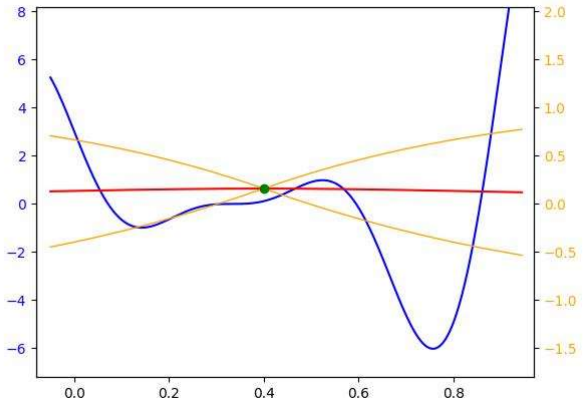

Figure 6. Gaussian Process modelling a 1-D Forrester function at first iteration

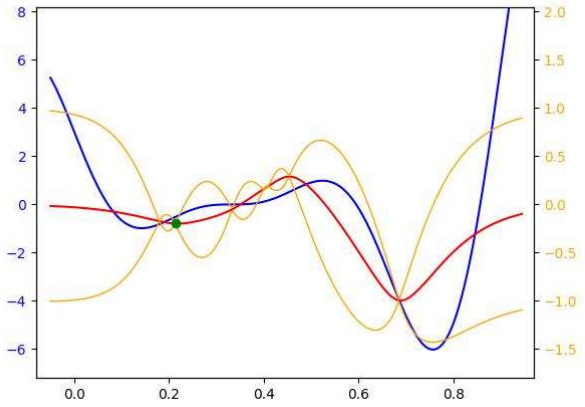

Figure 7. Gaussian Process modelling a 1-D Forrester function at 6th iteration

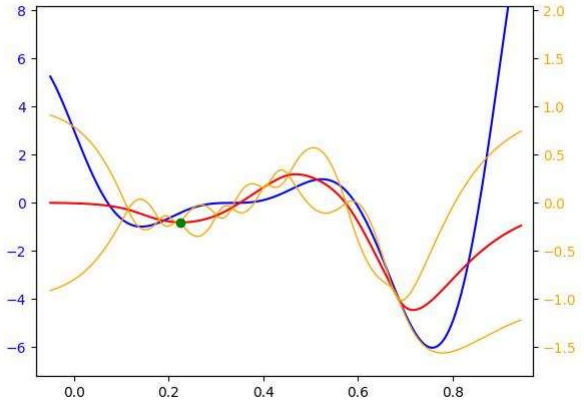

Figure 8. Gaussian Process modelling a 1-D Forrester function at final iteration

Gaussian Processes use an iterative approach to create models, they select a new point to add to the model at each iteration. As points are added to the model its predictions become more accurate and the uncertainties diminish. We reach a point where adding more points does not significantly improve the model.

Figure 6 is the first iteration when only a single point, the dataset point being explained, has been added to the model. Figure 7 is when five addition points have been selected, and Figure 8 is the final iteration, when the model is complete.

A Gaussian Process is significantly more complex than linear models, but we can obtain feature importances from the model, which is essential in XAI.

Each iteration of a Gaussian Process model uses an acquisition function to select the next point for the model, UnRAvEL uses a novel acquisition function called FUR.

The choice of acquisition function significantly affects the performance of the GP model, and investigating UnRAvEL and Gaussian Process models was the route the project took rather than considering LIME. The main part of the project involved reconfiguring the FUR acquisition function to improve model performance.

3 Quantifying XAI Performance

Table 1. Model Properties used to Access Performance

Property	Description
Correctness	Measures how faithfully the explainer model replicates the behaviour of the black box model. E.g. are the same features as important to both models?
Output-Completeness	This property looks at output of the explainer model compared to the black box model and measures the accuracy of the explainer model predictions.
Reasoning-Completeness	Measures how much the black box and explainer models are in step with their reasoning. E.g. do changes to input features have the same effect on the predictions of both models?
Consistency	The measures the consistency of explainer models when modelling the same feature space with each being based on different black box models.
Continuity	The measures the resilience of explainer models to perturbations of the features.
Jaccard Similarity	Measures the consistency of feature importance across all points in feature space.
Calibration	Calibration is a measure of well a model makes it predictions. It gauges how overconfident or under confident a model is performing.

The paper from Meike Nauta et al [7] is a detailed and far-reaching literature review of the current methods used to assess the performance of XAI models. The paper identifies twelve properties (referred to as the Co-12 properties) of XAI models that appear repeatedly in the reviewed papers. The paper looks to define common methods that are used across the reviewed set of papers that can be used determine these properties.

Five Co-12 properties have been chosen for this project, ones that can be objectively determined and can provide a measure of model performance. These are described in Table 1. There are two additional important metrics calculated that are not Co-12 properties. These are Jaccard score and calibration.

4 Methodology

FUR and FUR_W

In the UnRAvEL process when a data point is selected for explaining a Gaussian process model is created around that point. Addition points are incrementally added to the Gaussian process model, and the selection of these points is determined by the FUR acquisition function.

In the process the first point added to the GP is the point that is being explained. To determine the next and subsequent points the FUR [6] acquisition function is used. Acquisition functions use the values of the feature space, X = (x_1, x_2, …, x_n), as input. A search is made in the feature space around the point to determine the point that has the maximum value of FUR. This point is added to the model for the next iteration. The selection of the next point can be written:

$$X_{next} = \arg\max(T_1 + T_2)$$

Equation 2

Where:

$$T_1 = \|X - X_0 - \bar{\sigma}\epsilon/\log(n)\|_2$$

$$T_2 = \sigma_n(X)$$

- X_0 is the original point.
- $\bar{\sigma}$ is the mean of the standard deviation in the training data.
- n is the iteration number.
- ϵ is a noise factor to move the selected point away from X_0. Dividing this noise factor by log(n) reduces this noise as the iterations progress, this brings selected values closer to X_0.
- $\sigma_n(X)$ is the uncertainty in the model prediction at point X.

Figure 9 shows the values of terms T_1 and T_2 along with their combined value in FUR (right axis) in a feature space $0 \leq x \leq 1$, where the 'model' is the 1-D Forrester function (left axis).

The two terms T_1 and T_2 in the FUR acquisition function have different aims. T_1 is intended to make the selected points be located around the vicinity of the original point, initially from a wide range, but this range is reduced during later iterations, by dividing by log(n).

T_2 selects points where the uncertainty of the Gaussian process is greatest, with the aim of reducing this uncertainty.

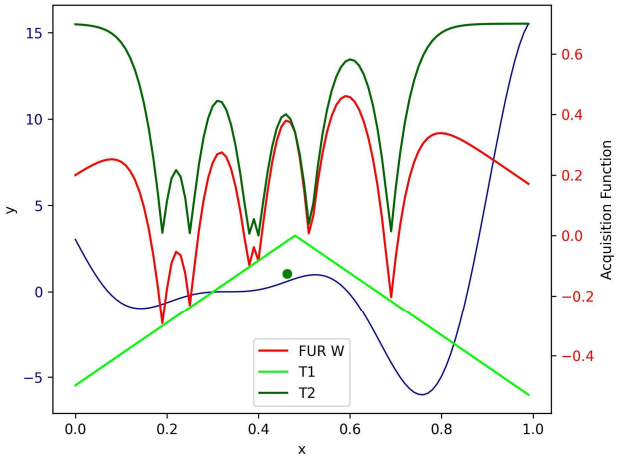

Figure 9. 1-D Forrester Function showing the terms T_1, T_2 from the FUR acquisition function

This project is to investigate these competing terms in the FUR acquisition function by introducing weights for each of the terms. We create a new acquisition function FUR_W, a weighted version of FUR. This is defined in Equation 3 below, where w is a weight between 0 and 1:

$$X_{next} = \arg\max(-wT_1 + (1-w)T_2)$$

Equation 3

Running Experiments

To investigate the effects of weighting the terms in FUR_W several datasets were used in each experiment.

Each dataset was used to train two black box models, a neural-network and random forest. Explainer models were then created to model and explain test points from the dataset.

A number UnRAvEL explainer models were generated for each black box model. Each explainer model was generated using a different weight value in its FUR_W functions. The weights used were 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95). The effect on the terms T_1 and T_2 in the FUR_W acquisition function can be seen in Figure 10 where T_2 is dominant, and Figure 11 where T_1 is dominant.

The performance of the explainer models was then measured using the selected properties in Table 1.

5 Results

One very noticeable property of the UnRAvEL explainer models is how accurate they are in their outcome predictions (the Output-Completeness property). Their predictions match closely the black box predictions, as shown in Figure 12. From this graph it appears the explainer model predictions are indistinguishable from those of the black box models.

A possible reason for this is that and there could be of over training in the explainer models. This is good when the explainer models are making predictions, but it could mean other properties of the explainer model will not score as well.

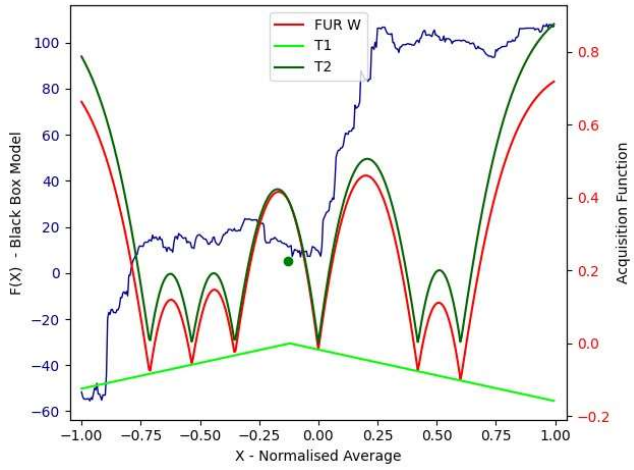

Figure 10. FUR_W acquisition function for weights of 0.1 for T1 and, 0.9 for T2. The dataset is synthetic regression data.

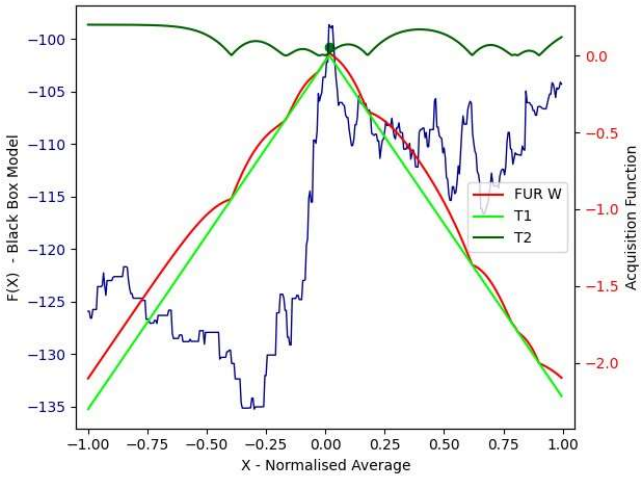

Figure 11. FUR_W acquisition function for weights of 0.8 for T1 and, 0.2 for T2. The dataset is synthetic regression data.

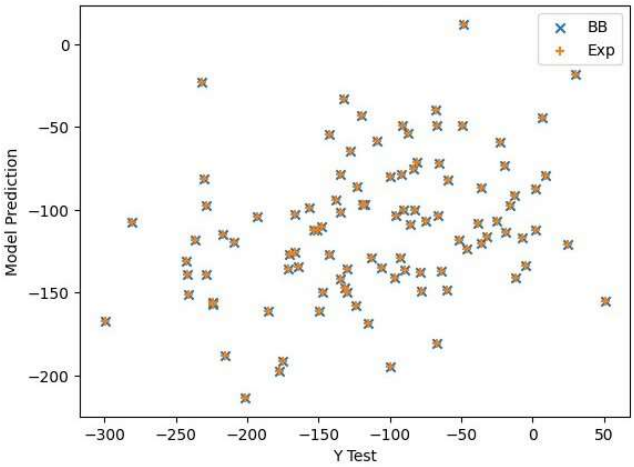

Figure 12. Comparison of predicted outcomes from black box model and explainer model for synthetic regression dataset.

Figure 13 shows the measurement of the Consistency metric for all datasets. This measures how well the predictions of two explainer models are correlated when they are trained on different types of black box models trained using the same dataset.

Other than the diabetes dataset there is high correlation between models, particularly for higher weight values. This indicates that the explainer models are modelling the dataset well at the individual points.

Unfortunately, many of the results resemble the graph in Figure 14. They show no clear patterns for the Co-12 properties as a result of varying the weights in the FUR_R function.

6 Conclusion

The models rate very highly in replicating the predictions of the black box models but less so on other metrics. As previously stated, the closeness in replicating black box model predictions may be a sign of overtraining the explainer model, the uncertainty values associated with the predictions are also very small (around 1e-5), which also indicates overtraining.

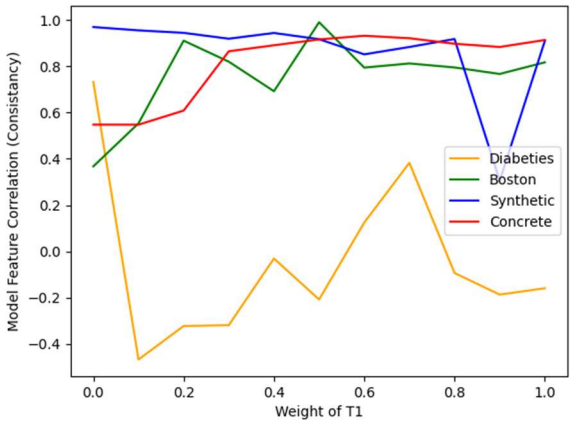

Figure 13. Consistency metric for all datasets.

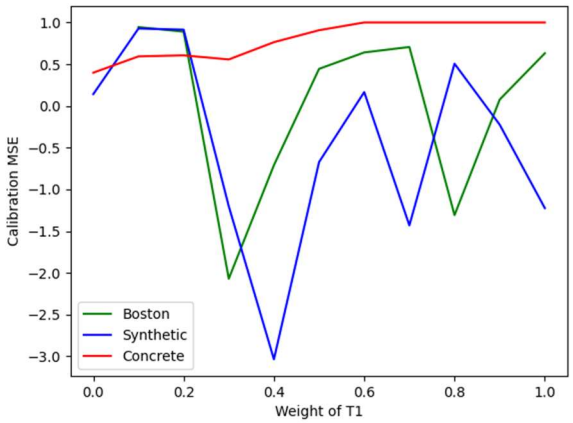

Figure 14. Correlation metric for all datasets

It would be interesting to reduce the number of points that are used to train the model to see if the other metrics could be improved upon without too great a loss of accuracy.

No results contain any clear patterns of behaviour that can be attributed to the settings of the weights. Drawing conclusions from the results on the effect of changing the weight of the terms in FUR_W is difficult.

Despite this apparent lack of success there have been benefits to the project. Looking at the Co-12 properties and how to implement their determination has been challenging. This work by itself could be the basis for future research, in developing better algorithms to calculate the Co-12 properties.

References

[1] T. Speith, “A Review of Taxonomies of Explainable Artificial Intelligence (XAI) Methods”, *FAccT '22: 2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 2239-2250, 2022.

[2] M. T. Ribeiro, S. Singh, C. Guestrin, “Why Should I Trust You?": Explaining the Predictions of Any Classifier”, *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135-1144, 2016.

[3] X. Zhao, X. Huang, V. Robu, D. Flynn, “Baylime: Bayesian local interpretable model-agnostic explanations”, in *Uncertainty in artificial intelligence*, pp. 887-896, PMLR, 2021.

[4] M. R. Zafar, N. M. Khan, “DLIME: a deterministic local interpretable model-agnostic explanations approach for computer-aided diagnosis systems”, arXiv:1906.10263, 2019.

[5] M. Shankaranarayana, D. Runje, “ALIME: Autoencoder based approach for local interpretability”, in *IDEAL*, Springer, 454–463, 2019.

[6] A. Saini, R. Prasad, “Select Wisely and Explain: Active Learning and Probabilistic Local Post-hoc Explainability”, *AIES '22: Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 599-608, 2022.

[7] M. I. Nauta et al., “From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI”, *ACM Computing Surveys 55* (13) Article 295, pp. 1-42, 2023.

[8] C. E. Rasmussen, C. K. I. Williams, “Gaussian Processes for Machine Learning”, *The MIT Press*, 2006. ISBN 0-262-18253-X

Towards a Catalog of Prompt Patterns to Enhance the Discipline of Prompt Engineering

Douglas C. Schmidt, Jesse Spencer-Smith, Quchen Fu, and Jules White
Dept. of Computer Science, Vanderbilt University; email: {douglas.c.schmidt, jesse.spencer-smith, quchen.fu, jules.white}@vanderbilt.edu

Abstract

The rapid advent of Large Language Models (LLMs), such as ChatGPT and Claude, is revolutionizing various fields, from education and healthcare to the engineering of reliable software systems. These LLMs operate through "prompts," which are natural language inputs that users employ to query and leverage the models' capabilities. Given the novelty of LLMs, the understanding of how to effectively use prompts remains largely anecdotal, based on isolated use cases. This fragmented approach limits the reliability and utility of LLMs, especially when they are applied in mission-critical software environments. To harness the full potential of LLMs in such crucial contexts, therefore, we need a systematic, disciplined approach to "prompt engineering" that guides interactions with and evaluations of these LLMs.

This paper provides several contributions to research on LLMs for reliable software systems. First, it provides a holistic perspective on the emerging discipline of prompt engineering. Second, it discusses the importance of codifying "prompt patterns" to provide a sound foundation for prompt engineering. Third, it provides examples of prompt patterns that improve human interaction with LLMs in the context of software engineering, as well as other domains. We conclude by summarizing ways in which prompt patterns play an essential role in providing the foundation for prompt engineering.

1 Introduction

Large language models (LLMs) [1, 2] with conversational interfaces, such as ChatGPT [3], are generating and reasoning about art, music, essays and computer programs. Startups using LLMs are attracting significant funding [4] and existing software is being enhanced using LLMs. The rapid uptake of these tools underscores the transformational—and disruptive—impact LLMs are having on society, research, and education. However, little disciplined knowledge about chat-adapted LLMs, their capabilities, and their limitations exist.

LLMs provide new computational models with unique programming and interaction paradigms and greatly expanded capabilities. Anyone with an Internet connection and web browser can instantly access vast intelligent computational abilities, such as explaining complex topics; reasoning about diverse data sets; designing, implementing, and testing computer software; and simulating complex systems. LLMs are programmed through prompts, which are natural language instructions provided to the LLM [5], such as asking it to answer a question or write an essay. These common examples of prompts, however, do not reveal the much more sophisticated computational abilities LLMs possess.

Harnessing the potential of LLMs in productive and ethical ways requires a systematic focus on *prompt engineering*, which is an emerging discipline that studies interactions with—and programming of—emerging LLM computational systems to solve complex problems via natural language interfaces. We contend that an essential component of this discipline are *Prompt Patterns* [6], which are similar to software patterns [7], but focus on capturing reusable solutions to problems faced when interacting with LLMs. Such patterns elevate the study of LLM interactions from individual *ad hoc* examples, to a more reliable and repeatable engineering discipline that formalizes and codifies fundamental prompt structures, their capabilities, and their ramifications.

This paper presents portions of our ongoing efforts to codify a catalog of domain-independent patterns to show the need for more research on prompt patterns and prompt engineering. We present these patterns in the context of engineering software-reliant systems, but they are applicable in many other domains.

The remainder of this paper is organized as follows: Section 2 gives an overview of the emerging discipline of prompt engineering; Section 3 describes portions of a catalog of prompt patterns that we are codifying; and Section 4 presents concluding remarks and lessons learned from our work on prompt patterns for prompt engineering thus far.

2 Towards a Discipline of Prompt Engineering

This section gives an overview of prompt engineering, focusing on its definition, value for both computer science (CS) and non-CS professionals, and the need for a holistic approach. For CS professionals we emphasize prompt engineering's role in enhancing AI interactions and accelerating prototyping, whereas for non-CS professionals it serves as a gateway to computational problem-solving without requiring traditional programming skills. Lastly, we advocate for considering the same type of quality attributes for prompt engineering as we do for software engineering.

2.1 What is Prompt Engineering?

Prompt engineering is the science and art of designing, formatting, and optimizing conversational prompts to better guide the discourse with AI or machine learning models. It involves crafting of stimuli or instructions to evoke specific responses from AI systems. With the increased use of AI platforms that respond to user queries, establishing a discipline of prompt engineering has become increasingly important.

The scope of prompt engineering encompasses a wide range of domains, including AI chatbots, AI customer service agents, voice-first applications, and other AI interaction interfaces. It plays a crucial role in tuning the model performance, enhancing the quality of interaction, and achieving user satisfaction. It spans both understanding the technical capabilities of AI models and the nuances of human communication.

Users of AI models need to understand the strength and weaknesses of AI models they interact with and should hone their creative capability to formulate prompts that evoke the desired response. Prompt engineering thus provides a bridge between the increasingly sophistication of AI models and the need for human-like interactions that appeal to users.

2.2 The Value of Prompt Engineering for Computer Science (CS) Professionals

For CS professionals, prompt engineering offers advantages that extend beyond the basics of code generation and code summarization. Programmers are generally proficient in coding and well-versed in the syntax and semantics of conventional programming languages, such as Java, C/C++, or Python. However, prompt engineering introduces a new paradigm that enables rapid prototyping and concept testing without the need to write code manually. It thus serves as a complementary tool that can expedite iterative and incremental development processes, enabling software engineers to rapidly sketch out algorithms, models, or systems using natural language. This capability accelerates the transition from idea to implementation, thereby saving time and effort.

Prompt engineering can also enhance software quality and coding practices, *e.g.*, by serving as a 'first-pass filter' to evaluate the feasibility of algorithms or system architectures before delving into implementation details. Articulating complex computational problems in natural language can enable developers to identify potential pitfalls or inefficiencies more rapidly. Emphasizing clarity in initial stages of the software development life-cycle allows a broader range of stakeholders (including architects, systems engineers, produce managers, and end-users) to create more robust, efficient, and maintainable software-reliant systems over the life-cycle by avoiding costly mistakes and architectural flaws that are cumbersome and costly to rectify later.

Prompt engineering can be an effective tool for collaborative work within the CS community. Prompts can be collected into libraries of "prompt templates", thereby simplifying the sharing of algorithms, ideas, and problem-solving approaches without needing to understand the detailed intricacies of software repositories. Since prompts are often more easily understood and modified by a range of (non-developer) stakeholders, they facilitate more inclusive and interactive development environments. For example, team members can propose modifications, tune parameters, or even reimplement subsystems without the steep learning curve often associated with understanding lower-level programs.

Prompt engineering also benefits educators and mentors within the field of computer science. Educators can use it to introduce complex computational concepts to students in a more intuitive and accessible manner. Similarly, it can enable more advanced students or junior developers to transition from conceptual understanding to practical application. By offering a natural language-based approach to problem articulation and solution, prompt engineering acts as an educational accelerator, easing the path from theory to practice.

2.3 The Value of Prompt Engineering for Non-CS Professionals

Prompt engineering can be viewed as a form of "programming" via natural language, which helps to democratize the application of computational problem-solving across a range of disciplines and professions. When used effectively, this approach can bypass conventional barriers set by the need to learn conventional programming languages, such as Java, C/C++, or Python. Mastering these structured programming language traditionally involved understanding their syntax and semantics, which can incur a daunting and time-consuming learning curve for non-CS professionals.

Moreover, the primary interest professionals in fields like chemistry, biology, physics, the social sciences, and the humanities often lies not in becoming programming experts, but rather in leveraging computational resources to advance their research or solve domain-specific problems. In such contexts, prompt engineering helps to shift the focus from mastering coding to mastering problem-solving. Using natural language as the medium reduces barriers to entry, thereby allowing a broader audience to employ computational tools in their respective domains more effectively.

The potential impact of shifting from conventional programming to problem-solving with LLMs is significant. Computation today is often limited to those with specialized training in CS or programming. With prompt engineering, however, experts in diverse fields ranging from analysis of ancient documents to radiology can harness the power of computational methods to drive innovation and discovery. In particular, they can articulate complex problems using familiar terminology and get computational assistance without the learning curve associated with conventional programming.

Prompt engineering is particularly relevant in interdisciplinary work, where insights from multiple fields are crucial. In these contexts, it serves as a bridge that facilitates a more holistic approach to problem-solving, unifying various areas of expertise under the umbrella of computational capability.

2.4 Towards a Holistic View of Prompt Engineering

The notion that prompt engineering is merely a passing trend [8], soon to be eclipsed by increasingly sophisticated LLMs, is a simplistic and short-sighted perspective. This view reduces prompt engineering to a set of "tricks" designed to navigate the current limitations of LLMs. However, this

view overlooks the inherent complexities and nuances of natural language, which necessitates a systematic approach to interaction. Unlike traditional programming languages, natural language lacks rigorously defined semantics, requiring a disciplined method like prompt engineering to ensure the effective use of LLMs in software-reliant systems.

Far from being a stopgap measure, prompt engineering should be integrated holistically into all phases of the software development life-cycle. In traditional software development, professionals address a broad range of considerations beyond just coding, including requirements specification, configuration management, testing, and version control. In much the same way, the discipline of prompt engineering must also address these considerations, especially in mission-critical systems where failure is not an option.

A focus on quality attributes across the life-cycle is essential for the broader application of LLMs in robust, long-lived software-reliant systems. Current uses of LLMs are often localized and tactical, not integrated into systems intended to endure for decades. As LLMs evolve, prompts that were once reliable may no longer function as intended. The same diligence applied to traditional software engineering—centered on maintainability, reliability, and compatibility—must therefore be applied to the domain of prompt engineering.

Failing to adopt a comprehensive view of prompt engineering risks limiting the application of LLMs to trivial or short-term projects. To unlock the full potential of these advanced models in shaping future software-reliant systems, therefore, a focus on quality attributes and a holistic methodology are not just advantageous, they are essential. This dependency underscores the need for a more mature and systematic discipline of prompt engineering that goes beyond mere prompt crafting and becomes an integral part of modern software engineering in the age of LLMs.

3 Towards a Catalog of Prompt Patterns

This section builds upon and briefly summarizes our prior work on prompt patterns [6]. Prompt patterns use a similar format to classic software patterns, with slight modifications to match the context of output generation with LLMs.

Organizing a catalog of prompt patterns into easily digestible categories helps users interact with and program LLMs more effectively. Table 1 outlines the classification of the patterns we implemented and tested with ChatGPT discussed in this paper. As shown in this table, there are four categories of prompt patterns in the classification framework presented in this paper: *Software Requirements*, *Interaction*, *Prompt Improvement*, and *Error Identification*. The *Software Requirements* patterns are a specialized subset, as discussed next.

Table 1: Classifying Prompt Patterns

Pattern Category	Prompt Pattern
Software Requirements	*Requirements Elicitation Facilitator* *Unambiguous Requirements Interpreter*
Interaction	*Game Play*
Prompt Improvement	*Question Refinement*
Error Identification	*Reflection*

3.1 Patterns as an Abstraction for Derivation of New Patterns

A benefit of patterns is that they can serve as an abstraction for specialization and adaptation to different domains. In prior work [6] we codified the *Flipped Interaction* pattern, which directs the LLM to ask the user questions until it obtains enough information to achieve a particular goal. The structure of this pattern is as follows:

Contextual Statements
I would like you to ask me questions to achieve X
You should ask questions until this condition is met or to achieve this goal (alternatively, forever)
(Optional) ask me the questions one at a time, two at a time, etc.

The *Flipped Interaction* pattern forms a broader abstraction that can be tailored and specialized to address different aspects of requirements gathering in software engineering, thereby leading to the creation of other, more context-specific patterns. Software engineering requirements' needs often call for such specialized patterns, which extend the parent pattern by integrating new attributes and focusing on specific tasks, such as requirements elicitation, requirements ambiguity resolution, requirements discrepancy analysis, and requirements traceability. Similar to how a super class in object-oriented programming can be inherited and specialized to cover different uses-cases, prompt patterns can act as abstractions for derivations of new prompt patterns. In this case, *Flipped Interaction* acts as a super pattern for flipped interactions focusing on requirements elicitation and management.

Sections 3.2 and 3.3 introduce patterns derived from the *Flipped Interaction* pattern that address more specific areas within requirements elicitation and management, thereby transforming the interaction's goal to achieve more specificity. Section 3.2 describes the *Requirements Elicitation Facilitator* pattern, which refines requirements through focused interaction, and Section 3.3 describes the *Unambiguous Requirements Interpreter* pattern, which reduces ambiguity in requirements through targeted questioning.

3.2 Requirements Elicitation Facilitator

3.2.1 Intent & Context

The *Requirements Elicitation Facilitator* pattern enables an LLM to cooperatively ask questions or propose scenarios, which motivate users to bring forward their implicit requirement expectations. The objective is to get sufficient insight into the subjective nature of their requirements and fill up possible communication gaps. This pattern is particularly beneficial when dealing with broad context applications.

In a context whereby many details are implicit or merely touched upon, using the *Requirements Elicitation Facilitator* pattern will guide an LLM into a systematic dialogue with the users, thereby aiding in the step-wise refinement of the requirements. The discussion focuses on ensuring clarity of notions, resolving contradictions if any, and communicating the intended functionalities/schema.

3.2.2 Motivation

The process of requirements elicitation is often tedious and time-consuming and filled with uncertainties. The process involves, among many other things, capturing a description of what the system should do, the desired behavior in different states, and operational constraints. This process is iterative, requiring involvement from different stakeholders and significant conversation to ensure that all parties are fully expressing their needs and vision for the system. Language model-driven approaches can facilitate this process by helping to direct the questioning and discussion to gather, analyze, and validate the requirements, thereby saving effort and time. This process is one of the most critical stages in software development, since it forms the basis on which the proposed system is built.

3.2.3 Structure & Key Ideas

Typical contextual statements for this pattern include the following:

Contextual Statements
I am creating the requirements for a software system Y using requirement format Z.
(Optionally) I am working on requirements for aspect Q of the system.
Ask me questions to help generate requirements for the system.
After each question, 1) based on my answer, generate the requirement in format Z and then 2) ask me the next question.
Keep asking me questions until stop condition V.
Ask me the first question.

3.2.4 Example Implementation

Examples of prompts that use the *Requirements Elicitation Facilitator* pattern might include:

> "I am creating the requirements for a web application that allows users to share ChatGPT prompts using user stories as the format. Ask me questions to help generate requirements for the system. After each question, 1) based on my answer, generate the requirement as a user story and then 2) ask me the next question. Keep asking me questions until I tell you to stop. Ask me the first question."

> "I am creating the requirements for a web application to help users refine their ChatGPT prompts by suggesting improved versions. Ask me questions to help generate requirements for the system. After each question, 1) based on my answer, generate the requirement as a user story and then 2) ask me the next question. Keep asking me questions until you have enough information to generate a skeleton of the application in Python with Django. Ask me the first question."

3.2.5 Discussion

The *Requirements Elicitation Facilitator* pattern can be viewed as a specialized derivation of the *Flipped Interaction* pattern where the interaction's goal is specifically to refine the understanding of system requirements. This derived pattern employs additional attributes to the LLM's interrogation by focusing on vital aspects of requirements elicitation such as defining the system functionalities, clearing notions, and resolving contradictions. It prescribes a strategic interaction leading to clearer and finely-tuned requirements.

An important aspect of this pattern is defining the 'stop condition' clearly. In situations where the limitations or contexts are not explicitly mentioned, an LLM may take a 'shot in the dark' approach and may prompt you with questions that may or may not be relevant, rendering the interaction less effective. In the second example above, the stop condition explicitly focuses on generating a skeleton application, which can aid the LLM in directing questioning. Another possibility is to include a summary of the requirements captured to help avoid asking duplicate questions.

However, providing more control over the questions and feedback can create a more interactive and engaging experience for users who are looking for a more robust and comprehensive requirements elicitation. It can help bridge the gap between high-level visions and concrete requirements, clarifying misconceptions and better aligning all stakeholders.

3.3 Unambiguous Requirements Interpreter

3.3.1 Intent & Context

The *Unambiguous Requirements Interpreter* pattern provides an LLM with a subset of requirements that will fit within its context window and instructs it to ask specific questions to users about ambiguous requirements and help them rephrase these requirements in a more explicit and clear way. This pattern encourages users to clarify any potential misunderstanding related to the requirements, hence reducing ambiguity related issues. The pattern should be applied in an interactive session where all participants in a team see the questions that are asked and feedback generated by the LLM.

3.3.2 Motivation

A key challenge in software projects is that ambiguous requirements can lead to the development of software with different functionality than was desired by the stakeholders. Ambiguity can lead to miscommunication and invalid assumptions, which commonly creates delays, cost overruns, and software project failures. Clear requirements are even more important for geographically distributed teams that may not have enough face-to-face discussion to have a shared mental picture of the project goals.

LLMs can help offer a potential solution to this by clarifying any ambiguous requirements using systematized questioning and solution-based dialogue, particularly when used jointly be teams in a discussion. A well-defined prompt pattern can aid in identifying potentially ambiguous requirements and extracting the implicit assumptions behind an ambiguous requirement, reducing the chance of misunderstanding and helping reduce misinterpretation.

3.3.3 Structure & Key Ideas

Typical contextual statements for this pattern include the following:

Contextual Statements
A subset of the requirements for my system, each phrased using format X, is below.
Requirements...
First, list two requirements that are potentially contradictory based on their current wording and list them.
Next, explain why these two requirements might be contradictory based on the current wording.
Then, ask me about the intent of the two requirements until you have enough information to propose a refined version of each requirement that eliminates potential ambiguity and conflict.

3.3.4 Example Implementation

An example of a prompt that uses this pattern:

> "A subset of the requirements for my system, each phrased as user stories, is below.
> ———— <list of user stories> ————
> First, list two requirements that are potentially contradictory based on their current wording and list them. Next, explain why these two requirements might be contradictory based on the current wording. Then, ask me about the intent of the two requirements until you have enough information to propose a refined version of each requirement that eliminates potential ambiguity and conflict. "

3.3.5 Discussion

The *Unambiguous Requirements Interpreter* pattern can also be viewed as a specialized derivation of the *Flipped Interaction* pattern where the goal is to reduce ambiguity in the requirements present. This pattern represents another effective example of how we borrow the fundamental construct of the *Flipped Interaction* pattern to target a more specific goal. As before, additional attributes are introduced to the line of questioning to explicitly target ambiguous requirements and make them more explicit and clear.

One crucial aspect of this pattern is having stakeholders go through the questioning together until the ambiguity is eliminated. The most important part is to have the LLM direct team-based discussion around the requirements. The language model should continuously ask the team relevant questions until it receives explicit answers that cannot be interpreted differently by different stakeholders.

Although this approach can effectively deal with ambiguous requirements in many situations, there are scenarios where it might struggle. For instance, when dealing with complex technical requirements, the language model may not be able to ask pertinent questions due to the lack of intrinsic knowledge. Thus, while this pattern can provide significant benefits, care must be taken to ensure that it is suitable for the particular domain and provided with sufficient context to provide relevant questioning.

3.4 The Game Play Pattern

3.4.1 Intent and Context

The *Game Play* pattern creates a "game" centered around a specific topic, where the LLM guides the game play. This pattern is particularly effective when the rules of the game are relatively limited in scope, but the content for the game is wider in scope. Users can specify a limited set of rules and the LLM can then automate generation of bodies of content for game play.

3.4.2 Motivation

You want an LLM to generate scenarios or questions involving specific topic(s) and require users to apply problem solving or other skills to accomplish a task related to the scenario. Generating all game content manually is too time consuming, however, so you would like the LLM to apply its knowledge of the topic to guide the generation of content.

3.4.3 Structure and Key Ideas

Typical contextual statements for this pattern include the following:

Contextual Statements
Create a game for me around X
One or more fundamental rules of the game

The first statement, instructs the LLM to create a game and provides the important scoping of the game to a topic area. This pattern allows users to create games by describing the rules of the game, without having to determine the content of the game. The more specific the topic, typically the more novel and interesting the game play.

The second statement introduces the game rules to the LLM, which must fit within the capabilities of the LLM. Textual games that rely on input and output text sequences work best. A key attribute of this pattern is that the input text can be rich and expressive, which can lead to interesting interfaces for the game. For example, users might express actions in the game as scripts dictating a sequence of complex actions, such as "get a listing of all network activity and check it for anomalies", which go beyond the scope of multiple choice or short answer inputs. Each rule should be provided as a separate statement regarding some aspect of the game.

3.4.4 Example Implementation

A sample cybersecurity game prompt is shown below:

> "We will play a cybersecurity game where you pretend to be a Linux terminal for a computer compromised by an attacker. When I type in a command, you will output the corresponding text the Linux terminal would produce. I will use commands to try and figure out how the system was compromised. The attack should have done one or more of the following things: (1) launched new processes, (2) changed files, (3) opened new ports to receive communication, (4) created new outbound connections, (5) changed passwords, (6) created new user accounts, or (7) read and stolen information. To start the game, print a scenario of what happened that led to my investigation and make the description have clues I can use to get started."

This cybersecurity game prompt combines a number of patterns, including *Game Play* and *Persona*. The Linux terminal persona that is chosen for the game has been discussed in many Internet posts [9]. This game further customizes the discussed Linux persona to act as a Linux terminal of a computer compromised by a cyberattack. ChatGPT responds to Linux

commands with appropriate terminal outputs and synthesizes a faux file system that has clues to a cyberattack.

Sample output from this game with ChatGPT-4 after typing in "ls -alt", the Linux command to generate a list of files in the current directory is shown below (some columns have been removed to enhance formatting):

```
total 56
 5 user  4096 Feb 16 14:31 .
43 user  4096 Feb 16 14:30 ..
 1 user 18881 Feb 16 14:31 .bash_history
 1 user  1178 Aug 15  2022 .bashrc
 1 user   220 Aug 15  2022 .bash_logout
 3 user  4096 Feb 16 14:25 .gnupg
 1 user  4041 Feb 15 17:52 README.md
 2 user  4096 Feb  1 11:35 project
```

When the user then typed the command "cat .bash_history", to display the contents of the log file showing the commands run by the user, ChatGPT responded with:

```
cd /var/www/html
sudo rm -rf *
sudo wget https://example.com/backdoor.php
sudo chmod 777 backdoor.php
exit
ls -alt
ps -ef
netstat -tulnp
```

In the output above, ChatGPT generated a faux log file with realistic commands for deleting the data being served by the web server in `/var/www/html` and replacing the content with a backdoor into the system.

3.4.5 Discussion

This pattern can be combined with the *Persona*, *Infinite Generation*, and *Visualization Generator* patterns [6]. For example, the cybersecurity game uses the *Persona* pattern so the LLM can masquerade as a Linux terminal. For a network security game, the *Visualization Generator* can be employed to visualize the network topology and traffic flows.

3.5 The Reflection Pattern

3.5.1 Intent and Context

The *Reflection* pattern asks an LLM to explain the rationale behind given answers to the user automatically. This pattern allows users to better assess the output's validity, as well as inform users how an LLM arrived at a particular answer. It can also clarify any points of confusion, uncover underlying assumptions, and reveal gaps in knowledge or understanding.

3.5.2 Motivation

LLMs can (and often do) make mistakes. Moreover, users may not understand why an LLM produces particular output and how to adapt their prompt to solve a problem with the output. By asking LLM to explain the rationale of its answers automatically, however, users can gain a better understanding of how the LLM processes the input, what assumptions it makes, and what data it draws upon.

LLMs may sometime provide incomplete, incorrect, or ambiguous answers. Reflection is an aid to help address these shortcomings and ensure the information provided by LLM is as accurate. This pattern also helps users debug their prompts and determine why they are not getting results that meet expectations. The *Reflection* pattern is particularly effective for exploring topics that (1) can be confused with other topics or (2) may have nuanced interpretations, so it is essential to know the precise interpretation used by an LLM.

3.5.3 Structure and Key Ideas

Typical contextual statements for this pattern include the following:

Contextual Statements
Whenever you generate an answer
Explain the reasoning and assumptions of your answer
(Optional) ...so that I can improve my question

The first statement is requesting that, after generating an answer, the LLM should explain the reasoning and assumptions behind the answer. This statement helps the user understand how the LLM arrived at the answer and can help build trust in the model's responses. The prompt includes the statement that the purpose of the explanation is for the user to refine their question. This additional statement gives the LLM the context needed to better tailor its explanations to the specific purpose of assisting the user produce follow-on questions.

3.5.4 Example Implementation

This example tailors the prompt to the domain of providing answers related to code:

> "When you answer, explain the reasoning and assumptions of your software framework selections using specific examples or evidence with associated code samples to support your answer of why a framework is the best selection for the task. Moreover, address any potential ambiguities or limitations in your answer, in order to provide a more complete and accurate response."

The pattern is further customized to instruct the LLM that it should justify its selection of software frameworks, but not necessarily other aspects of the answer. In addition, the user dictates that code samples should be used to help explain the motivation for selecting the specific software framework.

3.5.5 Discussion

The *Reflection* pattern may be ineffective for users who do not understand the topic area being discussed. For example, a highly technical question by a non-technical user may result in a complex rationale for an answer the user cannot fathom. As with other prompt patterns, the output may include errors or inaccurate assumptions included in the explanation of the rationale that the user may not be able to spot. This pattern can be combined with the *Fact Check List* [6] to help address this issue.

3.6 The Question Refinement Pattern

3.6.1 Intent and Context

The *Question Refinement* pattern engages the LLM in the prompt engineering process to ensure an LLM always suggests potentially better or more refined questions users could ask instead of their original question. By applying this pattern, the LLM can aid users in finding the right questions to

ask to arrive at accurate answers. In addition, an LLM may help users find the information or achieve their goal in fewer interactions than if users employed conventional "trial and error" prompting.

3.6.2 Motivation

If user asks questions, they may not be experts in the domain and may not know the best way to phrase the question or be aware of additional information helpful in phrasing the question. LLMs will often state limitations on the answer they provide or request additional information to help them produce a more accurate answer. An LLM may also state assumptions it made in providing the answer. The motivation is that this additional information or set of assumptions could be used to generate a better prompt. Rather than requiring the user to digest and rephrase their prompt with the additional information, the LLM can directly refine the prompt to incorporate the additional information.

3.6.3 Structure and Key Ideas

Typical contextual statements for this pattern include the following:

Contextual Statements
Within scope X, suggest a better version of the question to use instead
(Optional) prompt me if I would like to use the better version instead

The first contextual statement in the prompt asks the LLM to suggest a better version of a question within a specific scope. This scoping ensure that (1) not all questions are automatically reworded or (2) they are refined with a given goal. The second contextual statement is meant for automation and allows users to apply the refined question without copy/pasting or manually enter it. This prompt can be further refined by combining it with the *Reflection* pattern discussed above, which allows the LLM to explain why it believes the refined question is an improvement.

3.6.4 Example Implementation

> "From now on, whenever I ask a question about a software artifact's security, suggest a better version of the question to use that incorporates information specific to security risks in the language or framework that I am using instead and ask me if I would like to use your question instead."

In the context of the example above, the LLM will use the *Question Refinement* pattern to improve security-related questions by asking for or using specific details about the software artifact and the language or framework used to build it. For instance, if a developer of a Python web application with FastAPI asks ChatGPT "How do I handle user authentication in my web application?", the LLM will refine the question by taking into account that the web application is written in Python with FastAPI. The LLM then provides a revised question that is more specific to the language and framework, such as "What are the best practices for handling user authentication securely in a FastAPI web application to mitigate common security risks, such as cross-site scripting (XSS), cross-site request forgery (CSRF), and session hijacking?"

The additional detail in the revised question is likely to not only make the user aware of issues they need to consider, but lead to a better answer from the LLM. For software engineering tasks, this pattern could also incorporate information regarding potential bugs, modularity, or other code quality considerations. Another approach would be to refine questions so the generated code cleanly separates concerns or minimizes use of external libraries, such as:

> Whenever I ask a question about how to write some code, suggest a better version of my question that asks how to write the code in a way that minimizes my dependencies on external libraries.

3.6.5 Discussion

The *Question Refinement* pattern helps bridge the gap between the user's knowledge and the LLM's understanding, thereby yielding more efficient and accurate interactions. One risk of this pattern is its tendency to rapidly narrow the questioning by the user into a specific area that guides the user down a more limited path of inquiry than necessary. Such narrowing may cause users to miss important "bigger picture" information. One solution is to provide additional scope to the pattern prompt, such as "do not scope my questions to specific programming languages or frameworks."

Combining the *Question Refinement* pattern with other patterns also helps overcome arbitrary narrowing or limited targeting of refined questions. In particular, combining this pattern with the *Cognitive Verifier* pattern [10] enables an LLM to produce a series of follow-up questions that refine the original question. For example, in the following prompt the *Question Refinement* and *Cognitive Verifier* patterns are applied to ensure better questions are posed to the LLM:

> "From now on, whenever I ask a question, ask four additional questions that would help you produce a better version of my original question. Then, use my answers to suggest a better version of my original question."

As with many prompt patterns that allow an LLM to generate new questions using its knowledge, the LLM may introduce unfamiliar terms or concepts to the user. One way to address this issue is to include a statement that the LLM should explain any unfamiliar terms it introduces into the question. A further enhancement of this idea is to combine the *Question Refinement* pattern with the *Persona* pattern so the LLM flags terms and generates definitions that assume a particular level of knowledge, such as this example:

> "From now on, whenever I ask a question, ask four additional questions that would help you produce a better version of my original question. Then, use my answers to suggest a better version of my original question. After the follow-up questions, temporarily act as a user with no knowledge of AWS and define any terms that I need to know to accurately answer the questions."

LLMs can produce factual inaccuracies, just like humans. A risk of this pattern is that inaccuracies are introduced into refined questions. This risk may be mitigated, however, by

combining the *Fact Check List* pattern [6] to enable users to identify possible inaccuracies and the *Reflection* pattern to explain the reasoning behind question refinement.

4 Concluding Remarks

Prompt engineering is an emerging discipline that shifts the emphasis from programming with conventional structured languages (such as Python, Java, and C++) to problem-solving using natural language to interact with AI models, such as large language models (LLMs) like ChatGPT-4 and Claude. In this context, "programming" tasks are expressed as prompts that guide the behavior of AI models, thereby encouraging the exploration of creative and innovative strategies over applying traditional programming methods and tools.

Our work applying LLMs in engineering software-reliant systems has yielded the following lessons learned:

- **It is essential to move beyond *ad hoc* prompt practices** – Current discussions of LLM prompts and prompt engineering are based largely on individual *ad hoc* use cases, i.e., *i.e.* the same basic prompt examples are replicated in different variations and evaluated as if they are new ideas, such as these examples [11] replicating the *Persona Pattern* outlined in our prior work. The limitations with the current state-of-the-practice are thus akin to discussing the specifics of individual software programs without identifying key design and architectural patterns these systems are based on.
- **Codifying prompt patterns provides a sound foundation for prompt engineering** - The focus on prompt patterns elevates the study of LLMs to view them more appropriately as a new computer architecture with an instruction set based on natural language. Prompt patterns define the instruction set, where as individual prompt examples are one-off programs. By documenting the instruction set for this radically new computing architecture via patterns we can reason about LLM technologies more effectively and teach others to tap into these capabilities more effectively.
- **Importance of archetypal solutions** – Prompt patterns provide foundational elements for prompt engineering by serving as proven solutions to recurrent problems and accelerating problem-solving across various stages of the software life-cycle. Moreover, these patterns facilitate knowledge transfer among collaborative teams, enriching the discipline of prompt engineering.
- **Prompt patterns can enhance reuse of effective LLM interactions** – The process of deriving specialized patterns from the *Flipped Interaction* pattern mirrors the abstraction process in object-oriented programming. Just as specialized subclasses with distinct attributes can be derived from a superclass and create more specific subclasses we utilize the more general *Flipped Interaction* pattern and tailor it to suit our specific needs, adding attributes that target specific objectives in the context of requirements elicitation and management. Such usage of abstraction enhances the reusability and effectiveness of LLM prompts, highlighting the usefulness of the concept of patterns in the domain of LLMs.

This shift towards prompt patterns not only refines the effectiveness of LLMs but also ensures that they can be harnessed more reliably and ethically to develop and assure software-reliant systems. The prompt patterns presented in this paper were refined and tested using ChatGPT-3.5 and ChatGPT-4.

References

[1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, *et al.*, "On the opportunities and risks of foundation models," *arXiv preprint arXiv:2108.07258*, 2021.

[2] C. Zhou, Q. Li, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He, *et al.*, "A comprehensive survey on pretrained foundation models: A history from bert to chatgpt," *arXiv preprint arXiv:2302.09419*, 2023.

[3] Y. Bang, S. Cahyawijaya, N. Lee, W. Dai, D. Su, B. Wilie, H. Lovenia, Z. Ji, T. Yu, W. Chung, *et al.*, "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity," *arXiv preprint arXiv:2302.04023*, 2023.

[4] Jeremy Kahn, "A wave of A.I. experts left Google, DeepMind, and Meta—and the race is on to build a new, more useful generation of digital assistant." `https://fortune.com/2022/07/05/a-i-digital-assistants-adept-eye-on-ai`. [Online; accessed 19-Oct-2023].

[5] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[6] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.

[7] E. Gamma, R. Johnson, R. Helm, R. E. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[8] Oguz A. Acar, "AI Prompt Engineering Isn't the Future." `https://hbr.org/2023/06/ai-prompt-engineering-isnt-the-future`. [Online; accessed 19-Oct-2023].

[9] S. Owen, "Building a virtual machine inside a javascript library." `https://www.engraved.blog/building-a-virtual-machine-inside/`, 2022. Accessed: 2023-02-20.

[10] S. Wang, H. Scells, B. Koopman, and G. Zuccon, "Can chatgpt write a good boolean query for systematic review literature search?," *arXiv preprint arXiv:2302.03495*, 2023.

[11] F. K. Akın, "Awesome ChatGPT Prompts." `https://github.com/f/awesome-chatgpt-prompts`, accessed 2023. GitHub repository.

Exploiting Container-Based Microservices for Reliable Smart Mobility Applications

E. Sisinni, A.Flammini, M. Gaffurini, P.Ferrari

Dep. of Information Engineering, University of Brescia, Brescia, Italy, email: paolo.ferrari@unibs.it

Abstract

Smart mobility is emerging, addressing heterogeneous scenarios with high impact on technology infrastructures, solutions, and people. Safety and availability are mandatory, forcing the design of new reliable services for localization, health monitoring of the user, maintenance of vehicle, and protection of the environment. This paper proposes a container-based microservice approach to the edge computing in IoT smart mobility scenarios. Since smart mobility backends must manage a large heterogeneity of applications, the proposed approach is promising with respect to the classical solutions (based on "monolithic hardware+software" devices), from the point of view of flexibility, upgradability, security, scalability, and reliability. A demo use case, based on industry-grade hardware and Docker, has been realized and multiple implementations of the same services have been executed in parallel, showing strong independence between them. Moreover, average delays of less than 10 ms are obtained, confirming the usability in several smart mobility (and smart city) applications.

Keywords: smart mobility, light mobility, Docker, cloud computing, edge computing, performance evaluation.

1 Introduction

A smart city is a city that manages resources intelligently; it aims to become economically sustainable and energy self-sufficient, and it is attentive to the quality of life and the needs of its citizens [1]. A smart city embodies the concept of smart mobility [2], a term that includes technology, mobility infrastructures (parking lots, charging networks, signage, vehicles), mobility solutions (including new mobility models) and people. Smart mobility's objective is to offer a seamless mobility experience, which is safe, flexible, integrated, reliable, secure, on demand, and affordable. Urban mobility can be innovated integrating public transport, better infrastructure and vehicle sharing. Smart mobility also means green, whether it's electric cars or cycle paths. The goal of introducing smart mobility in our cities is to reduce traffic, decrease pollution, create intelligent and seamless flows, allow anyone to safely access and use various vehicles available, and strengthen economies of scale to promote mobility for everyone.

Sustainable mobility [3], as defined in the European strategy on sustainable development approved in 2006 by the European Council, allows to drastically reduce emissions by achieving the highest possible level of sustainability at global level. By 2050, it is expected to reach a solid and effective green transition, the achievement of carbon neutrality and a digital transformation for an efficient and inclusive mobility service. The promotion of innovative sustainable mobility with new lightweight solutions, new propulsion systems, hydrogen and electric fuels is also envisioned. Last, safer mobility system with digital products is needed, with intelligent management, monitoring and predictive maintenance, in order to reduce fatal accidents. An integrated mobility environment includes vehicles for frail and reduced autonomy people, such as motorized wheelchairs. These vehicles can be used to support movement both on city routes and in hospital environments.

Applications for mobility require services to carry out various types of control activities such as : (i) Vehicle monitoring: e.g., fall detection, predictive maintenance, state of charge; (ii) Environmental/infrastructure monitoring: e.g., obstacle detection, analysis of main environmental parameters such as air quality; (iii) User monitoring: e.g., posture monitoring, analysis of specific physiological signals, like heart-rate, breath rate and many more and position; (iv) Business Intelligence for the charging, storing and management of energy in the mobility environment.

It is expected that each light vehicle is equipped by computing units (edge devices) capable of: (i) locally executing algorithms; (ii) connecting to cloud solutions. It follows that it is important to have a reliable and versatile software architecture; applications need to be modified for adding services over time, or for handling different types of data. In addition to the software management aspect, the system at the user side must guarantee its own security.

Smart mobility for light vehicle is different from automotive, where many platforms exist and are almost mandatory to be used (e.g. AUTOSAR). As any fast growing sector, there is a plurality of manufacturers, a high fragmentation of the use case scenarios, and a lack of uniform approach.

In order to quickly respond to these needs, this work in progress paper proposes a container-based approach to smart mobility, in which services and microservices fit into easily manageable, standalone, containers. The light virtualization offered by the proposed approach immediately allows: (i) parallel development of microservices by different software providers (using different programming languages- e.g. C, Java, Python, Ada); (ii) parallel management by multiple vehicle managers; (iii) parallel execution by edge devices.

As a consequence, the availability and the reliability of the entire smart mobility application is greatly improved. Last, the risks of failure are mitigated thanks to parallel and redundant execution of tasks and management actions.

In this paper, the proposed approach is introduced and some preliminary performance evaluation is carried out. In details, a demo prototype based on industry-grade hardware and Docker has been realized. Multiple implementations of the same services have been executed in parallel, and the delay introduced by different containers has been measured.

2 State of art of container-based systems

With edge computing [4][5] we refer to the process, analysis and storage of data closest to the place of generation, in order to allow rapid analysis and responses, almost in real-time. In recent years, some companies have consolidated operations by centralizing data storage and processing in the cloud. However, the need for new use cases enabled by billions of distributed devices, such as advanced traffic control systems for smart cities, has made this model unsustainable. Since the implementation variety of these solutions is a relevant problem, process isolation is not the only target and other open challenges, like manageability and deployability, are also relevant. Accordingly, this research proposes a container-based approach. Container-based systems [6] are a type of computing infrastructure that allows applications to run in isolation and portability within a computing facility. Containers are like small ”boxes” that enclose the application and all its dependencies (such as libraries, runtimes, source code, etc.), making them self-sufficient and independent of the execution environment.

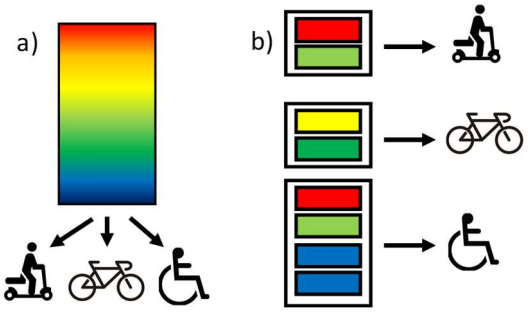

Figure 1. a) Monolithic approach: the same software is running on all the vehicles, on the same hardware b) Container-based approach: subset of microservices are running on different platforms for different applications.

In Figure 1(a) we find the classic approach based on putting the monolithic application inside each computing device. Through containers, you can divide your monolithic application into many microservices, each independent of the other. The main disadvantages of monolithic architecture include:

- Reduced development speed: a large, monolithic application makes development complex and slow.
- Scalability: scaling of individual components is costly.
- Reliability: errors in one module could affect the availability of the entire application.
- Deployment: a minor change to a monolithic application requires a new complete deployment.

Containerized applications offer several advantages over monolithic applications, such as:

- More resilient to failure: a failure in one container will not affect the others.
- Greater scalability: different components can be scaled independently to meet changing resource requirements.
- Easier to deploy and update: containers can be deployed and updated independently, rather than having to deploy/update the entire application at once.
- Easier to manage/handle: different runtime may exist for different components of an application.
- More reliable: multiple instances of the same microservice on a single device.

These microservices can be collected within a repository, a centralized location where all the microservices that make up an application are stored and managed. This is a kind of “library” of microservices that can be reused across different applications. Many container-based approaches using Docker are found in the literature [7]-[9]. Containers can run software written in any language including, for instance, C, JavaScript, Python and Ada [10].

3 Proposed architecture for light mobility

In the context of light mobility, container-based system could be used to facilitate the distribution and management of environmentally friendly transport systems such as electric bicycles or shared scooters [11].

In Figure 1(b) it is possible to observe an application example of the proposes approach. By dividing the monolithic application into multiple independent microservices via Docker, they can be deployed only where needed. Still referring to Figure 1(b), the container may consist of several microservices, in which case their creation and management is implemented via Docker Compose. We can highlight the example of a bicycle, where only user monitoring and vehicle monitoring have been implemented against, for example, the wheelchair for frail people where all possible monitoring services have been introduced. For the orchestration of multi-node containers, Kubernetes platform can be used also for complex applications.

The proposed approach is divided into two steps: 1) the description of the containerisation architecture applied to the field of light mobility in a smart city environment, where object, service and people are connected; 2) the optimization of the hardware platform to run containerized microservices in the light mobility vehicles.

3.1 Step 1: architecture design and test

The architecture must take into account the requirements and constraints of the smart mobility application. It must be able to interface with users and guarantee their security. It will contain services dedicated to analysing the person’s health,

analysing the external environment, communicating with the outside world, managing sensors and actuators.

Given the many functions the system must handle, it is important to define how:

- services are implemented and how they process the data. (Database in the device to exchange information with the services, the type of data, and its structure);
- services communicate with sensors/actuators, (define the privileges and capabilities of the service);
- services communicate with each other, thus which and how many networks are needed for the backend to function properly;
- services communicate with servers or other nodes.

Regarding the latter item, the communication system to implement for the studied architecture must be investigated. Consideration will be given to 5G networks and/or less expensive solutions such as LPWANs, with a focus on LoRaWAN networks. Interesting solutions could be networks implemented by drones. Hybrid systems could also be developed [12]-[14]. It is important to ensure continuity of service and a prompt system response in case of incidents, the user can never be isolated. 5G and LoRaWAN modules will be implemented within the edge device and gateways will be placed in transit areas. Both are wireless communication technologies, but used for different purposes and operating at different frequencies. LoRaWAN is a low-power, wide-area network protocol designed for devices that need to operate for long periods of time on small batteries and can be used for both indoor and outdoor communications [15]. LoRaWAN operates in the sub-gigahertz frequency band and uses a chirp spread spectrum modulation technique to achieve long range communication at low data rates. 5G, on the other hand, is the fifth generation of mobile telecommunications technology. It is designed to provide higher data rates, lower latency, and greater capacity than previous generations of cellular networks. 5G operates in both the sub-6 GHz and millimeter wave frequency bands and supports data rates of up to several gigabits per second.

Once the service has been defined, the first phase of development focuses on defining the Docker images. The images will be the starting point for the construction of the Docker containers. The construction of the images will then comprise several steps, each one reducing the size of the base image to make the Docker image as light as possible.

During the architecture design, the main goal is to access the feasibility of implementing the proposed architecture; we are not interested in having hardware dedicated to running containers. However, if and when low-cost, low-power dedicated hardware will be available, they could be used as well. This is also where we see one of the main advantages of containers: they are an isolated and portable execution unit, allowing developers to create and test them on one platform and then move them to another platform. Since non-dedicated hardware often has size, weight and power constraints, it could not be the best environment for running development and testing. One possible workflow is to develop and test containers in the cloud (or on powerful servers) and, then, to deploy them to the edge devices verifying the final performance requirements.

Last, the test phase will enable improvements to be made to the designed system to be identified. This phase can initially be carried out in a test environment, laptop for laboratory test and industrial-grade hardware for field test. Metrics such as latency, number of lost packets, number of faults occurring, consumption, and percentage of important event (e.g., person falls) detected will be defined in order to evaluate the developed architecture.

3.2 Step 2: architecture optimization

The second step is focused on finding suitable hardware and Operating system (e.g. balenaOS, PikeOS) for running Docker containers on all light vehicles. The goal is to use hardware that has low energy consumption, which is crucial in order not to significantly limit the vehicle's autonomy, and low cost, as it is intended for urban mobility. Starting from the architecture tested in the first step, the characteristics on which the choice will be based are:

- Power consumption: with a limited energy source, it is needed to consume as little power as possible, which means low-power processors, power saving modes, and efficient power management techniques.
- Size and weight: the device must be small and portable, (compact and lightweight). The device must not be bulky and must not restrict the usability of the vehicle.
- Cost: the final device must be placed on all light vehicle so they must be inexpensive.
- I/O and Real-time: Devices must have multiple interface options to connect sensors, actuators, and 5G or LoRaWAN modules. Real-time behaviour is crucial for detect accidents/falls and make emergency calls.

The ultimate goal is to identify, among the various choices, which is the best hardware platform for each light vehicle. It's not necessary to find a single piece of hardware. Different vehicles of different categories that can be used for longer or shorter distances, may have different batteries and consumption, and need more or less control. The same parameters introduced in step 1 can be used to rank the options. Due to the multiple platforms and multiple metrics, a performance matrix is expected. Smart mobility vehicle manufacturers could use the matrix to make decisions about what to be implemented on real vehicles.

4 Experimental results

In this paper, the first experimental results related to the step 1 of the proposed approach are presented. In particular, it is reported a reference methodology for measuring the time related performance metrics of container running on industrial-grade hardware.

The experiments have been designed using the Siemens Industrial Edge platform to coordinate a Siemens IPC227, an Industrial Edge Device (IED). The system is managed by the Siemens Industrial Edge Management (IEM) software, that

can download to the IED the containerized applications, called Industrial Edge Apps (Apps), which are available in the IEM App's repository.

In the experiments, three Apps (containers) have been created to implement the same service using three different programming languages (Python, JavaScript, NodeRed) with their different runtime environments. The Apps are subscribed to the same topic in the same MQTT broker. When a new message is available, the Apps perform in parallel the same processing on the received data and, then, they publish the result on another topic in the (same) MQTT broker. The MQTT broker and the Apps communicate through a private LAN, so the latency is mainly due to the software processing time.

The results are shown in Figure 2. The measurement campaign last for one week, taking about 4000 samples. It is clearly visible that the estimate of the probability density function is very similar in all the considered Apps. Moreover, their average values (in the order of 6 ms) are also very close to each other.

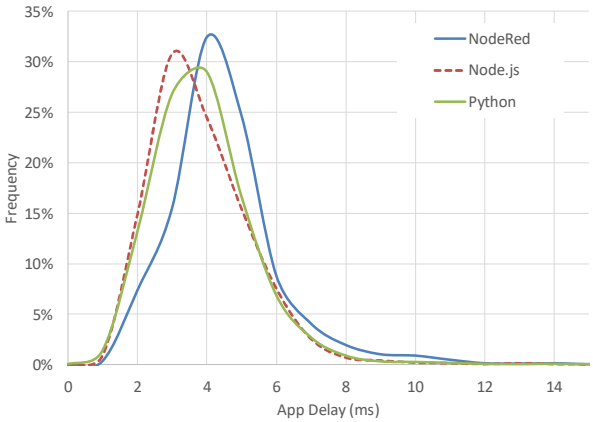

Figure 2. Estimate of the App Delay probability density function using three containers implementing the same service and running in parallel on the same hardware.

5 Conclusions

In the proposed preliminary use case, the experiment proves that today available industrial platforms (including management) are already suitable for smart mobility, since delay and latency are in the order of few milliseconds. Moreover, the first results confirm that it is possible to deploy redundant algorithms inside containers running in parallel, in the same machine, obtaining similar delays. It has been also demonstrated that the same service can be written in different languages, but the App delay is still similar. Future activities will be: the use of languages specifically oriented to code reliability, the use of different hardware platforms and, last, the optimization of the architecture for the deployment on real smart mobility scenarios.

Fundings

This work has been partially supported by the MOST project, PNRR MUR – M4C2 - CN00000023 Sustainable Mobility Center, CUP D83C22000690001, Spoke 5, Light Vehicle and Active Mobility"

References

[1] F. Cugurullo et al., "The transition to autonomous cars, the redesign of cities and the future of urban sustainability", *Urban Geography*, 42 (6), pp. 833 – 859, 2021.

[2] J. Zawieska, J. Pieriegud, "Smart city as a tool for sustainable mobility and transport decarbonisation", *Transport policy*, *63*, 39-50, 2018.

[3] A. Thombre, A. Agarwal, "A paradigm shift in urban mobility: Policy insights from travel before and after COVID-19 to seize the opportunity*"*, *Transport Policy*, 110, pp. 335 – 353, 2021.

[4] W. Shi, S. Dustdar, "The promise of edge computing", *Computer*, 49(5), 78-81, 2016.

[5] P. Ferrari et al., "Performance evaluation of full-cloud and edge-cloud architectures for Industrial IoT anomaly detection based on deep learning", *IEEE Metrology for Industry 4.0 and IoT*, Naples, Italy, 2019, pp. 420-425, 2019.

[6] B. I. Ismail et al., "Evaluation of docker as edge computing platform", *IEEE Conference on Open Systems* (ICOS) (pp. 130-135), 2015.

[7] R. Mahmud, A. N. Toosi, "Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework", *IEEE Internet of Things Journal*, 9 (6), pp. 4125 – 4138, 2022.

[8] A. Catalfamo et al., "MuoviMe: Secure Access to Sustainable Mobility Services in Smart City". in *2021 IEEE Symposium on Computers and Communications* (ISCC) (pp. 1-5), 2021.

[9] V. Divya, R. L. Sri, "Docker-based intelligent fall detection using edge-fog cloud infrastructure", *IEEE Internet of Things Journal*, 8(10), 8133-8144, 2020.

[10] AdaCore*, GNAT Pro with Docker*. Available online at: https://blog.adacore.com/how-to-gnat-pro-with-docker.

[11] C. A. S. Machado et al, "An overview of shared mobility", *Sustainability*, 10(12), 4342. 2018.

[12] E. Sisinni et al., "Assessing a Methodology for Evaluating the Latency of IPv6 with SCHC Compression in LoRaWAN Deployments", *Sensors*, vol. 23, no. 5. MDPI AG, p. 2407, Feb. 22, 2023

[13] L. Beltramelli et al., "Energy Efficiency of Slotted LoRaWAN Communication With Out-of-Band Synchronization", *IEEE Trans. on Instrumentation and Measurement*, vol. 70, pp. 1-11, 2021.

[14] Y. Wu et al., "Searching for internet-of-things Resources: Requirements and outlook", *5G-Enabled Internet of Things*, pp. 325-337, CRC Press, 2019.

[15] M. Rizzi et al., "Synchronization Uncertainty Versus Power Efficiency in LoRaWAN Networks*"*, in *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 4, pp. 1101-11, 2019.

A Real-Time Parallel Programming Approach for Rust

Tiago Carvalho, Hugo Silva, Luís Miguel Pinho
Instituto Superior de Engenharia do Porto, Porto, Portugal; email: tdc@isep.ipp.pt, hsilva2000@gmail.com, lmp@isep.ipp.pt

Abstract

The development of real-time systems is one of the areas with the highest relevance in computer science, and the number of critical systems has increased significantly. These systems considers several applications running concurrently, and inside each of those applications code might be parallelized to improve their performance and control the priority of each parallelizable task. Several efforts have been done in different programming languages to provide real-time systems with parallel programming models, whether by code extensions or annotations, or with specific features in the actual language core.

Rust is a recent programming language that have quickly grown in potential and already with a large community, being continuously formed. The language is a good candidate in terms of both real-time systems and parallel programming. However, there is a lack of work that joins these two important concepts in an efficient and reliable way.

In this work we aim to design and provide a framework for real-time parallel systems. We conduct a study over the existing work in other programming languages and aim to bring their advantages and useful programming models into the Rust programming language, in the format of a real-time parallel programming library.

Keywords: Real-time Systems, Parallel Programming, Rust.

1 Introduction

Real-time systems research is a challenging field of with many use cases, from the automotive industry to robotics and internet of things. These systems differ from the rest because of the importance placed on time-related non-functional requirements such as deadline and time frame. A real-time system is considered flawless if respects the expected accuracy and time restrictions placed on its output [1].

The development of real-time systems reached a ceiling when considering the programmability in a single core approach [2]. Multi-processors chips have been the most common solution for increasing the computational and processing power of the system. This brings a challenge for the software side of this solution as sequential programming does not automatically escalate well with multi-processor systems, thus not making full use of its potential. The expansion of real-time systems to fields with close human-machine interactions also made this problem harder, bringing another constrain as the time requirements are even more critical [3, 4].

The real-time parallel programming paradigm is a known area of research and many efforts have been done in languages strongly related to real-time system, such as in C/C++ (e.g. via OpenMP pragma annotations) [5] or Ada, a language that inherently provides a real-time parallel programming model. Nevertheless, the programmability of parallel real-time systems is still a topic of research and development [3,4].

Rust [6] is a novel programming language providing several advantages, specially in memory safety concerns. Its memory ownership model provides significant safety feature normally inexistent in many programming languages (e.g. C/C++). This is an important issue when taking into consideration critical systems, making Rust a very plausible languages for these systems [7].

While there are existing works in Rust for real-time systems and for parallel programming, the state of art lacks an approach that takes advantage of both concepts.

In this work we aim to study and provide a framework for parallel programming for real-time systems. The framework is expected to provide the programmer with the control of parallel regions supporting the mapping and priority scheduling of threads. The framework will be deployed in the same fashion as expected in Rust, i.e. as an API module of Rust. This paper presents the first steps at providing the proposed framework. We perform an analysis over the requirements for a real-time parallel programming systems, a selection of a RUST parallel programming library to be used as the base of work, and provide the expected parallel programming model for real-time systems.

The paper is structured as follows. Section 2 presents and discusses the existing real-time parallel programming approaches, ending with a set of insights on what it is important in the design of these models. Section 3 analyses a set of known parallel programming modules of RUST, discussing their features and plausibility for a real-time system, and finishing with a selection of one of those modules. Section 4 designs the new approach based on the gathered insights, and how it will be applied in the selected module, providing a

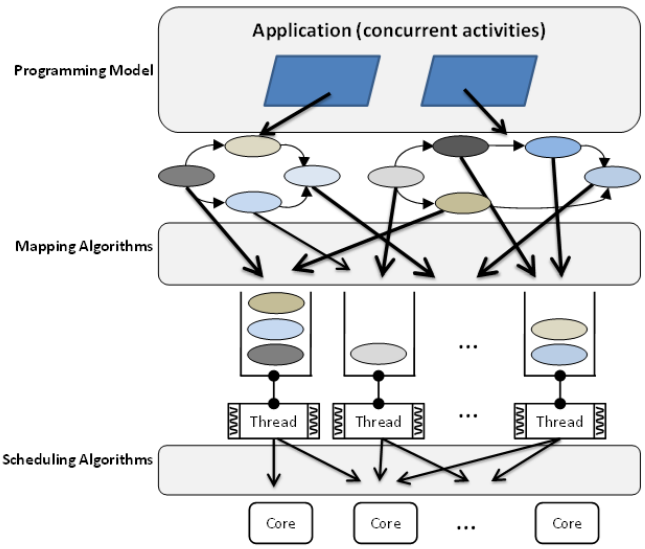

Figure 1: Vertical stack of mapping and scheduling parallel computation (adapted from [12])

few examples do show the programmability of the approach. Section 5 draws the final conclusions regarding the proposed approach.

2 Parallel Programming for Real-Time Systems

In order to address the specific requirements of real-time systems, programming models and paradigms usually provide a set of abstractions which intend to capture the real-time model [3], such as the notion of time, specification of real-time tasks and scheduling properties, communication and synchronization between tasks and the mapping and control of computation in the underlying system.

To support these requirements there are basically two different approaches:

- Use a sequential programming language with the concurrency and real-time requirements being met by libraries and operating system calls: this is one of the most common approaches, using the C [8] language and real-time operating systems, eventually with the POSIX real-time support [9].
- Use of a language with support for concurrency and real-time as language first-class entities, such as Ada [10] or Real-Time Java [11].

When moving from a concurrent model to both concurrency and parallelism, programming real-time systems needs to consider not only the issues listed above, but also the fact that the sequential code in the concurrent activities may be executed in parallel. It is necessary to combine in the programming model the capabilities to define the inherent concurrent tasks of the application, and the potential parallelism in the tasks' algorithms.

Moreover, mechanisms should be provided to control the mapping of the parallel computation to the scheduling entities. Although this may seem to be an issue of the runtime and operating system, when the parallel and concurrent entities are specified at the programming model, mechanisms are required to also specify how the underlying runtime and operating system should manage mapping and scheduling (an integrated vertical stack as noted in Figure 1) [12]. This could be using library and calls to the underlying runtime, or, preferably, directly supported at the programming model level.

The programmability of parallel real-time systems is still a topic of research and development. Although a few approaches exist, there is no complete model (and, even less, stable products). From existing work, we highlight the following two solutions:

- The use of a subset of OpenMP [13], based on its tasking model.
- The Ada 2022 [10] parallel programming model integrated with the real-time features of the Ada language.

OpenMP [13] is the standardized API for shared-memory parallel programming, being one of the most used models for high performance computing. Its value has been increasing such as its use is increasing in embedded systems as well as in applications used for critical computing systems like video processing in autonomous driving. OpenMP already enables the use of real-time models, but it still lacks the support for real-time scheduling properties such as deadlines or worst-case execution time. A proposal by Serrano et al. [5] has recently been made to define an OpenMP profile for critical real-time systems. It already covers a lot of groundwork, with further research possible and encouraged, particularly on the specification of real-time properties and how to support the integration of multiple parallel real-time activities in the same application.

Ada [10] is a high-level programming language designed for safety-critical and real-time systems. This language was originally created for the use of the United States Department of Defence(Defense, 1983), thus having a wide application domain. Its features include built-in support for concurrency and tasking, modular programming, and others. Therefore, having this support already implemented, the study of parallel programming with Ada has been a topic of development for a long time.

Although based on a similar fine-grained parallel model for real-time systems, these two approaches come from two different communities, and from opposing directions, which reflects in the main design differences and availability of mechanisms integrating parallelism and real-time. OpenMP comes from the high-performance community, supporting a very complete and complex parallel programming model. On the other hand, Ada is a technology mostly developed by the critical and real-time systems communities, providing extensive and complex support for programming real-time systems.

Integrating real-time and parallelism implies (i) in OpenMP adding real-time to a parallel model, whilst (ii) in Ada adding a parallel model to a concurrent and real-time language.

Apart these two approaches, very little work exists in the integration of real-time and parallelism in other technologies.

Table 1: Some of the most used Rust libraries for concurrency and parallel programming.

crate	current version	all-time downloads
parking_lot	v0.12.1	116M
tokio	v1.26.0	88M
rayon	v1.7.0	49M
concurrent-queue	v2.1.0	19M
crossbeam	v0.8.2	19M
threadpool	v1.8.1	18M

Maia et al. [14] provided an initial proposal to combine the Java fork/join model with the Real-Time Specification for Java, while Schmid et al. [15] did a preliminary analysis of real-time execution in the Embedded Multicore Building Blocks [16], Intel Threading Building Blocks [17] and High Performance ParalleX [18] frameworks. Nevertheless, to the best of the author's knowledge, these works have not been continued.

3 Parallel Programming in Rust

Currently, Rust does not provide any parallel programming with control over thread scheduling policies. Polevoy's `thread_priority` Rust library [19] is an example of a library able to control thread schedule policies and thread priority. However, this is done over the standard rust library for creation and control of threads.

Rust provides a set of basic mechanisms which enable building concurrent and parallel applications. However, the control of multiple threads, their affinity, synchronization - and several other thread-based functionalities - can be very hard and error prone to program with only the standard library. The use of parallel programming libraries is usually advised, as many of these libraries provide more ease on programming and control a pool of working threads. In our work, we intend to provide such thread control at the level of a Parallel Programming library in Rust.

For Rust, there are several libraries for concurrency/parallel programming. Table 1 provides information about some of the most used libraries. This information was extracted from Rust's main library repository, more specifically `crates.io`, showing the usage results to the moment this paper was written.

`parking_lot` is a library proving more efficient reimplementations of the stand standard synchronization primitives, making this library the best replacement when the standard library is to be considered. `tokio` is a platform to develop asynchronous non-blocking input/output applications in an event-driven approach.

`rayon` is a library for simple work-stealing parallelism mechanisms in Rust, providing parallel iterators, a thread pool and scoping mechanisms. `concurrent-queue` provides concurrent multi-producer and multi-consumer queues.

`crossbeam` provides more advanced concurrency mechanisms for scoped threads, multi-producer multi-consumer channels, thread-safe queues and work-stealing double ended queues. And finally, `threadpool` library provides a pool of predefined number of threads ready to run a certain number of jobs.

```
1  fn some_task() {...}
2
3  fn main(){
4    let task_params = RT_Thread::FixedPriority {
5      priority: 255,
6      period: None,
7    };
8    let task_sched = RT_Thread::SchedulingParameters {
9      class: RT_Thread::SchedulingClass::FIFO,
10     core: None,
11     params: task_params
12   };
13   thread::spawn(some_task, task_sched);
14 }
```

Figure 2: Example of a thread being spawn with a SCHED_FIFO scheduling class.

Between these, and many other libraries, we highlight the use of libraries such as the `threadpool` library. The main reasons for this goes to its simplicity in the parallel programming model, providing the of a thread pool able to execute closures, a similar approach to the logic of OpenMP tasks [], and to the version stability and maturity of the library.

4 Real-Time Parallel Programming for Rust

The proposed approach to integrate real-time parallel programming in Rust is to follow a similar approach as presented in Figure 1. As presented in the previous section, Rust already provides a significant number of packages for parallelism. By integrating in some of these packages the capability to (transparently) use the underlying operating systems real-time capabilities, real-time parallel programming can be easily integrated.

The approach aims for a high-level, yet customizable, abstraction framework in which it is possible to select a scheduling algorithm provided by the target operating system. Figure 2 shows an example of spawning a thread that will execute a given task (line 13) as convention of the `std::thread` standard library, extended with an extra argument: the scheduling algorithm and corresponding properties. In this example, we are using the SCHED_FIFO scheduling class of POSIX, specifying the priority of this task and without a period (lines 4 to 7). The scheduling algorithm and corresponding parameters are specified as a `SchedulingParameters` instance, where we specify the scheduling class, the target core, and the parameters of the class (lines 8 to 12). Although simple, this example provides the first steps on providing real-time thread scheduling.

Figure 3 considers the use of the SCHED_DEADLINE scheduling class of POSIX. This class expects three essential parameters (defined in lines 4 to 8): the usual runtime of the task, the period to trigger the task, and the deadline in which the task must be executed. The

```
1  fn some_task() {...}
2
3  fn main(){
4    let task_params = RT_Thread::ConstantBandwithServer {
5      runtime: Duration::from_millis(2),
6      period: Duration::from_millis(50),
7      deadline: Duration::from_millis(10)
8    };
9    let task_sched = RT_Thread::SchedulingParameters {
10     class: RT_Thread::SchedulingClass::CBS,
11     core: 0,
12     params: task_params
13   };
14   thread::spawn(some_task, task_sched);
15 }
```

Figure 3: Example of a thread with SCHED_DEADLINE, spawn in a specific core.

`SchedulingParameters` instance uses these parameters, the corresponding SCHED_DEADLINE class, and it specifies that this thread has to be executed in core 0. The spawn of the thread is done similar to the previous example, in which we specify the task and the scheduling parameters.

Figure 4 shows a simple example to create a `ThreadPool` with threads using the SCHED_DEADLINE scheduling class. The example shows two example for the expected extensions to the `threadpool` library, which expects an extra argument providing the task scheduling policy and parameters.

The example shows the spawn of a thread that uses the SCHED_DEADLINE scheduling class for the parent thread (lines 3 to 15). Then, inside this task, two thread pools are built. The first thread pool is built by simply inheriting the scheduling parameters of the parent task (line 18), which in this case is the use of SCHED_DEADLINE class with the parameters specified in lines 3 to 7.

The second approach is fully customized, in which we specify the scheduling parameters for each thread in the pool. Each thread has to be assign their scheduling parameters, as depicted in the previous examples. It is still possible to inherit the parameters from the parent thread by using a method that retrieves the current parameters (line 21 to 24). Lines 25 to 34 show two examples with distinct scheduling parameters, one using fixed priority, and another one with the SCHED_DEADLINE class.

After specifying the parameters and building the ThreadPool with those parameters, the ThreadPool can be used to execute tasks. A task can now be executed by specifying not only the task itself but, optionally, the thread in which it should be executed, as exemplified in line 39.

5 Conclusions

This paper presented a first proposal of extensions to the Rust programming language, in the form of a library, for real-time parallel programming. This approach intends to bring real-time functionalities to a parallel programming library of Rust, a feature not yet present in any of the existing parallel programming libraries. The approach allows the specification

```
1  fn main(){
2
3    let task_params = RT_Thread::ConstantBandwitdhServer {
4      runtime: Duration::from_millis(2),
5      period: Duration::from_millis(50),
6      deadline: Duration::from_millis(10),
7    };
8
9    let task_sched = RT_Thread::SchedulingParameters {
10     class: RT_Thread::SchedulingClass::CBS,
11     params: task_params
12   };
13
14
15   RT_Thread::create_periodic_task(task_sched, body: move | {
16     let n_workers = 4;
17
18     let pool = ThreadPool::new_inherited(n_workers);
19
20     let params = vec![];
21     let param_0 : RT_Thread::get_current_sched_params();
22     param_0.core = 0;
23
24     params.push( param_0);  // inherit parameters for core zero
25     params.push(SchedulingParameters { //fixed priority in core 1
26       class: SchedulingClass::FIFO,
27       core: 1,
28       params: FixedPriority {priority: 255}
29     });
30     params.push( SchedulingParameters { //CBS on core 2
31       class: SchedulingClass::CBS,
32       core: 2,
33       params: ...
34     });
35     // ...
36
37     let pool = ThreadPool::new_rt(n_workers, params);
38     ...
39     pool.execute(num_thread, some_task); //in specific thread
40   });
41 }
```

Figure 4: Example of a threadpool with specific scheduling algorithms per thread.

of parallel regions ale to execute real-time systems. We exemplify this use in a selected library, to be extended with scheduling policies capabilities. The next steps of this work includes finalizing and validating the approach with benchmarks commonly used in real-time systems validation.

6 Acknowledgments

This research has been co-funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 871669, in the context of the AMPERE project.

References

[1] P. T. Ward, *Structured development for real-time systems: Vol. I: Introduction and tools*. Pearson Education, 1986.

[2] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[3] A. Burns and A. J. Wellings, *Real-time systems and programming languages: Ada 95, real-time Java, and real-time POSIX*. Pearson Education, 2001.

[4] L. M. Pinho, V. Nélis, P. M. Yomsi, E. Quiñones, M. Bertogna, P. Burgio, A. Marongiu, C. Scordino, P. Gai, M. Ramponi, *et al.*, "P-socrates: A parallel software framework for time-critical many-core systems," *Microprocessors and Microsystems*, vol. 39, no. 8, pp. 1190–1203, 2015.

[5] M. A. Serrano, S. Royuela, and E. Quiñones, "Towards an openmp specification for critical real-time systems," in *Evolving OpenMP for Evolving Architectures: 14th International Workshop on OpenMP, IWOMP 2018, Barcelona, Spain, September 26–28, 2018, Proceedings 14*, pp. 143–159, Springer, 2018.

[6] N. D. Matsakis and F. S. Klock, "The rust language," *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 103–104, 2014.

[7] S. Klabnik and C. Nichols, *The Rust programming language*. No Starch Press, 2023.

[8] B. W. Kernighan and D. M. Ritchie, "The c programming language," 2002.

[9] D. Lewine, *POSIX programmers guide*. " O'Reilly Media, Inc.", 1991.

[10] A. Burns and A. Wellings, *Concurrent and real-time programming in Ada*. Cambridge University Press, 2007.

[11] P. Dibble, *Real-time Java platform programming*. Prentice Hall Professional, 2002.

[12] L. M. Pinho, E. Quinones, M. Bertogna, A. Marongiu, V. Nelis, P. Gai, and J. Sancho, *High-Performance and Time-Predictable Embedded Computing*. Wharton, TX, USA: River Publishers, 2018.

[13] R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald, *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[14] C. Maia, L. Nogueira, and L. M. Pinho, "Combining rtsj with fork/join: a priority-based model," in *Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems*, pp. 82–86, 2011.

[15] M. Schmid, F. Fritz, and J. Mottok, "Parallel programming in real-time systems," in *ARCS Workshop 2019; 32nd International Conference on Architecture of Computing Systems*, pp. 1–7, VDE, 2019.

[16] Siemens AG, "Embedded multicore building blocks." URL: https://embb.io/. Accessed: 06-03-2023.

[17] C. Pheatt, "Intel® threading building blocks," *Journal of Computing Sciences in Colleges*, vol. 23, no. 4, pp. 298–298, 2008.

[18] H. Kaiser, P. Diehl, A. S. Lemoine, B. A. Lelbach, P. Amini, A. Berge, J. Biddiscombe, S. R. Brandt, N. Gupta, T. Heller, *et al.*, "Hpx-the c++ standard library for parallelism and concurrency," *Journal of Open Source Software*, vol. 5, no. 53, p. 2352, 2020.

[19] V. Polevoy, "thread-priority v0.13.1." URL: https://crates.io/crates/thread-priority. Accessed: 06-03-2023.

A POSIX/RTEMS Monitoring Tool and a Benchmark to Detect Real-Time Scheduling Anomalies

Blandine Djika+*, Georges Kouamou*, Frank Singhoff+, Alain Plantec+
**University of Yaounde 1, Cameroon*
+Lab-STICC UMR CNRS 6285, University of Brest, France
email: blandine.djika@univ-brest.fr, georges.kouamou@gmail.com, frank.singhoff@univ-brest.fr, alain.plantec@univ-brest.fr

Abstract

This article deals with scheduling anomalies in real-time systems. We present MONANO, a POSIX user-level library allowing applications to dynamically detect a pre-identified set of real-time scheduling anomalies.

The MONANO library is based on the modelling of architecture and runtime constraints. MONANO monitors during the runtime the timing behavior of the application and deduces properties needed to identify scheduling anomalies.

We present also a benchmark to evaluate our approach. The benchmark is composed of several programs implementing the most frequent real-time scheduling anomalies.

Keywords: *Real-Time Scheduling Anomalies, Real-Time System, RTEMS, Cheddar.*

1 Introduction

This article focuses on scheduling anomalies in real-time systems. In real-time systems, tasks may have deadlines to meet. As defined by Luis Almeida in [1], a scheduling anomaly refers to a counter-intuitive phenomenon in which increasing the system resources or relaxing the application constraints can lead to missed deadline. While deadline can be verified at software design time, scheduling anomalies may arise at runtime depending on the dynamic behavior of the application.

[Problem Statement] In a previous work [2], we have proposed a model of scheduling anomalies composed of software architecture and runtime constraints. Architecture constraints, called *static* constraints, can be verified prior to execution. However, even after their verification, scheduling anomalies may occur at execution time and deadlines can be actually missed. We called *dynamic* constraints the conditions that may raise scheduling anomalies at runtime. To actually detect and properly handle scheduling anomalies, we have to monitor such dynamic constraints.

[Contribution of this article] In this article, we propose MONANO, a user-level monitoring library which can be used by an application to check dynamic constraints. This library is POSIX compliant. We are experimenting it on the RTEMS operating systems.

In case of an arising scheduling anomaly, MONANO signals the anomaly and allows the application to run specific actions to recover the anomaly. Specific actions can be any operation allowing the application to adapt itself as mixed-criticality theory promotes it [3]. We also provide MONANO with a benchmark implementing most of the real-time scheduling anomalies identified in the literature. We are using this benchmark to validate MONANO. The benchmark may also be used in any research activities related to real-time scheduling anomalies.

The remainder of this article presents background about scheduling anomalies in section 2. MONANO and its companion benchmark are introduced in section 3. Related works and the conclusion are finally presented respectively in section 4 and 5.

2 Background

In this section, we first present with an example what a scheduling anomaly is. Then, we introduce the overall approach we previously proposed in [2].

2.1 Scheduling anomalies

As defined in [1], *a scheduling anomaly* refers to a counter-intuitive phenomenon in which increasing the system resources or relaxing the application constraints can make the application unschedulable.

Let us illustrate a scheduling anomaly with an example from [4]. This example is composed of three periodic tasks scheduled with a non-preemptive fixed-priority scheduler. Each task τ_i is defined by a 5-tuples with its release time $r_i(\tau_i)$, its WCET (Worst Case Execution Time) $C_i(\tau_i)$, its deadline $D_i(\tau_i)$, its period $T_i(\tau_i)$ and its fixed priority $\pi_i(\tau_i)$. Following this notation, the tasks of figure 1 have the parameters given in Table 1 :

Figure 1 presents two schedules of such task set: (a) when the tasks are executed during all their WCET, i.e. each task has

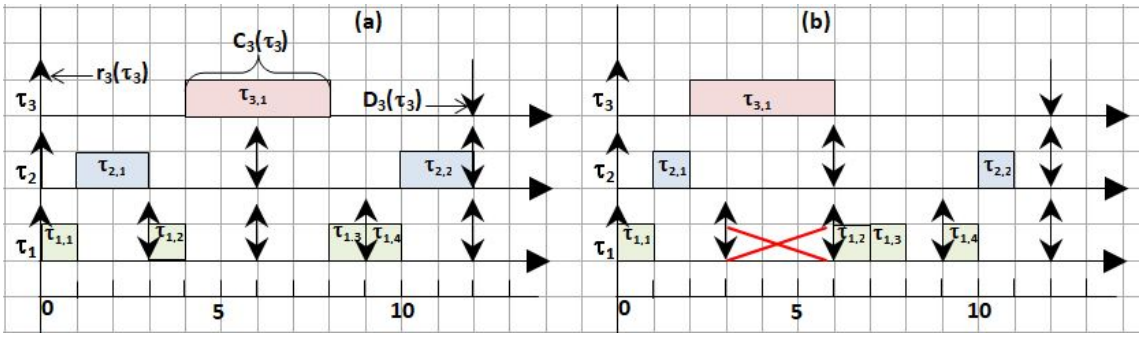

Figure 1: Anomaly when reducing task execution time

Tasks	$r_i(\tau_i)$	$C_i(\tau_i)$	$D_i(\tau_i)$	$T_i(\tau_i)$	$\pi_i(\tau_i)$
τ_1	0	1	3	3	1
τ_2	0	2	6	6	2
τ_3	0	4	12	12	3

Table 1: Task parameters

an execution time equal to its WCET, (b) when the execution times of the task are shortest than their WCET, which is the usual case when running such application. All task deadlines are met in figure 1 (a) while a deadline of τ_1 is missed in figure 1 (b). In figure 1 (b), we can notice that at time 1, τ_2 runs during 1 unit of time while its WCET is about 2 units of time. This real execution time shortest than its WCET, implies that τ_3 executes immediatly after τ_2 at time 2 and cannot be interrupted since the system is a non-preemptive one. Then, when τ_1 is released at time 3, although it is a higher priority task, it has to wait for τ_3 completion at time 6 before starting to work. This finally leads to a missed deadline for τ_1.

Scheduling anomalies were identified and classified by the community in seven types according to how they may occur. Table 2 summarizes them.

Num	**Types**
1	Reducing the task execution time [5,6,7,8,9,10,11]
2	Changing task priorities [5]
3	Weakening task precedence constraints [5,6]
4	Increasing processor speed [10]
5	Delaying the execution of the tasks [10]
6	Increasing task period [12, 13, 14]
7	Increasing the number of processors of the execution platform [5]

Table 2: Types of scheduling anomalies

2.2 Modeling scheduling anomalies

Each above scheduling anomalies arises under specific conditions. Such conditions can be modeled as constraints related to the architecture of the real-time system and its behavior at runtime.

In [2], we proposed to model scheduling anomalies according to two types of complementary constraints: *static* constraints and *dynamic* constraints. Static constraints are only related to the architecture design specification. They can be verified prior to execution. We have identified 9 static constraints related to the execution platform and 8 related to the task models. Dynamic constraints are related to particular events that actually raise scheduling anomalies. The verification of such dynamic constraints can only be done at runtime. We have identified the dynamic constraints of each scheduling anomaly described in table 2.

2.3 Proposed anomaly analysis

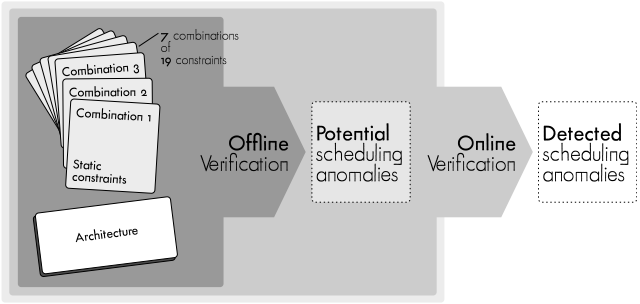

Figure 2: Analysis approach

As shown in figure 2, we propose in [2] a two-step analysis process to detect scheduling anomalies:

1. First, at design time, static constraints are verified with Cheddar [15]. When static constraints hold, it means that scheduling anomalies may occur when dynamic constraints hold at runtime.

2. Second, at runtime, to assess scheduling anomaly, a second analysis is required to check dynamic constraints. In the sequel, we describe MONANO, a monitoring user-level library to verify such dynamic constraints.

3 MONANO monitoring service design and its companion benchmark

This section presents MONANO, a user-level library written in C. Currently, it is implemented on top of RTEMS (Figure 3) but the library is POSIX compliant, then it could be used on many POSIX real-time operating systems. To verify MONANO, we implemented a benchmark based on ROSACE to raise scheduling anomalies identified in the literature. This benchmark is also briefly described in the sequel.

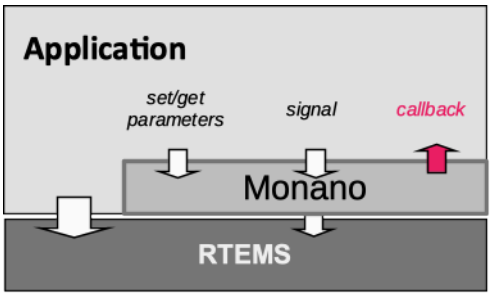

Figure 3: MONANO user-level POSIX library

3.1 In a nutshell

MONANO provides services to create and monitor periodic threads.

MONANO requires application code instrumentation : programmers have to call MONANO services in their programs to monitor the timing behavior of their threads.

MONANO relies on POSIX to create threads, to schedule threads, to implement periodic releases and to monitor threads.

MONANO maintains a static as well as a dynamic view of each monitored thread. MONANO API requires applications to specify the static view of the threads before starting them.

The detection of scheduling anomalies relies on an oracle capable of evaluating the dynamic constraints at runtime. To detect scheduling anomalies during runtime, the oracle checks if dynamic constraints hold for each thread. Dynamic constraints are expressed on runtime events that are either monitored by MONANO or pointed out by the application itself. The monitoring is stopped when the threads are completed.

As described in [2], runtime events that may occur during a thread lifetime and that are part of dynamic constraints leading to scheduling anomalies are priority, precedence or period changes, departure or completion of thread periodic job, or processor speed change.

Regarding scheduling anomaly detection, the interaction between the application and MONANO relies on a specific callback. During its initialization, the application may register a callback that is called by MONANO when an anomaly is detected. The anomaly type together with the involved thread are passed as the callback arguments.

This callback is generated by an oracle integrated into MONANO implementing dynamic constraints leading to scheduling anomalies.

3.2 Application Programming Interface

Figure 4 shows a diagram that depicts the main API elements. The library is composed of two parts: the MONANO manager and views of the application threads. A thread view consists in a *pthread_monano_t* associated with a *pthread_monano_attr_t* struct type instance.

pthread_monano_attr_t stores, for each thread, a set of attributes that are classic static task parameters in real-time scheduling theory. Parameters maintained for each thread are

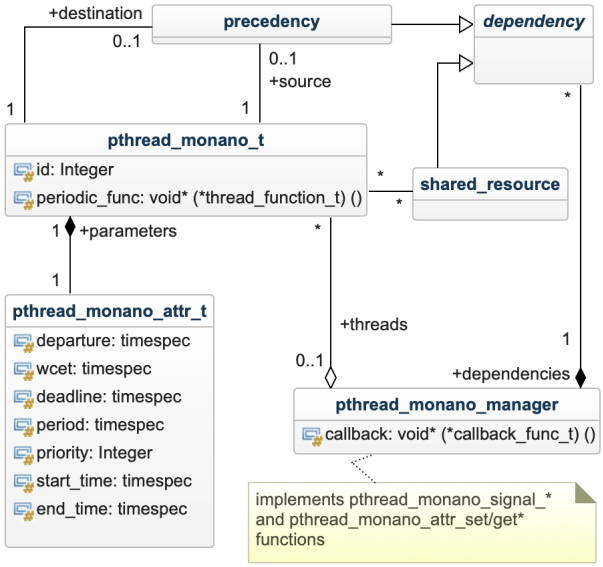

Figure 4: MONANO design

```
1 #include "monano.h"
2
3 struct pthread_monano_t my_monano;
4 struct pthread_monano_attr_t my_attr;
5 pthread_monano_id_t id;
6
7 void * my_callback(int anomaly_number,
      pthread_monano_id_t tid) {
8   printf ("Anomaly %d is detected in thread number
      %d\n", anomaly_number, tid);
9   exit (0) ;
10 }
11
12 void * a_periodic_thread(void * arg) {
13   pthread_monano_signal_departure_time( & my_monano, id);
14   /* Run its periodic program */
15   ...
16   pthread_monano_signal_end_time( & my_monano, id);
17   return NULL;
18 }
19
20 void * POSIX_Init(void * argument) {
21   struct timespec period, wcet, ...;
22   int priority = ...;
23
24   pthread_monano_attr_setperiod( & my_attr, period);
25   pthread_monano_attr_setwcet( & my_attr, wcet);
26   pthread_monano_attr_setpriority( & my_attr, priority ) ;
27   pthread_monano_register_anomaly_callback( & my_monano,
      my_callback);
28
29   pthread_monano_periodic_thread_create( & my_monano, &
      my_attr, a_periodic_thread, & id, NULL);
30   return NULL;
31 }
```

Figure 5: MONANO program example

namely, the *wcet*, the *deadline*, the *period* which are set as POSIX *timespec structs*. The *priority* is also set as an integer. MONANO also maintains dynamic parameters of each thread such as the last *release time* and the last *completion time* of the thread. Notice that MONANO is only currently supporting periodic threads.

The MONANO manager is in charge of creating threads and of detecting scheduling anomalies. The manager provides 2 functions to respectively create threads and register the callback C function that is called when a scheduling anomaly occurs.

Functions allowing the application to send events to MONANO are named *pthread_monano_signal* in the figure 4.

3.3 MONANO program example

Figure 5 shows a MONANO example program. Lines 24 to 26 setup the static parameters of the MONANO thread to create and line 27 registers the callback that is run when a scheduling anomaly occurs. In this example, the function $my\_callback$ is called in case of a scheduling anomaly. After such initialization steps, the program creates the MONANO periodic threads and starts to run. If an anomaly occurs, then the callback $my\_callback$ is called and the RTEMS application is stopped with the $exit$ function. Notice that the code of the thread notifies two events to MONANO: when the thread starts to run (with the *pthread_monano_signal_departure_time* function) and when it completes (with the *pthread_monano_signal_end_time* function).

3.4 Benchmark to investigate scheduling anomalies in real-time systems

The first goal of the benchmark is to enforce situations in which scheduling anomalies arise. The second goal is to serve as an extensible example to verify the correctness and the usability of MONANO itself.

In the literature, there is today no benchmark gathering programs that raise the more frequent known scheduling anomalies for real-time systems. In this section, we present the MONANO benchmark that contributes to fulfill such a need.

As explained previously, detecting scheduling anomalies requires verifying if both static and dynamic constraints hold for a given application. The different services implemented in MONANO intend to verify them.

Scheduling anomalies may arise both in uniprocessor and multiprocessor architectures. Several scenarios can lead to a type of scheduling anomaly. We have 19 scenarios for the 7 types of scheduling anomalies identified in both uniprocessor and multiprocessor systems. However in this article, we only focus on uniprocessor systems and then, the benchmark only handles the 5 anomalies occuring in uniprocessor systems. The 5 uniprocessor anomalies occur in 9 scenarios. Each scenario is implemented by a program in the benchmark.

The current MONANO benchmark only implements the dynamic constraints (DC) related to one of the 5 uniprocessor anomalies:

- constraint D1 becomes true when MONANO detects that the thread execution time is reduced (see scheduling anomaly 1 in Table 2).
- constraint D2 becomes true when MONANO detects that a thread has changed its current priority (see scheduling anomaly 2 in Table 2).
- constraint D3 becomes true when MONANO detects that a thread dependency was not met (see scheduling anomaly 3 in Table 2).
- constraint D4 becomes true when MONANO detects that the processor speed has changed (see scheduling anomaly 4 in Table 2).
- and the constraint D5 becomes true when MONANO detects that a thread execution is delayed (see scheduling anomaly 5 in Table 2).

To implement the MONANO benchmark, we adapted ROSACE, an open-source avionic control command software developed by [16] in C. First, ROSACE was adapted to run on RTEMS with the POSIX API. Second, we implemented 9 different versions of ROSACE corresponding to the 9 uniprocessor scheduling anomaly scenarios. Each of these 9 programs is a ROSACE program modified to comply with the static and the dynamic constraints of the related scheduling anomaly.

Table 3 summarizes, for the 9 programs their static constraints and their dynamic constraints that the program implements.

DC	Programs	Static constraints
D1	P1	Threads may have precedence constraints
	P2	Non-preemptive scheduling
	P3	Deadline Monotonic scheduling
		Threads may access shared resources
	P4	EDF scheduling
		Threads are asynchronously released
		Threads may be suspended
D2	P5	Threads are independent
D3	P6	Threads may have precedence constraints
D4	P7	Threads are asynchronously released
		Threads may access shared resources
	P8	Deadline Monotonic scheduling
		Threads are asynchronously released
		Threads may access shared resources
D5	P9	Threads may access shared resources
		Threads may be suspended

Table 3: Static and dynamic constraints of each program of the benchmark

We have implemented the 9 programs on a uniprocessor preemptif fixed priority scheduling RTEMS target. One may notice that some of the programs of Table 3 require a different scheduling policy (e.g. program P4 requires a EDF scheduling). Furthermore, ROSACE is implemented by a set of threads that communicate by flow of data. One may notice

also that in Table 3 some of the programs are not compliant with this ROSACE implementation (.e.g program P9 uses shared resources to thread communications).

To experiment all types of scheduling anomalies on the same RTEMS/POSIX operating system and with the same application baseline (ROSACE), we have implemented in the 9 programs specific mechanisms to enforce all static constraints. For example, with programs 4 and 9, scheduling anomalies will occur when threads are suspended while the original ROSACE program does not suspend any thread. To implement suspended threads and to actually raise the corresponding scheduling anomalies, threads of programs 4 and 9 are blocked on a counting POSIX semaphore. Table 4 gives a short description of what we have implemented to make static constraints compliant with ROSACE and RTEMS.

Constraints	Implementation
Precedence constraints	Communication implemented with a counting semaphore initialized to 0.
Deadline monotonic scheduling	Priority assignment with $setschedparam$ according to the deadline
Shared resources	Implemented by POSIX mutexes
Non-preemptive scheduling	Non-preemptive scheduling enforced with a mutex shared by all threads
Threads asynchronously released	Release times are delayed with $nanosleep$
Suspended threads	Threads are suspended with $nanosleep$

Table 4: Implementation of the static constraints on POSIX/uniprocessor RTEMS

The static and the dynamic constraints rely on various data that are either given at application startup by the programmer or measured by MONAO during execution or computed at runtime by MONANO. Table 5 gives for each scheduling anomaly data that are either monitored by MONANO or either computed by MONANO. Let consider $S = \{\tau_1, ..., \tau_n\}$, a set of n periodic threads as defined in section 2.1 and monitored by MONANO. Table 5 presents each data continuously updated by MONANO during runtime and if they are computed or measured :

- $start\_time(\tau_i)$ and $end\_time(\tau_i)$ are respectively the start time and the end time of each job of τ_i.
- $execution\_time(\tau_i)$ is the real execution time of a each job of τ_i.
- $blocking\_time(\tau_i)$ is the computed blocking time of τ_i on the shared resources.
- $preemption\_time(\tau_i)$ is the amount of time τ_i is preempted by threads with a higher priority level.
- $suspending\_time(\tau_i)$ is the amount of time τ_i has decided to suspend itself.
- $priority(\tau_i)$ is the current priority of τ_i.
- $dependencies\_list$ stores the real execution order of the threads that are constrained by thread precedencies.
- $processor\_speed$ is the current processor speed.

DC	Measured data	Computed data
D1	$start\_time(\tau_i)$, $end\_time(\tau_i)$	$execution\_time(\tau_i)$, $blocking\_time(\tau_i)$, $preemption\_time(\tau_i)$
D2	$priority(\tau_i)$	
D3	Threads execution order in $dependencies\_list$	Missed precedency constraints
D4	Current processor speed	
D5	$start\_time(\tau_i)$, $end\_time(\tau_i)$	$suspending\_time(\tau_i)$

Table 5: Metrics monitored on POSIX/uniprocessor RTEMS by MONANO

4 Related work

Previous research on scheduling anomalies has mostly focused on identifying and presenting different types of scheduling anomalies in both uniprocessor and multiprocessor systems [5, 6, 7, 8, 9, 10, 11, 12, 13, 14].

However, detecting anomalies also requires to detect events occuring when scheduling anomalies are raised. Many works have investigated how to monitor events in real-time systems and some of them could be applied to detect scheduling anomalies.

First, monitoring tools devoted to stream-based systems were developed by the community. For example, Copilot [17, 18] monitors systems by regularly capturing values (called samples) of variables of the system. The overall values of a given variable constitute a data stream on which verifications can be applied. As Copilot, RTLola [19, 20] also operates on data streams and monitors them. However, in the case of RTLola, the software components charged to monitor are generated from a specification written in a formal language. StreamLAB [21] is another framework example using RTLola for monitoring purposes.

R2U2(Realizable, Responsive, Unobtrusive Unit) [22, 23] is focusing on the monitoring of security properties for Unmanned Aerial Systems (UAS) built on FPGA. The objective is to detect security attacks.

Hili [24] proposed a model-based approach to monitor real-time systems during their runtime. The approach provides a means to integrate and configure various monitors.

Yibing [25] identifies variations between predicted behavior and monitored behavior. The method is using a digital twin.

Finally, Reinier [26] proposed MuSADET, a tool that looks for timing anomalies in event traces for real-time systems. The framework classifies anomalies with metrics between event arrivals.

As seen above, many authors have carried out technics for monitoring real-time systems during their execution, but without, most of the time, focusing specifically on the detection of scheduling anomalies.

In the contrary, we propose in this article a monitoring tool specifically devoted to detecting scheduling anomalies of a real-time system during runtime.

5 Conclusion

This article focuses on scheduling anomalies in real-time systems. The context in which an anomaly may occur have been studied and presented in [2]. In [2], a model of scheduling anomalies composed of software architecture and runtime constraints has been proposed. From this model, we proposed in this article a monitoring tool called MONANO which is able to detect runtime constraints leading to real-time scheduling anomalies. We are experimenting MONANO in Cheddar [15]. MONANO comes with a benchmark prototyped in RTEMS. We are currently running experiments to evaluate MONANO performance and intrusivity.

To fully evaluate MONANO, the ability to handle *false positive* and *false negative* anomalies is necessary to avoid inappropriate callback invocations. For now, *false positive* and *false negative* anomalies are not taken into account. A *false positive* result would occurs if an analysis gives an invalid positive result (an anomaly is detected whereas it should not be detected), and a *false negative* result would occurs if an analysis gives a invalid negative result (an anomaly is not detected whereas it should be detected). In the current state of our work, we are not able to provide such an analysis. In our future work, we plan to experiment with more use cases to improve dynamic anomaly analysis to include *false positive* and *false negative* detection.

For this article, we focused on uniprocessor systems. We also expect to improve the MONANO tool by integrating the detection of scheduling anomalies in multiprocessor systems.

Finally, another future work would be to investigate how MONANO could be used to monitor other real-time properties.

6 Artefact

MONANO and its companion benchmark are available at `http://beru.univ-brest.fr/svn/CHEDDAR/trunk/src/framework/scheduling_anomalies`

References

[1] L. Almeida, P. Pedreiras, and R. Marau, "Traffic scheduling anomalies in temporal partitions," in *From Model-Driven Design to Resource Management for Distributed Embedded Systems: IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006), October 11–13, 2006, Braga, Portugal*, pp. 95–104, Springer, 2006.

[2] B. Djika, F. Singhoff, A. Plantec, and G. E. Kouamou, "Work-in-progress: Models and tools to detect real-time scheduling anomalies," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, pp. 540–543, IEEE, 2021.

[3] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.

[4] G. Phavorin, P. Richard, J. Goossens, T. Chapeaux, and C. Maiza, "Scheduling with preemption delays: anomalies and issues," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pp. 109–118, 2015.

[5] R. Graham, "Bounds on the performance of scheduling algorithms," *Computer and job scheduling theory*, pp. 165–227, 1976.

[6] M. Richard, P. Richard, E. Grolleau, and F. Cottet, "Contraintes de précédences et ordonnancement monoprocesseur," in *Proc. Real Time and Embedded Systems (RTS 2002)*, pp. 121–138, 2002.

[7] S. Pailler, *Analyse hors ligne d'ordonnançabilité d'applications temps réels comportant des tâches conditionnelles et sporadiques*. PhD thesis, Poitiers, 2006.

[8] R. Ha and J. W. Liu, "Validating timing constraints in multiprocessor and distributed real-time systems," in *14th international conference on distributed computing systems*, pp. 162–171, IEEE, 1994.

[9] F. Ridouard, P. Richard, F. Cottet, and K. Traore, "Some results on scheduling tasks with self-suspensions," *Journal of Embedded Computing*, vol. 2, no. 3-4, pp. 301–312, 2006.

[10] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer Science & Business Media, 2011.

[11] P. Richard, "On the complexity of scheduling real-time tasks with self-suspensions on one processor," in *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, pp. 187–194, IEEE, 2003.

[12] J. Goossens, "Introduction à l'ordonnancement temps réel multiprocesseur," in *École d'été" Temps réel"*, 2007.

[13] B. Andersson and J. Jonsson, "Preemptive multiprocessor scheduling anomalies," in *Proceedings 16th International Parallel and Distributed Processing Symposium*, pp. 8–pp, IEEE, 2002.

[14] B. Andersson, "Static-priority scheduling on multiprocessors.," 2004.

[15] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *Proceedings of the 2004 annual ACM SIGAda international conference on Ada: The engineering of correct and reliable software for real-time & distributed systems using Ada and related technologies*, pp. 1–8, 2004.

[16] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron, "The rosace case study: From simulink specification to multi/many-core execution," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 309–318, IEEE, 2014.

[17] I. Perez, F. Dedden, and A. Goodloe, "Copilot 3," tech. rep., 2020.

[18] L. Pike, A. Goodloe, R. Morisset, and S. Niller, "Copilot: A hard real-time runtime monitor," in *International Conference on Runtime Verification*, pp. 345–359, Springer, 2010.

[19] P. Faymonville, B. Finkbeiner, M. Schwenger, and H. Torfah, "Real-time stream-based monitoring," *arXiv preprint arXiv:1711.03829*, 2017.

[20] J. Baumeister, B. Finkbeiner, S. Gumhold, and M. Schledjewski, "Real-time visualization of stream-based monitoring data," in *International Conference on Runtime Verification*, pp. 325–335, Springer, 2022.

[21] P. Faymonville, B. Finkbeiner, M. Schledjewski, M. Schwenger, M. Stenger, L. Tentrup, and H. Torfah, "Streamlab: stream-based monitoring of cyber-physical systems," in *International Conference on Computer Aided Verification*, pp. 421–431, Springer, 2019.

[22] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "Runtime analysis with r2u2: a tool exhibition report," in *Runtime Verification: 16th International Conference, RV 2016, Madrid, Spain, September 23–30, 2016, Proceedings 7*, pp. 504–509, Springer, 2016.

[23] P. Moosbrugger, K. Y. Rozier, and J. Schumann, "R2u2: monitoring and diagnosis of security threats for unmanned aerial systems," *Formal Methods in System Design*, vol. 51, pp. 31–61, 2017.

[24] N. Hili, M. Bagherzadeh, K. Jahed, and J. Dingel, "A model-based architecture for interactive run-time monitoring," *Software and Systems Modeling*, vol. 19, pp. 959–981, 2020.

[25] Y. Li, Z. Tao, L. Wang, B. Du, J. Guo, and S. Pang, "Digital twin-based job shop anomaly detection and dynamic scheduling," *Robotics and Computer-Integrated Manufacturing*, vol. 79, p. 102443, 2023.

[26] R. Torres Labrada, "Multi-signal anomaly detection for real-time embedded systems," Master's thesis, University of Waterloo, 2020.

Safety of the Intended Functionality Concept Integration into a Validation Tool Suite

Víctor J. Expósito Jiménez  ***, Bernhard Winkler, Joaquim M. Castella Triginer***
Virtual Vehicle Research GmbH, Inffeldgasse 21A, 8010, Graz, Austria; Tel: +43 316 873 9001; email: {victor.expositojimenez,bernhard.winkler,joaquim.castellatriginer}@v2c2.at

Heiko Scharke, Hannes Schneider
AVL List GmbH, Hans-List-Platz 1, 8020, Graz, Austria; Tel: +43 316 787 0; email: {heiko.scharke, hannes.schneider}@avl.com

Eugen Brenner, Georg Macher 
Graz University of Technology, Rechbauerstraße 12, 8010, Graz, Austria; Tel: +43 316 873 6401; email: {brenner, georg.macher}@tugraz.at

Abstract

Nowadays, the increasing complexity of Advanced Driver Assistance Systems (ADAS) and Automated Driving (AD) means that the industry must move towards a scenario-based approach to validation rather than relying on established technology-based methods. This new focus also requires the validation process to take into account Safety of the Intended Functionality (SOTIF), as many scenarios may trigger hazardous vehicle behaviour. Thus, this work demonstrates how the integration of the SOTIF process within an existing validation tool suite can be achieved. The necessary adaptations are explained with accompanying examples to aid comprehension of the approach.

Keywords: SOTIF, Safety of the Intended Functionality, Scenario Validation, ADAS/ADS

1 Introduction

Scenario validation plays a significant role in the entire vehicle validation process as an increasing number of features and safety systems rely on sensors. Unlike functional safety [1] or cybersecurity [2], which covers failures and malfunctions, and external attacks respectively, the Safety of the Intended Functionality (SOTIF) standard [3] focuses on the technical shortcomings and human misuses that may result in hazardous behaviour at vehicle level. Its focus is to increase the identification of hazardous scenarios to be validated as well as to minimise the area in which unknown hazardous scenarios could appear. Figure 1 shows the cause-and-effect model in which is depicted how a potential triggering condition could result in a hazardous behaviour at the end of the process. According to the ISO21448, a triggering condition is a *"specific condition of a scenario that serves as an initiator for a subsequent system reaction contributing to either a hazardous behaviour or an inability to prevent or detect and mitigate a reasonably foreseeable indirect misuse. The concept of 'triggering' includes the possibility that there can be multiple conditions that can gradually happen, leading to hazardous behaviour or the inability to prevent or detect and mitigate a reasonably foreseeable misuse. The term "potential triggering condition" can be used when the ability to initiate a corresponding reaction is not yet established"*. Another concise definition is given in [4], where a triggering condition is defined as *"an external condition (relative to ego-vehicle) in a scenario that triggers one or multiple functional insufficiencies and further results in hazardous behaviour. They are system-dependent was well"*. The SOTIF standard also defines a performance insufficiency as a *"limitation of the technical capability contributing to a hazardous behaviour or inability to prevent or detect and mitigate reasonably foreseeable indirect misuse when activated by one or more triggering conditions*. Examples of performance insufficiencies could be the limitation of the actuation or the perception range of the sensor used detect objects. Consequently, a functional insufficiency is defined as an *insufficiency of specification or performance insufficiency*. Finally, the definition of hazard is adapted from the given in the ISO26262, *"potential source of harm caused by malfunctioning behaviour of the item"*. SOTIF standard replaces the word "malfunctioning" by "hazardous" and the phrase "of the item" by "at the vehicle level" in comparison with the given by the ISO26262 to adapt the definition to the scope of the standard. For clarification, the insufficiencies of specification are out of the scope of the related project to this work, therefore, a functional insufficiency is considered the same as a functional insufficiency in this work as is shown in the box of the project scope in Figure 1. The inclusion of a triggering condition could start a reaction in the system that could activate a functional insufficiency and could finally result in a hazardous behaviour. A hazardous behaviour is defined based on the result of the Key Performance Indicators (KPIs) or Safety Performance Indicators (SPIs). These metrics are used to discern if the result of the tests is within a defined tolerable value or, on the other hand, is outside the tolerable windows and is set as a hazardous behaviour. A KPI

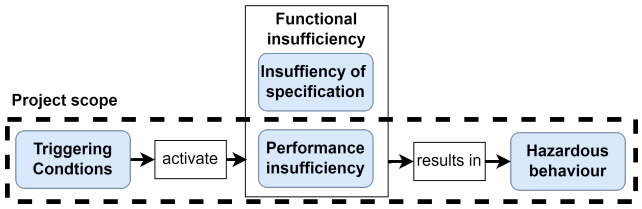

Figure 1: Cause-and-effect model from ISO21448

is a metric that is used for measure a specific parameter of the system. In a similar way, a SPI defines a metric but focused on the safety domain such as the Minimum Safe Distance Violation (MSDV) or the Time-to-Collision (TTC) [5]

The Operational Design Domain (ODD) is a crucial concept in the scenario validation process. While it is defined in the SOTIF standard, the definition provided in the UL4600 standard [6] outlines more precisely what an ODD constitutes from our perspective. As per this standard, an ODD refers to *"the set of environments and situations the item is intended to operate within. This includes not only direct environmental conditions and geographic restrictions, but also a characterization of the set of objects, events, and other conditions that will occur within that environment"*. The scenario development for the ODDs utilises the widely accepted methodology of The 6-Layer Scenario Model [2] [7]. This model splits the definition of each scenario into six layers, each concentrating on the context of the scenario. The layers and their definitions are:

- *Layer 1 – Road network and traffic guidance objects*: e.g. road markings, and traffic signs and traffic lights.
- *Layer 2 – Roadside structures*: e.g. buildings, vegetation, streets lamps, and advertising boards.
- *Layer 3 – Temporary modifications of L1 and L2*: e.g. roadwork signs, temporary markings, and covered markings.
- *Layer 4 – Dynamic Objects*: e.g. vehicles (moving and non-moving), pedestrians (moving and non-moving), trailers, and animals.
- *Layer 5 – Environmental Conditions*: e.g. illumination, precipitation, and road weather.
- *Layer 6 – Digital information*: e.g. state of traffic lights, switchable traffics signs, and V2X messages.

In this context, attempts have been made to define a taxonomy that can describe most scenarios in the most detailed manner. For example, the one from the British Standards Institution [8] or the taxonomy from the Society of Automotive Engineers (SAE) [9] are widely recognized. Additionally, Annex B of the SOTIF standard also addresses this matter.

Once the main concepts have been outlined, the following section describes the integration process into the validation tool. Finally, section 3 provides a summary of all the work presented in this publication and establishes the direction for future research.

2 SOTIF Concept Integration Process

The integration has been implemented within the AVL SCENIUS™ [10] tool suite that is developed with the scenario-based validation approach in mind. The complete validation process is covered, from scenario design to scenario management, test case generation, test allocation, and result reporting. The suite is based on three main tools modules. First, the scenario designer allows the user to handle all aspects of the scenario including the parametrisation. It fully supports ASAM OpenScenario [11] and OpenDrive [12]. All scenarios are immediately verified for standard conformity as well as by the enhanced data and logic checks. Then, the user could manage all stored scenarios in the scenario data manager. All the elements relevant for the sufficient description of scenarios such as road content, traffic content, and other environmental data are managed and stored in a central database. Finally, the test case generator provides the user the possibility of defining test orders in a simulation or transfer to another different execution environment. The implemented smart testing algorithms enable the automatic reduction of the vast amount of test cases and parameter variations. In addition to the main benefits provided by the tool suite such as time-cost saving, efficiency, fast integration and traceability; the inclusion of the SOTIF concept extends and improves the identification and validation of both unknown and known hazardous scenarios of a ADAS/AD function to obtain a more precise safety argumentation.

An ontology is used to describe the scenarios that will be used for testing. ASAM OpenXOntology [13] is used as a reference, but modifications are included to better fit the requirements of the tool chain. Internally, the ontology and its relationships are defined by using four kind of entities:

- *Node:* A node is the entity in which the hierarchy of the ontology is built. It can be a child of another node, or a parent for Enums or Params. Examples of nodes are the ambient or weather conditions of the scenario, which are the parents of scenario parameters such as rain or illumination parameter.
- *Enum:* Defines a list of values that are related to each other. For example, an enum is the snowfall condition, which is defined by three different levels of severity: heavy snow, light snow, and moderate snow.
- *Value:* Defines an entity that is an abstraction of a phenomenon, but it is not yet modelled. In this case, the phenomenon already exits in the system but has not yet been parametrised. As example, this type of entity are the one previously mentioned: heavy snow, light snow, and moderate snow.
- *Param:* Defines an entity that can be quantified. Each one is associated with an unit to be measured. For example, scenario illuminance, which is associated with lux units.

The integration of the SOTIF concept requires the addition of a new node in the system, which is the parent of all triggering conditions defined in the validation tool. The next integration step is to linking the triggering conditions

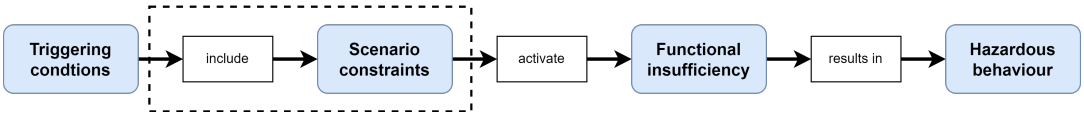

Figure 2: Updated cause-and-effect model including the scenarios constraints

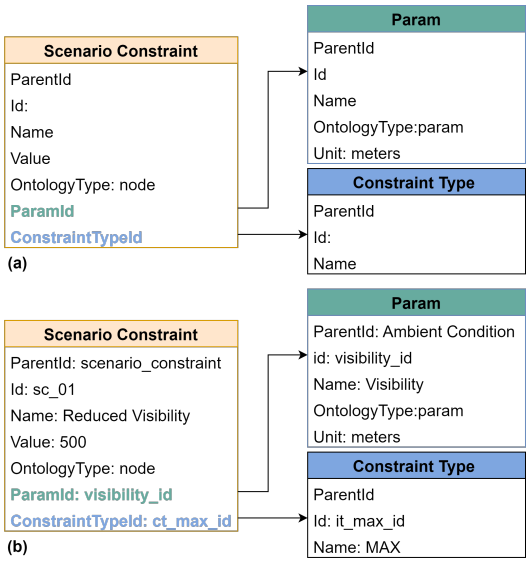

Figure 3: Definition of a scenario constraint

with the existing defined ontology to be able to parametrise each triggering condition and include them into the test scenario. Therefore, an intermediate block has to be added to the cause-and-effect model, which is shown in Figure 2. In this new approach, a scenario constraint block is added, which connects the triggering conditions with the performance insufficiencies. By using this approach, we are able to define each triggering conditions as a combination of one or many scenario constraints. Following the scenario constraint parametrisation shown in Figure 4(a), a scenario constraint is a node entity that is linked to a *Param* and a new entity called *Constraint Type*, which sets the type of constraint such as a maximum or a minimum value. The *Param* entity is associated to the existing ontology of the system. For example, a heavy snow condition is defined as a visibility limitation up to 500 meters according to BSI PAS 1883 standard [8]. This scenario constraints is defined in Figure 4(b), where relationship and value is given in the *Node* entity that, in turn, is associated to the *Param* from the ontology (visibility) and the type of constraint (MAX). The complete hierarchy and relationship tree of the system for this specific triggering condition is given in Figure 4.

The SOTIF concept is designed with scalability in mind due to the test scenario shall increase the complexity of the triggering conditions and their parametrization. The Figure 5 shows the triggering condition *Heavy Snow during Night-time* as an example of this increasing complexity. These types of triggering conditions are treated as the combination of two independent triggering conditions: *Heavy Snow* + *Night-time*. In contrast to the previous example, the weather condition *Heavy Snow* is more finely parametrised. In this context, not only is the impact in the visibility is considered, but also the effect on the scenario illumination and the asphalt friction. Following the standards again, the illuminance in a heavy snow scenario can be parametrised from 1 lux to 2000 lux. Additionally, a reduction factor of 0.8 is applied on the asphalt friction in this potential triggering condition. However, this particular triggering condition occurs during night-time, therefore, the illuminance condition due to night-time is also applied (illuminance less than 1 lux according to the standards). Therefore, there are two illuminance constraints in this triggering condition parametrization. In this situation, the most limiting conditions is applied. It means that in this definition, the illuminance parametrised in night-time overrides the value of the illuminance parametrised in the heavy snow condition.

Finally, when one or more potential triggering conditions are selected in the scenario for testing, the scenario constraints (e.g., limited visibility, reduced friction...) associated to each potential triggering condition are also included in the generated test cases. The resulting metrics of the matrix test cases show the impact of the selected potential triggering conditions on the function, which are compared with the nominal performance of the function (i.e., no potential triggering conditions included) to determine them not longer as potential but triggering conditions for the function, and to identify the thresholds at which they are relevant to impact and effect on the function output.

3 Conclusions and future work

In this publication, the integration of a SOTIF concept has been explained, where some adaptations and parametrisation of the scenario constraints have to be done in order to integrate triggering conditions into an existing scenario ontology. As a first step, an extensive list of potential triggering conditions has been investigated based on current state-of-the-art and available standards. They are then parametrised by using an existing system ontology, which is used to model the scenarios and defining the needed entities and relationship to be able to link the triggering conditions, scenario constraints, and the existing ontology.

As a future task, we will define the triggering conditions that cannot be parametrised using existing standards. Furthermore, we are researching a methodology that will allow us to capture the majority of potential triggering conditions from the perception side based on the performance insufficiencies due to the infinite number of triggering conditions in the real world, which are not possible to cover manually.

4 Acknowledgement

The publication was written at Virtual Vehicle Research GmbH in Graz, Austria. The authors would like to acknowledge the financial support within the COMET K2

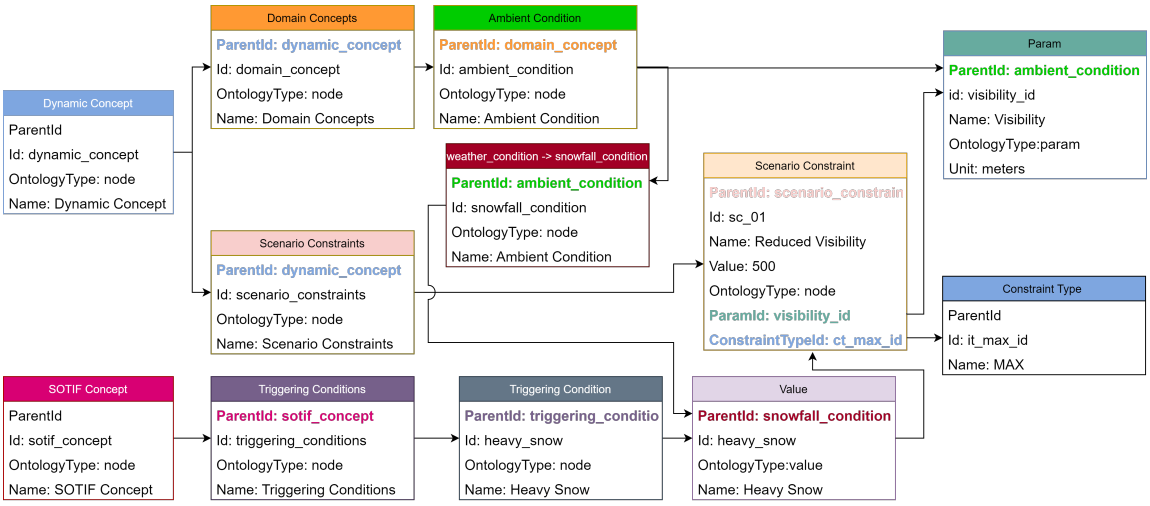

Figure 4: Heavy Snow triggering condition in SCENIUS tool suite

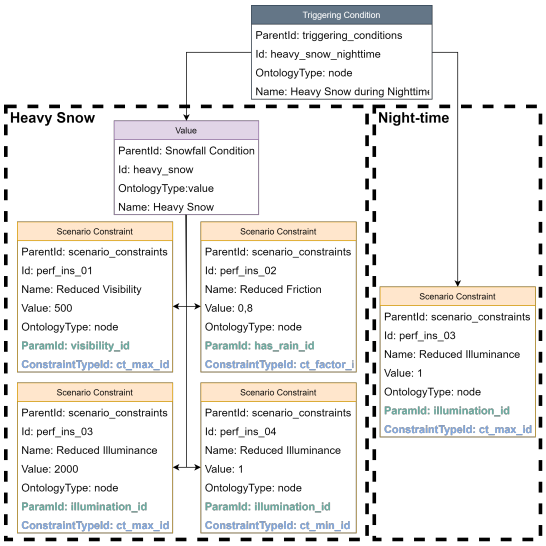

Figure 5: Extended Triggering Condition: "Heavy Snow during Night-time"

Competence Centers for Excellent Technologies from the Austrian Federal Ministry for Climate Action (BMK), the Austrian Federal Ministry for Labour and Economy (BMAW), the Province of Styria (Dept. 12) and the Styrian Business Promotion Agency (SFG). The Austrian Research Promotion Agency (FFG) has been authorised for the programme management. They would furthermore like to express their thanks to their supporting industrial project partner, namely AVL List GmbH.

References

[1] "ISO26262: Road vehicles — Functional safety," standard, International Organization for Standardization, Geneva, CH, 2018.

[2] "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," standard, SAE International, 2021.

[3] "ISO 21448:2022 Road vehicles — Safety of the intended functionality," standard, International Organization for Standardization, Geneva, CH, 2022.

[4] Z. Zhu, R. Philipp, C. Hungar, and F. Howar, "Systematization and identification of triggering conditions: A preliminary step for efficient testing of autonomous vehicles," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, pp. 798–805, 2022.

[5] J. Vaicenavicius, T. Wiklund, A. Grigaite, A. Kalkauskas, I. Vysniauskas, and S. D. Keen, "Self-driving car safety quantification via component-level analysis," *SAE International Journal of Connected and Automated Vehicles*, vol. 4, pp. 35–45, mar 2021.

[6] "Ul4600 standard for safety: Evaluation of autonomous products," standard, International Organization for Standardization, 2021.

[7] M. Scholtes, L. Westhofen, L. R. Turner, K. Lotto, M. Schuldes, H. Weber, N. Wagener, C. Neurohr, M. H. Bollmann, F. Körtke, J. Hiller, M. Hoss, J. Bock, and L. Eckstein, "6-layer model for a structured description and categorization of urban traffic and environment," *IEEE Access*, vol. 9, pp. 59131–59147, 2021.

[8] "BSI PAS 1883:2020-Operational design domain (ODD) taxonomy for an automated driving system (ADS). Specification," standard, The British Standards Institution, 2020.

[9] "AVSC Best Practice for Describing an Operational Design Domain: Conceptual Framework and Lexicon," standard, SAE International, 2020.

[10] AVL List GmbH, "AVL SCENIUS." `https://www.avl.com/en/testing-solutions/automated-and-connected-mobility-testing/avl-scenius`, 2023. Accessed: 2023-09-17.

[11] "ASAM OpenScenario," standard, ASAM e. V., 2022.

[12] "ASAM OpenDrive," standard, ASAM e. V., 2022.

[13] "ASAM OpenXOntology," standard, ASAM e. V., 2022.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard

Ada-Deutschland

Dr. Hubert B. Keller CEO
ci-tec GmbH
Beuthener Str. 16
76139 Karlsruhe
Germany
+491712075269
Email: h.keller@ci-tec.de
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch

Ada-Europe Sponsors

Ada Edge

27 Rue Rasson
B-1030 Brussels
Belgium
Contact:Ludovic Brenta
ludovic@ludovic-brenta.org

AdaCore

46 Rue d'Amsterdam
F-75009 Paris
France
sales@adacore.com
www.adacore.com

AdaLabs
innovate.all

506 Royal Road
La Caverne, Vacoas 73310
Republic of Mauritius
Contact: David Sauvage
david.sauvage@adalabs.com

ADA LOG

2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

Deep Blue Capital

Jacob Bontiusplaats 9
1018 LL Amsterdam
The Netherlands
Contact: Wido te Brake
wido.tebrake@deepbluecap.com
www.deepbluecap.com

Ellidiss Technologies

24 Quai de la Douane
29200 Brest, Brittany
France
Contact: Pierre Dissaux
pierre.dissaux@ellidiss.com
www.ellidiss.com

KONAD
Software for Control and Administration

In der Reiss 5
D-79232 March-Buchheim
Germany
Contact: Frank Piron
info@konad.de
www.konad.de

PTC®
Developer Tools

3271 Valley Centre Drive,Suite 300
San Diego, CA 92069
USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tools

SYSADA

Enterprise House
Baloo Avenue, Bangor
North Down BT19 7QT
Northern Ireland, UK
enquiries@sysada.co.uk
sysada.co.uk

systerel
Safe real-time solutions

1090 Rue René Descartes
13100 Aix en Provence
France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/

Tid rum

Tiirasaarentie 32
FI 00200 Helsinki
Finland
Contact: Niklas Holsti
niklas.holsti@tidorum.fi
www.tidorum.fi

WhiteElephant

Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch


Ada
europe