

The journal for the international
Ada community

Ada User Journal



Volume 45
Number 2
June 2024

Editorial	71
Quarterly News Digest	72
Conference Calendar	80
Forthcoming Events	87

Articles from the AEIC 2024 Work-in-Progress Session

V. Manjunath, M. Baunach <i>A Framework for Improving Portability and Ensuring Correctness of Operating System Kernels</i>	89
F. Siebert, M. Lill, M. Teufel <i>Algebraic Effects and Static Analysis for Safety-Critical Applications in Fuzion</i>	94
S. Levieux, F. Singhoff, S. Rubini, P. Plasson, P.V. Gouel, L.R. Malac-Allain, L. Miné, G. Brusq <i>An Iterative Benchmark Configuration Method for Quantifying Multi-core Interference</i>	99
G. Rincon, C.F. Nicolas, T. Poggi <i>Improving Reliability in a Robotic Application without Loss of Safety</i>	105
L. Sousa, J. Cecílio, P. M. Ferreira, A. Oliveira de Sá <i>Reconfigurable and Scalable Honeynet for Cyber-physical Systems</i>	109
J. Cecílio, A. Oliveira de Sá, A. Souto <i>Software-Based Security Framework for Edge and Mobile IoT</i>	113
P. Pirkelbauer, C. Liao, P.H. Lin, D. Wright, C. Reynolds, D. Quinlan <i>Supporting Ada in the ROSE Compiler</i>	118
M. Samadi, T. Carvalho, L. M. Pinho, S. Royuela <i>Task-to-Thread Mapping in OpenMP Using Fuzzy Decision Making</i>	124

Produced by Ada-Europe

Editor in Chief

António Casimiro

University of Lisbon, Portugal
AUJ_Editor@Ada-Europe.org

Ada User Journal Editorial Board

Luís Miguel Pinho

Associate Editor

Polytechnic Institute of Porto, Portugal

lmp@isep.ipp.pt

Jorge Real

Deputy Editor

Universitat Politècnica de València, Spain

jorge@disca.upv.es

Patricia López Martínez

Assistant Editor

Universidad de Cantabria, Spain

lopezpa@unican.es

Dirk Craeynest

Events Editor

KU Leuven, Belgium

Dirk.Craeynest@cs.kuleuven.be

Alejandro R. Mosteo

News Editor

Centro Universitario de la Defensa, Zaragoza, Spain

amosteo@unizar.es

Ada-Europe Board

Tullio Vardanega (President)

University of Padua

Italy

Dirk Craeynest (Vice-President)

Ada-Belgium & KU Leuven

Belgium

Dene Brown (General Secretary)

SysAda Limited

United Kingdom

Ahlan Marriott (Treasurer)

White Elephant GmbH

Switzerland

Luís Miguel Pinho (Ada User Journal)

Polytechnic Institute of Porto

Portugal

António Casimiro (Ada User Journal)

University of Lisbon

Portugal



Ada-Europe General Secretary

Dene Brown

SysAda Limited

Signal Business Center

2 Innotec Drive

BT19 7PD Bangor

Northern Ireland, UK

Tel: +44 2891 520 560

Email: Secretary@Ada-Europe.org

URL: www.ada-europe.org

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 45

Number 2

June 2024

Contents

	<i>Page</i>
Editorial Policy for Ada User Journal	70
Editorial	71
Quarterly News Digest	72
Conference Calendar	80
Forthcoming Events	87
Articles from the AEiC 2024 Work-in-Progress Session	
V. Manjunath, M. Baunach <i>“A Framework for Improving Portability and Ensuring Correctness of Operating System Kernels”</i>	89
F. Siebert, M. Lill, M. Teufel <i>“Algebraic Effects and Static Analysis for Safety-Critical Applications in Fuzion”</i>	94
S. Levieux, F. Singhoff, S. Rubini, P. Plasson, P. V. Gouel, L. R. Malac-Allain, L. Miné, G. Brusq <i>“An Iterative Benchmark Configuration Method for Quantifying Multi-core Interference”</i>	99
G. Rincon, C. F. Nicolas, T. Poggi <i>“Improving Reliability in a Robotic Application without Loss of Safety”</i>	105
L. Sousa, J. Cecílio, P. M. Ferreira, A. Oliveira de Sá <i>“Reconfigurable and Scalable Honeynet for Cyber-physical Systems”</i>	109
J. Cecílio, A. Oliveira de Sá, A. Souto <i>“Software-Based Security Framework for Edge and Mobile IoT”</i>	113
P. Pirkelbauer, C. Liao, P. H. Lin, D. Wright, C. Reynolds, D. Quinlan <i>“Supporting Ada in the ROSE Compiler”</i>	118
M. Samadi, T. Carvalho, L. M. Pinho, S. Royuela <i>“Task-to-Thread Mapping in OpenMP Using Fuzzy Decision Making”</i>	124
Ada-Europe Associate Members (National Ada Organizations)	128
Ada-Europe Sponsors	Inside Back Cover

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

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

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a

wider audience. This includes papers published in North America that are not easily available in Europe.

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

I would like to start this editorial by mentioning that there are still no new developments concerning the merge between the Ada User Journal and Ada Letters. On the other hand, significant efforts are ongoing for the creation of the new organization that I mentioned in my previous editorial: the Ada User Society. It is to expect that this organization will soon be founded, operating alongside Ada-Europe to keep the Ada language alive, as a recognized standard. This is surely a relevant topic to which we will keep paying attention.

Also to be highlighted is the fact that we had the 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) taking place this month, in Barcelona, Spain. It was a very well-organized conference, hosting several tutorials and workshops, and including diverse social events that, despite the unexpected rain, allowed the participant to taste the Catalan culture while providing further interaction opportunities. Therefore, not surprisingly, this issue provides the Proceedings of the AEiC 2024 Work-in-Progress session, which includes 8 papers. They address diverse topics relevant to the conference, notably methods and tools for software analysis and verification (applicable to operating system kernels and safety-critical applications), a method for quantifying task interference in multi-core systems, an approach for designing robot systems with improved availability while meeting safety goals, solutions for improving the security of cyber-physical and IoT systems, developments for adding Ada support to the ROSE source-to-source compiler, and work towards the performance improvement of task-to-thread mapping in OpenMP. Despite being presented as work-in-progress, we believe the reader will appreciate the high quality and even maturity of these works.

For the News Digest section, Alejandro R. Mosteo, its editor, selected the most relevant news concerning the past three months, collected from the relevant newsgroups. For the Calendar and Events section, Dirk Craeynest collected information from several sources to let us know which relevant events for the Ada community will be happening over the forthcoming months. We hope you can profit from their work!

Finally, I would like to call your attention for the announcement included in page 87. It is about the 29th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2025), which will take place on June 10-13, 2025, in Paris, France.

*Antonio Casimiro
Lisboa
June 2024
Email: AUJ_Editor@Ada-Europe.org*

Quarterly News Digest

Alejandro R. Mosteo

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

Contents

Preface by the News Editor	72
Ada-related Events	72
Ada-related Resources	76
Ada-related Tools	77
Ada Practice	78

[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,

In this quiet period leading up to the summer holidays, there are a couple of more meaty topics in the News Digest. For one, you can peruse the final & detailed information about the latest Ada-Europe conference [1]. You can even find on-line recordings of some of the sessions, as is the case for the Ada Developers Workshop [2]. (Link included in the thread.)

Times are always a-changin', and Alire is now a strong player in the Ada open source community. Not everyone has jumped on the train, and in this issue you can get an idea on the complexities of contributing your projects with the lending hand of a maintainer [3].

Sincerely,
Alejandro R. Mosteo.

- [1] "Ada-Europe Int. Conf. Reliable Software Technologies, AEiC 2024", in Ada-related Events.
- [2] "Ada Developers Workshop @ AEiC 2024, Speaker and Talk List", in Ada-related Events
- [3] "Software Engineer Seeks Compatible Cratifier", in Ada Practice.

Ada-related Events

Ada-Europe Int. Conf. Reliable Software Technologies, AEiC 2024

From: Dirk Craeynest
<dirk@orka.cs.kuleuven.be>
Subject: Ada-Europe Int. Conf. Reliable Software Technologies, AEiC 2024
Date: Tue, 16 Apr 2024 10:55:23 -0000
Newsgroups: comp.lang.ada,
fr.comp.lang.ada, comp.lang.misc

Call for Participation

28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024)

11-14 June 2024, Barcelona, Spain

www.ada-europe.org/conference2024

*** Online registration open! ***

*** Extensive info on conference site ***

Organized by Ada-Europe and Barcelona Supercomputing Center (BSC), in cooperation with ACM SIGAda, ACM SIGBED, ACM SIGPLAN, and Ada Resource Association (ARA), supported and sponsored by ASCENDER project, ACM-W, Eurocity, AdaCore, Rising STARS project, ACM-W Barcelona Chapter, and OpenMP

#AEiC2024 #AdaEurope
#AdaProgramming

*** Early registration discount until May 20 ***

*** Highly recommended to book your hotel ASAP ***

Exciting News!

Preparations for AEiC 2024, the 28th Ada-Europe International Conference on Software Reliable Technologies, are well underway!

Registrations opened Tuesday April 16, and we've got some exciting offers lined up for you, courtesy of our generous sponsors Rising STARS and AdaCore, as well as an inspiring program.

See below for an overview, and visit our website for more details about accepted

contributions, registration fees, social events and many more.

General Information

The 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) will take place in Barcelona, Spain. The conference schedule comprises a keynote and an invited talk, a panel with invited experts, a journal track, an industrial track, a work-in-progress track, a vendor exhibition, parallel tutorials and hackaton, and satellite workshops. There will be time for networking during breaks and lunches, as well as various social events.

AEiC 2024 is the latest in a series of annual international conferences started in the early 80's, under the auspices of Ada-Europe, the international organization that promotes knowledge and use of the Ada programming language and reliable software in general, into academic education and research, and industrial practice.

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

The 2024 edition of the conference continues a number of important innovations started in previous years:

- reduced conference registration fee for all authors;
- low registration fee for all tutorials and workshops;
- journal-based open-access publication model for peer-reviewed papers;
- compact program with two core days (Wednesday & Thursday);
- tutorials on Tuesday, followed by welcome event for all participants;
- workshops on Friday, optional chill event on Thursday evening

Overview of the Week

- Tue 11: six half-day tutorials, full-day hackaton, welcome reception

- Wed 12: core technical program, conference banquet
- Thu 13: core technical program, post conference chill-out
- Fri 17: four full-day workshops

Extensive information on AEiC 2024 is on the conference website, such as an overview of the program, the list of accepted papers and presentations, and descriptions of workshops, tutorials, hackaton, keynote and invited presentations, panel, and social events. Also check the conference site for registration, accommodation and travel information. The Advance Program brochure will be available there as well.

Venue

The conference will take place in UPC Campus Nord, easily accessible by metro from the airport and city centre. If you can stay over before or after the conference, there's a lot to see around. Check the Practical Information section of the conference website for more info.

Invited Speakers

This year the conference will feature a keynote talk on the first day, and a panel with three invited speakers on the second, plus an invited talk. All will address topics of relevance in the conference scope, with time for questions and answers.

- On Wed June 12, a keynote talk by Francisco J. Cazorla and Jaume Abella, from Barcelona Supercomputing Center, who will talk about "Strategies to Build Safety Relevant High-Performance HW/SW Platforms for Critical Embedded Systems".
- On Thu June 13, a panel on "AI for Safety-Critical Systems: How 'I' Should the AI be?", moderated by Cristina Seceleanu, Mälardalen University, with three invited experts: Kerstin Bach (Norwegian University of Science and Technology), Irune Yarza (Ikerlan), Marta Barroso (Barcelona Supercomputing Center).
- And an invited talk by Rosa Maria Badia, Barcelona Supercomputing Center, on "Simplifying the Life-Cycle Management of Complex Application Workflows".

Conference Core Composition

The core conference program features three distinct types of technical presentations, with different duration, in addition to the keynote talk and the panel session: journal-track talks (25 minutes), industrial-track talks (15 minutes), work-in-progress-track talks (10 minutes).

All papers presented in the journal track, the industrial track and the work-in-progress track have undergone peer

review. Presentations are combined into by-theme and not by-track sessions, in order that authors and participants alike enjoy all flavors of the program in a mixed as opposed to segregated combination.

Papers and Presentations:

- 8 sessions with a mix of presentations on specific topics
- 13 journal-track talks
- 8 work-in-progress reports
- 5 industrial presentations and experience reports
- submissions from around the world
- accepted contributions by authors from Belgium, China, France, Germany, India, Italy, Portugal, Spain, Sweden, UK, USA

Tutorials

Six three-hour tutorials are offered on Tuesday 11th:

- "Lock-Free Programming in Ada-2022: Implementing a Work-Stealing Scheduler for Ada-2022's Light-Weight Parallelism", by S. Tucker Taft, AdaCore, USA
- "Ada for Business Applications", by Gautier de Montmollin, Ada-Switzerland, Switzerland
- "Rust Fundamentals", by Luis Miguel Pinho and Tiago Carvalho, ISEP, Portugal
- "Concurrency and Parallelism in Rust", by Luis Miguel Pinho and Tiago Carvalho, ISEP, Portugal
- "Modeling Concurrent State Machines in TLA+", by J. Germán Rivera, Tesla, USA
- "Introduction to the Development of Safety-Critical Software", by Jean-Pierre Rosen, Adalog, France
- "METASAT: Programming High Performance RISC-V Technologies for Space", by Leonidas Kosmidis, Barcelona Supercomputing Center, Alejandro Calderon, Ikerlan, Aridane Alvarez Suarez, fentISS, Lorenzo Lazzara, Collins Aerospace, Eckart Göhler, OHB
- "Introduction to Certifiable General Purpose GPU Programming for Safety-Critical Systems", by Leonidas Kosmidis, Barcelona Supercomputing Center, Rod Burns and Verena Beckham, Codeplay/Intel

as well as a "hackaton":

- "Optimizing AI-driven Workflows within a Mission-Critical Cyber-Physical System", if you're keen to explore the latest AI techniques for Adaptive Optics applications in giant telescopes with Damien Gratadour, Observatoire de Paris, CNRS, France

Satellite Events

Four workshops are held on Friday 14th:

- 3rd ADEPT workshop "AADL by its practitioners"
- 9th International Workshop on "Challenges and New Approaches for Dependable and Cyber-Physical System Engineering" (DeCPS 2024)
- "Enabling the use of AI in Safety-Critical Systems"
- "Ada Developers Workshop", an informal yet dynamic gathering for developers in the Ada community to meet, share insights, and present their latest projects or project updates, using the Ada programming language and Ada-related technology

Social Program

The conference provides several opportunities to socialize:

- Each day: coffee breaks and lunches offer ample time for interaction and networking with participants and vendors.
- Tuesday early evening: welcome reception at the picturesque gardens of Torre Girona. Guests will be treated to a curated selection of local wines paired with the globally renowned Iberian ham, and an array of delectable appetizers representing the rich culinary heritage of Catalonia and Spain. Attendees will have the unique opportunity to explore the cutting-edge facilities of the Barcelona Supercomputing Center, and marvel at its latest addition, the Marenostrum V supercomputer, and its predecessor, Marenostrum IV, housed within the historic chapel of Torre Girona.
- Wednesday evening: Conference Banquet at the emblematic restaurant "7 portes". Attendees will have the opportunity to savor the finest flavors of the Catalan and Mediterranean cuisines, such as the renowned "Paella Perallada", a masterpiece that harmoniously combines semi-dry rice with succulent peeled shellfish, delectable seafood and tender meats. With a history spanning over 180 years, "7 portes" stands as a witness to the evolution of some of the most illustrious artists of their time, including Pablo Picasso and Antoni Tàpies. Their presence has left an indelible mark, forming a captivating small art gallery within the restaurant's walls, waiting to be discovered by guests.
- Thursday evening: Chill event at the Moritz Barcelona Brewery, the brewery of the first beer of Barcelona. The event is divided in three parts: a visit to the brewery, a welcome drink at the Brasserie room, offering an exclusive vantage point overlooking the maceration tanks, and a banquet served

within the same Brasserie room, by renowned chef Jordi Vilà, adorned with a Michelin star, promising a gastronomic experience to be savored and remembered.

Further Information

Registration:

- registration information is provided at <http://www.ada-europe.org/conference2024/registration.html>
- early registration discount until Monday May 20, 2024
- payment possible by credit card or bank transfer
- special low conference fee for authors
- discount for Ada-Europe, ACM SIGAda, SIGBED and SIGPLAN members
- registration includes coffee breaks, lunches and social events
- low tutorial and workshop fees for all participants
- strong discount on all fees for students
- minimal fee for AI Hackaton and Ada Developers Workshop
- see registration page for all details

Promotion:

- recommended Twitter hashtags: #AEiC2024 #AdaEurope #AdaProgramming

AEiC 2024 Sponsors:

- Barcelona Supercomputing Center: <https://www.bsc.es/>
- ASCENDER project: <https://www.bsc.es/research-and-development/projects/ascender-arquitectura-software-para-entornos-de-computo-continuo>
- ACM-W: <https://women.acm.org/>
- Eurocity: <https://eurocity.be/>
- AdaCore: <https://www.adacore.com/>
- Rising STARS project: <https://risingstars-project.eu/>
- ACM-W Barcelona Chapter: https://twitter.com/BCN_ACM_W
- OpenMP: <https://www.openmp.org/>

The conference is supported and sponsored by

- Ada-Europe: <http://www.ada-europe.org/>

and organized in cooperation with

- ACM SIGAda: <http://www.sigada.org/>
- ACM SIGBED: <https://sigbed.org/>
- ACM SIGPLAN: <http://www.sigplan.org/>
- ARA: <https://www.adaic.org/community/>

Please make sure you book accommodation as soon as possible. For more info and latest updates see the conference website at <http://www.ada-europe.org/conference2024>.

We look forward to seeing you in Barcelona in June 2024!

Our apologies if you receive multiple copies of this announcement.

Please circulate widely.

Dirk Craeynest, AEiC 2024 Publicity Chair

Dirk.Craeynest@cs.kuleuven.be

* 28th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2024)

* June 11-14, 2024, Barcelona, Spain, www.ada-europe.org/conference2024

(V7.1)

Ada Monthly Meetup, May 2024

From: Fernando Oleo / Irvise

irvise_ml@irvise.xyz

Subject: Ada Monthly Meetup, May 2024

Date: Wed, 17 Apr 2024 12:29:46 +0200

Newsgroups: comp.lang.ada

I would like to announce the May (2024) Ada Monthly Meetup which will be taking place on the 11th of April at 13:00 UTC time (15:00 CEST). As always, the meetup will take place over at Jitsi. The Meetup will also be livestreamed to Youtube.

If someone would like to propose a talk or a topic, feel free to do so! We currently have one talk that will be given by A.J. Ianozi about GetAda[1], an all-batteries included installer for Ada tooling in a short single command!

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 Jitsi Meet 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://www.getada.dev/>

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

Date: Wed, 17 Apr 2024 14:49:57 +0200

> I would like to announce the May (2024) Ada Monthly Meetup which will be taking place on the 11th of April

Presumably, you meant May 4th... (1st saturday of May)

From: Fernando Oleo / Irvise

irvise_ml@irvise.xyz

Date: Wed, 17 Apr 2024 16:27:36 +0200

> Presumably, you meant May 4th... (1st saturday of May)

Ouch! My mistake, thank you for noticing it! But it is indeed the 11th of *May*, so April -> May. I changed it to the second Saturday since I will most likely not be available the first week due to some holidays here in Spain :)

[...]

From: Fernando Oleo / Irvise

irvise_ml@irvise.xyz

Date: Thu, 9 May 2024 22:19:26 +0200

This is a kind reminder that the next Ada Monthly Meetup will take place this Saturday, so in less than 48h!

The main topics to be talked about are going to be AJ's GetAda and the Ada Developers Workshop that will take place on the 14th of June!

There will be no Ada Monthly Meetup in June due to the aforementioned Workshop.

Ada Developers Workshop @ AEiC 2024, Speaker and Talk List

From: Fernando Oleo / Irvise

irvise_ml@irvise.xyz

Subject: Ada Developers Workshop @ AEiC 2024, Speaker and Talk list

Date: Wed, 15 May 2024 13:03:51 +0200

Newsgroups: comp.lang.ada

[The video recordings and slides from the workshop are now available via <http://www.ada-europe.org/conference2024/adadev.html> —arm]

The list of speakers, talks and schedule for the Ada Developer Workshop has been published [1]. The Workshop will take place on the 14th of June in Barcelona. The entry has a small price of 10€ if paid before the 20th of May and 20€ if later. The price covers the cost of two coffee breaks and a meal. You can find more information in the registration page [2]

Here is a short summary of what will be shown during the workshop:

- "SweetAda: a Multi-architecture Embedded Development Framework" by Gabrielle Galeotti (Italy), Fernando Oleo Blanco (Spain)
- "Avoiding Access Types" by Jeffrey R. Carter (Belgium)
- "G-NAV: Soaring the Clouds with AdaWebPack" by Guillermo A. Hazenbrouck (Belgium)

- "Alire 2.0: a Quality of Life Update" by Alejandro Mosteo (Spain)
- "HiRTOS: a Multicore RTOS Written in SPARK Ada" by J. German Rivera (USA)
- "Ironclad: a Formally Verified OS Kernel Written in SPARK/Ada" by Cristian Simon (Spain)
- "An Ada Story of Time" by Jean-Pierre Rosen (France)
- "Controlled I/O: a Library for Scope-Based Files" by Jeffrey R. Carter (Belgium)
- "Ada Community Advocacy" by Fernando Oleo Blanco (Spain)

[1] <http://www.ada-europe.org/conference2024/adadev.html>

[2] <http://www.ada-europe.org/conference2024/registration.html>

Best regards,

The Ada Developers Workshop organisation team

From: Dirk Craeynest

<dirk@orka.cs.kuleuven.be>

Date: Tue, 28 May 2024 10:15:33 -0000

Update: thanks to sponsoring, in-person participants will only pay the low registration fees specified above; remote participants are required to register as well, but participation will be totally free!

Dirk Craeynest

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

* 28th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2024)

* June 11-14, 2024, Barcelona, Spain, www.ada-europe.org/conference2024

From: Dirk Craeynest

<dirk@orka.cs.kuleuven.be>

Date: Sat, 1 Jun 2024 10:10:55 -0000

Remember that registration is still open for the Ada Developers Workshop #AdaDevWS at the #AdaEurope conference #AEiC2024 in Barcelona on Friday 14 June 2024.

Some places remain for in-person participation, many remain for remote participation. :-)

Dirk Craeynest

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

* 28th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2024)

* June 11-14, 2024, Barcelona, Spain, www.ada-europe.org/conference2024

Ada-Europe - AEiC 2024 Early Registration Deadline Imminent

From: Dirk Craeynest

<dirk@orka.cs.kuleuven.be>

Subject: Ada-Europe - AEiC 2024 early registration deadline imminent

Date: Sat, 18 May 2024 11:01:38 -0000

Newsgroups: comp.lang.ada,
fr.comp.lang.ada, comp.lang.misc

UPDATED Call for Participation

*** Early registration DEADLINE
May 20 ***

28th Ada-Europe International
Conference on Reliable Software
Technologies (AEiC 2024)

11-14 June 2024, Barcelona, Spain

www.ada-europe.org/conference2024

* Extensive info and registration online *

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

#AEiC2024 #AdaEurope
#AdaProgramming

Organized by Ada-Europe and Barcelona Supercomputing Center (BSC), in cooperation with ACM SIGAda, ACM SIGBED, ACM SIGPLAN, and Ada Resource Association (ARA), supported and sponsored by ASCENDER project, Eurocity, Collins Aerospace, ACM-W, BSC Severo Ochoa Center of Excellence, AdaCore, Rising STARS project, ACM-W Barcelona Chapter, and OpenMP

UPDATE

Ada-Europe - AEiC 2024 early registration deadline imminent

Come to the Ada-Europe conference in Barcelona, experience a packed program in an exciting town, benefit from tutorials or a hackaton on Tuesday, join a workshop on Friday, enjoy the social events and some sightseeing!

Register now: discounted fees until May 20!

<<http://www.ada-europe.org/conference2024/registration.html>>

Extra conference sponsorship allows for an extremely low 10 EUR fee for the AI Hackaton on Tuesday and the Ada Developers Workshop on Friday!

See below for an overview, and visit our website for more details about accepted contributions, registration fees, social events and many more.

[General Information, Overview of the Week, Venue, Invited Speakers, Conference Core Composition, Tutorials,

Satellite Events and Social Program sections are the same as in "Ada-Europe Int. Conf. Reliable Software Technologies, AEiC 2024" in this AUJ issue, pp. 70-72 —arm]

*** Further Information

Registration:

- registration information is provided at <<http://www.ada-europe.org/conference2024/registration.html>>
- early registration discount until Monday May 20, 2024
- payment possible by credit card or bank transfer
- special low conference fee for authors
- discount for Ada-Europe, ACM SIGAda, SIGBED and SIGPLAN members
- registration includes coffee breaks, lunches and social events
- low tutorial and workshop fees for all participants
- strong discount on all fees for students
- minimal fee for AI Hackaton and Ada Developers Workshop
- see registration page for all details

Promotion:

- recommended Twitter hashtags: #AEiC2024 #AdaEurope #AdaProgramming

The conference is organized by:

- Ada-Europe <<http://www.ada-europe.org/>>
- Barcelona Supercomputing Center <<https://www.bsc.es/>>

in cooperation with:

- ACM SIGAda <<http://www.sigada.org/>>
- ACM SIGBED <<http://www.sigbed.org/>>
- ACM SIGPLAN <<http://www.sigplan.org/>>
- Ada Resource Association (ARA) <<http://www.adaic.org/community/>>

supported and sponsored by:

- ASCENDER Project: <<https://www.bsc.es/research-and-development/projects/ascender-arquitectura-software-para-entornos-de-computo-continuo>>
- Eurocity <<https://eurocity.be/>>
- Collins Aerospace <<https://www.collinsaerospace.com/>>
- ACM-W <<https://women.acm.org/>>
- BSC Severo Ochoa Center of Excellence <<https://www.bsc.es/news/bsc-news/bsc-achieves-severo-ochoa-center-excellence-accreditation>>
- AdaCore <<https://www.adacore.com/>>

- Rising STARS project
<<https://risingstars-project.eu/>>
- ACM-W Barcelona Chapter
<https://twitter.com/BCN_ACM_W>
- OpenMP <<https://www.openmp.org/>>

Please make sure you book accommodation as soon as possible.

For more info and latest updates see the conference website at
<<http://www.ada-europe.org/conference2024>>.

We look forward to seeing you in Barcelona in June 2024!

Our apologies if you receive multiple copies of this announcement.

Please circulate widely.

Dirk Craeynest, AEiC 2024 Publicity Chair

Dirk.Craeynest@cs.kuleuven.be

* 28th Ada-Europe Int. Conf. Reliable Software Technologies (AEiC 2024)

* June 11-14, 2024, Barcelona, Spain, www.ada-europe.org/conference2024

(V8.1)

2024 Ada-Belgium General Assembly

From: Dirk Craeynest

<dirk@orka.cs.kuleuven.be>

Subject: 2024 Ada-Belgium General Assembly

Date: Tue, 28 May 2024 08:52:59 -0000
Newsgroups: comp.lang.ada

To all Ada-Belgium members who didn't register yet for the 2024 Ada-Belgium General Assembly meeting, to be held online Tuesday 28 May 2024 18:30 CEST, please check your mailbox for the convocation that was sent some time ago, and register ASAP.

Ada-related Resources

[Delta counts are from May 28th to July 11th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo

<amosteo@unizar.es>

Subject: Ada on Social Media

Date: 11 Jul 2024 18:24 CET[d]

To: Ada User Journal readership

Ada groups on various social media:

- Reddit: 8_741 (+36) members [1]
- LinkedIn: 3_521 (+11) members [2]
- Stack Overflow: 2_411 (+6) questions [3]

- Gitter: 258 (+5) people [4]
- Ada-lang.io: 241 (+21) users [5]
- Telegram: 205 (+4) users [6]
- Libera.Chat: 73 (-2) concurrent users [7]

- [1] <https://old.reddit.com/r/ada/>
- [2] <https://www.linkedin.com/groups/114211/>
- [3] <https://stackoverflow.com/questions/tagged/ada>
- [4] https://app.gitter.im/#/room/#ada-lang_Lobby:gitter.im
- [5] <https://forum.ada-lang.io/u>
- [6] https://t.me/ada_lang
- [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: 11 Jul 2024 18:27 CET[c]

To: Ada User Journal readership

- GitHub: >740* (=) developers [1]
- Rosetta Code: 979 (+29) examples [2]
- 42 (=) developers [3]
- Alire: 412 (+7) crates [4]
- 1_068 (+20) releases [5]
- Sourceforge: 252 (+1) projects [6]
- Open Hub: 214 (=) projects [7]
- Codelabs: 57 (=) repositories [8]
- Bitbucket: 37 (-1) repositories [9]

*This number is a 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] ``alr search --list --full``
- [6] <https://sourceforge.net/directory/language:ada/>
- [7] <https://www.openhub.net/tags?names=ada>
- [8] https://git.codelabs.ch/?a=project_index
- [9] <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: 11 Jul 2024 18:38 CET[d]

To: Ada User Journal readership

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

- TIOBE Index: 24 (-2) 0.78% (-0.05%) [1]
- PYPL Index: 17 (+2) 0.96% (+0.14%) [2]
- Languish Trends: 192 (-12) 0.00% (-0.01%) [3]
- Stack Overflow Survey: 42 (=) 0.77% (=) [4]
- IEEE Spectrum (general): 36 (=) Score: 0.0107 (=) [5]
- IEEE Spectrum (jobs): 29 (=) Score: 0.0173 (=) [5]
- IEEE Spectrum (trending): 30 (=) Score: 0.0122 (=) [5]

- [1] <https://www.tiobe.com/tiobe-index/>
- [2] <http://pypl.github.io/PYPL.html>
- [3] <https://tjpalmer.github.io/languish/>
- [4] <https://survey.stackoverflow.co/2023/>
- [5] <https://spectrum.ieee.org/top-programming-languages/>

We Are Hiring Software Engineers

From: Björn Lundin <bnl@nowhere.com>

Subject: We are hiring software engineers

Date: Thu, 25 Apr 2024 16:11:31 +0200

Newsgroups: comp.lang.ada

This is not technical, just a note that we are hiring.

We are hiring software engineers to work on our WCS (Warehouse Control System). It means communication with automation devices, and making them move pallets or totes in a way that we and customers are happy with.

We have mostly largish customers, but some less large as well.

It is circa 1.2 Mloc of Ada - a mix from Ada83 -> Ada22.

We (can) adapt quite some to customers so usually

- * design
- * implementation
- * testing
- * commissioning
- * support

is part of the package

Remote is ok, but willingness to travel will then be needed sometimes

The ad is here

<<https://careers.consafelogistics.com/jobs/3613985-software-developer-to-the-wcs-team>>

To give some insight - the site shown here - the conveyors and Autostore are controlled by our software. We tell the devices where to pickup and where to deliver - different protocols. It went live this spring.

<<https://www.youtube.com/watch?v=57oSqX19C4w>>

What is an Autostore?

<<https://www.youtube.com/watch?v=iHC9ec591II&t=43s>>

What we do? We control devices like the ones in this video

<<https://www.youtube.com/watch?v=IW3PkMMN8ns>>

Ada-Lang and Its Forum

From: Fernando Oleo / Irvise

<irvise_ml@irvise.xyz>

Subject: Ada-Lang and it's (more active than CLA) forum

Date: Sat, 11 May 2024 13:30:32 +0200

Newsgroups: comp.lang.ada

This is a simple and quick reminder (or announcement if it's the first time you get notified) regarding Ada-Lang [1] and its Forum [2].

Ada-Lang is a community maintained and supported webpage whose intent is to give a nice "landing page" to anybody wanting to learn Ada and become a hub for all Ada users. It has a few nice links to social media and chat-rooms (at the bottom), a section to read C.L.A directly on your web-browser [3], a formatted version of the ARM [4], tutorial and examples (WIP) [5] and a few other nice features.

The Forum is quite active and it has a lot of topics. You can think about it as a modern web version of C.L.A :) Though, an account is required... And I do not mean to belittle C.L.A, it is a great resource!

Everybody is more than welcomed to participate in the forums and help the website grow in quality and content. Everything is open source, so it is very easy to help around!

[1] <https://ada-lang.io/>

[2] <https://forum.ada-lang.io/>

[3] <https://usenet.ada-lang.io/comp.lang.ada/>

[4] <https://ada-lang.io/docs/arm>

[5] <https://ada-lang.io/docs/learn/why-ada>

Ada-related Tools

GCC 14.1.0 for MacOS

From: Simon Wright

<simon@pushface.org>

Subject: ANN: GCC 14.1.0 for macOS

Date: Thu, 16 May 2024 21:09:55 +0100

Newsgroups: comp.lang.ada

GCC 14.1.0 suite available for macOS:

aarch64,

<https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-14.1.0-aarch64>

x86_64,

https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-14.1.0-x86_64

From: Simon Wright

<simon@pushface.org>

Date: Thu, 23 May 2024 16:00:32 +0100

> GCC 14.1.0 suite available for macOS:

> aarch64,

> <https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-14.1.0-aarch64>

> x86_64,

> https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-14.1.0-x86_64

and an aarch64-based cross-compiler to arm-eabi, at <https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-14.1.0-aarch64-arm-eabi>

From: Orangefish

<orangefish@invalid.invalid>

Date: Fri, 17 May 2024 11:01:09 -0400

> aarch64,

> <https://github.com/simonjwright/distributing-gcc/releases/tag/gcc-14.1.0-aarch64>

Do you know how this differs from Iain Sandoe's port

(<https://github.com/iains/gcc-darwin-arm64>)?

From: Simon Wright

<simon@pushface.org>

Date: Sat, 18 May 2024 21:02:05 +0100

> Do you know how this differs from Iain Sandoe's port?

It's from Iain's repo <https://github.com/iains/gcc-14-branch>.

I _think_ that this corresponds to GCC's releases/gcc-14.1.0 tag ????

I can't find it now, but when I asked why he separated the release "branches" from the development one (gcc-darwin-arm64) Iain referenced some difficulty with Github.

From: Lawrence D'Oliveiro

<ldo@nz.invalid>

Date: Sat, 18 May 2024 21:42:34 -0000

> Iain referenced some difficulty with Github.

People need to remember that GitHub ≠ Git.

PragmAda Reusable Components

From: Pragmada Software Engineering

<pragmada@pragmada.x10hosting.com>

Subject: [Reminder] The Pragmada Reusable Components

Date: Sat, 1 Jun 2024 14:48:06 +0200

Newsgroups: comp.lang.ada

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

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

Simple Components v4.69

From: Dmitry A. Kazakov

<mailbox@dmitry-kazakov.de>

Subject: ANN: Simple Components v4.69

Date: Sat, 29 Jun 2024 09:30:47 +0200

Newsgroups: comp.lang.ada

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

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

Changes the previous version:

- The procedure Compile was added to Python bindings in order to get a handle to the compiled module;
- The function Object_Super of the bindings is an equivalent of x.super().y. It returns the implementation of y from the x's parent class.

Ada Practice

Was the Mandate Change a Poor Decision?

*From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Subject: Was the mandate change a poor decision?
Date: Thu, 4 Apr 2024 00:47:20 -0000
Newsgroups: comp.lang.ada*

<https://dl.acm.org/doi/pdf/10.1145/260096.260385>

This paper completely ignored arguments such as the mandate apparently only requiring demonstrating an expectation of being more cost effective than Ada. Replaced by a mandate that is apparently ignored as it is basically reduced to an aim of being cost effective without competition to Ada which was designed to be cost effective and so hard to beat.

History seems to have shown that he was completely wrong and in my opinion the mandate change has inflated costs to taxpayers significantly, such as in the F-35/JSF program.

Do you think that this paper had any influence on making the mandate impotent?

Am I ignorant of improvements that the mandate change has brought about?

Software Engineer Seeks Compatible Cratifier

*From: Jeffrey R. Carter
<spam.jrcarter.not@spam.acm.org.not>
Subject: Software Engineer Seeks Compatible Cratifier
Date: Tue, 18 Jun 2024 15:05:38 +0200
Newsgroups: comp.lang.ada*

<https://forum.ada-lang.io/t/2024-crate-of-the-year-awards/923/8?u=jc001>

[Contents of the above link follow. —arm]

I don't use gprbuild or Alire, but I, too, have a collection of software on GitHub that might benefit from cratification, if someone would find that a worthwhile activity.

*From: Pascal Obry <pascal@obry.net>
Date: Tue, 18 Jun 2024 15:40:01 +0200*

The first thing would be to accept adding a GPR project file into your projects. You have rejected my PR proposing just this. How one is supposed to build a set of unrelated Ada files?

*From: Jeffrey R. Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Tue, 18 Jun 2024 20:06:06 +0200*

IIUC, there has to be a project file in the crate. That does not mean there has to be a project file in my Github repository.

*From: Pascal Obry <pascal@obry.net>
Date: Tue, 18 Jun 2024 20:27:16 +0200*

But then you impose on someone to create a project file. And this would be needed by all packagers, there are not only crates around to build a component. And moreover, you let the "packager" find the correct options to be used which is certainly not a good solution.

That's exactly why I had proposed a PR with a project file. It "documents" the way the code must be compiled (at least with GNAT) and installed for everyone wanting to package your components.

*From: Simon Wright
<simon@pushface.org>
Date: Tue, 18 Jun 2024 22:12:15 +0100*

>> IIUC, there has to be a project file in the crate. That does not mean there has to be a project file in my Github repository.

The crate in the index contains (amongst other things) the name of the .gpr and a link to the source; e.g.

```
project-files = "minimal_containers.gpr"
[origin]
commit = "592661c64b8ad6fa40864e9584a8faa3a1d2b283"
url = "git+https://github.com/simonjwright/minimal_containers.git"
```

The project file is in the source.

It would be possible to 'cratify' the PragmaARCs by forking the repo and adding an alire.toml and a .gpr - which would be very simple if there was no particular need for special compilation switches (and one wouldn't need compile_all.adb). But it would feel a bit daunting without a helpful README.

*From: Luke A. Guest
<laguest@archeia.com>
Date: Wed, 19 Jun 2024 09:36:33 +0100*

> How one is supposed to build a set of unrelated Ada files?

alr init --lib --in-place

*From: Simon Wright
<simon@pushface.org>
Date: Wed, 19 Jun 2024 11:22:19 +0100*

> <https://forum.ada-lang.io/t/2024-crate-of-the-year-awards/923/8?u=jc001>

I'm prepared to do this, if we can agree.

I would fork your repo, and make changes/distribute from there.

I propose an alire.toml, on these lines:

```
name = "pragmarc"
### needs to be in lower case
description = "Utility library"
```

```
version = "4.0.0"
### you have 4 releases on Github, the
### current code is the 4th, I believe.
### The release needs to be in this
### form; I did think about
### e.g. 2024.03.23, that might work
```

```
authors = ["Jeffrey R. Carter"]
maintainers = ["Simon Wright
<simon@pushface.org>"]
maintainers-logins = ["simonjwright"]
licenses = "BSD-3-Clause"
```

```
website = https://github.com/jrcarter/Pragmarc
### is there a better one? if not, this
### should probably be to my fork
```

```
tags = ["utility", "library"]
### what should this list be
### extended to?
```

```
[build-switches]
"*.Style_Checks = "no"
### alire's style checks disagree
### violently with your usage!
"*.Ada_Version = "Ada12"
### this is what you specify in the code
```

(email: simon@pushface.org)

*From: Alastair Hogge <agh@riseup.net>
Date: Sun, 23 Jun 2024 08:10:04 -0000*

> That's exactly why I had proposed a PR with a project file.

+1 for a .gpr project file to enable package maintainers; it makes it easier to integrate Ada systems with the host system.

*From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sun, 23 Jun 2024 15:21:51 +0200*

> +1 for a .gpr project file to enable package maintainers; it makes it easier to integrate Ada systems with the host system.

Yes, but the system must generate a parent gpr file with the target settings. Fedora's GNAT has such a thing in a very primitive form. It must define the target OS description, default switches, 32 vs 64 bit, architecture, availability of pragma Atomic for 64-bit scalars, some standard libraries for the linker etc. Then the user gpr will be able to refer to these in the main gpr.

OT, the reason why I do not use Alire is because it has no upload function. I cannot manually submit my projects each time I ship a new version of some of them. It should support automated uploads.

*From: G.B.
<bauhaus@notmyhomepage.invalid>
Date: Tue, 25 Jun 2024 22:47:49 +0200*

> The first thing would be to accept adding a GPR project file into your projects.

Might a good README actually be better? And also stable?

The number of programming languages in production used to be really large. So large, 400+, that a reduction project was

given green light. To get the number up again, it seems that the market is having every programming language multiplied by at least two build tools' description language.

As before, mostly single vendors are providing the definitions of a respective formalism, versions, obsolescence, life cycle policies, all included.

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.

2024

- | | |
|------------|---|
| July 01-05 | 24th IEEE International Conference on Software Quality, Reliability and Security (QRS'2024), Cambridge, UK. 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. |
| July 08-12 | Software Technologies: Applications and Foundations (STAF'2024), Twente, the Netherlands. Topics include: practical and foundational advances in software technology. |
| July 08-12 | 20th European Conference on Modelling Foundations and Applications (ECMFA'2024), Twente, the Netherlands. Co-located with STAF'2024. Topics include: all aspects of model-based engineering (MBE); foundations of MBE, including model transformations, domain-specific languages, verification and validation approaches, ...; application of MBE methods, tools, and techniques to specific domains, e.g., automotive, aerospace, cyber-physical systems, robotics, Artificial Intelligence or IoT; educational aspects of MBE; tools and initiatives for the successful adoption of MBE in industry; etc. |
| July 09-12 | 36th Euromicro Conference on Real-Time Systems (ECRTS'2024), Lille, France. Topics include: all aspects of timing requirements in computer systems; elements of time-sensitive software systems, such as operating systems, hypervisors, middlewares and frameworks, programming languages and compilers, runtime environments, ...; real-time applications topics, such as modeling, design, simulation, testing, debugging, and evaluation in domains such as automotive, avionics, control systems, industrial automation, robotics, space, railways telecommunications, multimedia, ...; foundational scheduling and predictability questions, such as schedulability analysis, synchronization protocols, ...; static and dynamic techniques for resource demand estimation, such as classic worst-case execution time (WCET) analysis, ...; formal methods for the verification and validation of real-time systems; the interplay of timing predictability and other non-functional qualities, such as reliability, security, quality of control, testability, scalability, ...; etc. |
| July 15-19 | 32nd ACM International Conference on the Foundations of Software Engineering (FSE'2024), Porto de Galinhas, Brazil. Topics include: debugging and fault localization; dependability, safety, and reliability; embedded software, safety-critical systems, and cyber-physical systems; model checking; model-driven engineering; parallel, distributed, and concurrent systems; program analysis; programming languages; software architectures; software engineering education; software evolution; software security; software testing; software traceability; symbolic execution; tools and environments; etc. |
| July 22-27 | 36th International Conference on Computer-Aided Verification (CAV'2024), Montreal, Canada. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems, algorithms and tools for verifying models and implementations, specifications and correctness criteria for programs and systems, deductive verification using proof assistants, program analysis and software verification, hybrid systems and embedded systems verification, formal methods for cyber-physical systems, verification methods for parallel and concurrent systems, testing and run-time analysis based on verification technology, applications and case studies in verification and synthesis, verification in industrial practice, formal models and methods for security, etc. |

- Jul 29 – Aug 01 **36th International Conference on Software Engineering Education and Training (CSEET'2024)**, Würzburg, Germany. Topics include: novel ideas, methods, and techniques for software engineering education; education experience & industrial training reports; teaching formal methods, teaching "real world" SE practices, software quality assurance education, motivating students and trainees, open source in education, cooperation between industry and academia, training models in industry, continuous integration and continuous delivery education, cyber-physical system or Internet of Things education, etc. Deadline for early registration: July 15, 2024.
- Jul 30 – Aug 2 **19th International Conference on Availability, Reliability and Security (ARES'2024)**, Vienna, Austria. Topics include: various aspects of dependability; crucial linkage between availability, reliability and security; availability, safety, confidentiality, integrity, maintainability and security in different fields of applications; dependability in emerging areas; compliance, certification and legal aspects related to security and privacy; dependability, and resilience; software security; static and dynamic code analysis for security and privacy; security and privacy for IoT, cyber-physical systems, and critical infrastructures; security requirements and secure design of applications; trusted computing; etc.
- ☺ August 26-30 **30th International European Conference on Parallel and Distributed Computing (Euro-Par'2024)**, Madrid, Spain. Topics include: all aspects of parallel and distributed processing, ranging from theory to practice, from small to the largest parallel and distributed systems and infrastructures, from fundamental computational problems to applications, from architecture, compiler, language and interface design and implementation, to tools, support infrastructures, and application performance aspects.
- August 28-30 **50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA'2024)**, Paris, France. Topics include: information technology for software-intensive systems; tracks on Cyber-Physical Systems (CPS), Emerging Computing Technologies (ECT), Model-Driven Engineering and Modeling Languages (MDEML), Software Process and Product Improvement (SPPI), Practical Aspects of Software Engineering (KKIO), etc.
- September 04-06 **27th Forum on specification & Design Languages (FDL'2024)**, Stockholm, Sweden. Topics include: results, experiences, advances and new trends related to languages, tools, and techniques used for developing software and hardware systems; targeted systems encompass cyber-physical systems, distributed systems, real-time systems, embedded systems, mechatronics, IoT, and reactive systems; based on the four non-limiting scientific areas of languages, semantics, verification and analysis, simulation. Deadline for submissions: July 12, 2024 (PhD/WiP papers).
- September 09-10 **20th International Conference on Formal Aspects of Component Software (FACS'2024)**, Milan, Italy. Co-located with FM'2024. 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, including verification techniques, ...; formal aspects of concrete software-intensive systems, including real-time/safety-critical systems, hybrid and cyber physical systems, ...; tools supporting formal methods for components and services; case studies and experience reports over the above topics; etc.
- September 09-11 **3rd Summer School on Security Testing and Verification (ST&V'2024)**, Brussels, Belgium. Topics include: static and dynamic security testing; software verification; security by design; etc. Deadline for early registration: July 21, 2024.
- ☺ Sep 09-11 **29th International Conference on Formal Methods for Industrial Critical Systems (FMICS'2024)**, Milan, Italy. Co-located with FM'2024. Topics include: case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; methods, techniques, and tools to support automated analysis, certification, debugging, learning, optimization, and transformation of complex, distributed, real-time, embedded, mobile, and autonomous systems; verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability (e.g., scalability and usability issues, tool qualification, and certification); application of formal methods in standardization and industrial forums; etc.
- September 09-13 **26th International Symposium on Formal Methods (FM'2024)**, Milan, Italy. Topics include: development and application of formal methods in a wide range of domains including trustworthy AI, software, computer-based systems, systems-of-systems, cyber-physical systems, security, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, healthcare and biology; techniques, tools, and experiences in interdisciplinary settings; experiences of applying formal

methods in industrial settings; design and validation of formal method tools; etc. Deadline for early registration: July 30, 2024.

Sep 09-10 **18th International Conference on Tests And Proofs (TAP'2024)**. Topics include: many aspects of verification technology, including foundational work, tool development, and empirical research; the combination of static techniques such as proving and dynamic techniques such as testing; verification and analysis techniques combining proofs and tests; static analysis of programs with the aid of dynamic techniques; deductive techniques supporting the automated generation of test vectors and oracles, and supporting (novel) definitions of coverage criteria; specification inference by deductive or dynamic methods; testing and runtime analysis of formal specifications; verification of verification tools and environments; applications of test and proof techniques in new domains; combined approaches of test and proof in the context of formal certifications; case studies, tool and framework descriptions, and experience reports; etc. Deadline for submissions: July 7, 2024 (artifacts).

September 09-13 **35th International Conference on Concurrency Theory (CONCUR'2024)**, Calgary, Canada. Topics include: verification and analysis techniques for concurrent systems such as abstract interpretation, model checking, race detection, run-time verification, static analysis, testing, theorem proving, type systems, security analysis, ...; distributed algorithms and data structures: design, analysis, complexity, correctness, fault tolerance, reliability, availability, consistency, ...; theoretical foundations, tools, and empirical evaluations of architectures, execution environments, and software development for concurrent systems such as multiprocessor and multi-core architectures, compilers and tools for concurrent programming, programming models such as component-based, object-oriented, ...; etc.

September 09-13 **22nd International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'2024)**, Calgary, Canada. Topics include: fundamental and practical aspects of systems with quantitative nature; modelling, design and analysis of computational systems; models and metrics for the correctness, performance, reliability, safety, and security of systems; techniques, algorithms, data structures for analysis, evaluation, and verification of the models mentioned above, e.g., for model checking, testing, constraint solving, scheduling, optimization, and worst-case execution time analysis; novel software tools to support practical application of research results in all of the above areas; etc. Deadline for submissions: July 16, 2024 (Work-in-Progress presentations).

September 11-13 **17th International Conference on the Quality of Information and Communications Technology (QUATIC'2024)**, Pisa, Italy. Topics include: all quality aspects in ICT systems engineering and management.

☺ Sep 16-20 **38th European Conference on Object-Oriented Programming (ECOOP'2024)**, Vienna, Austria. Topics include: programming languages, software development, systems and applications. Deadline for submissions: June 30 - July 8, 2024 (workshop papers).

☺ Sep 17-20 **43rd International Conference on Computer Safety, Reliability and Security (SafeComp'2024)**, Florence, Italy. Topics include: all aspects related to the development, assessment, operation, and maintenance of safety-related and safety-critical computer systems; safety guidelines and standards; safety/security co-engineering and tradeoffs; safety and security qualification, quantification, assurance and certification; model-based analysis, design, and assessment; formal methods for verification, validation, and fault tolerance; testing, verification, and validation methodologies and tools; etc. Domains of application include: railways, automotive, space, avionics & process industries; highly automated and autonomous systems; telecommunication and networks; critical infrastructures; medical devices and healthcare; surveillance, defense, emergency & rescue; logistics, industrial automation, off-shore technology; education & training; etc. Deadline for submissions: July 1, 2024 (position papers).

Sep 29 – Oct 03 **19th International Conference on Software Engineering Advances (ICSEA'2024)**, Venice, Italy. 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.

- Sep 29 – Oct 04 **Embedded Systems Week 2024** (ESWEEK'2024), Raleigh, North Carolina, USA. Includes CASES'2024 (International Conference on Compilers, Architectures, and Synthesis for Embedded Systems), CODES+ISSS'2024 (International Conference on Hardware/Software Codesign and System Synthesis), EMSOFT'2024 (International Conference on Embedded Software). Deadline for submissions: July 8 - August 15, 2024 (workshop papers), July 15, 2024 (competitions), July 31, 2024 (education classes).
- October 03 **Workshop on Time-Centric Reactive Software** (TCRS'2024). Topics include: automotive systems, compiler construction, cyber-physical systems, distributed systems, embedded systems, formal verification, programming languages, model-based design, modeling languages, middleware, real-time systems, etc. Deadline for submissions: July 8, 2024.
- October 03-04 **22nd ACM/IEEE International Symposium on Formal Methods and Models for System Design** (MEMOCODE'2024). Topics include: formal methods in system design that address the foundations, engineering methods, tools, or experimental case studies; modeling languages, methods, and tools (programming languages and models, software and system modeling languages, architecture and high-level hardware description languages, ...), formal methods and tools (correct-by-construction methods; contract-based design and verification; static, dynamic, and type theoretic analysis; verification; validation; test generation; ...), models and methods for developing critical systems (security-critical and safety-critical systems, cyber-physical systems, autonomous systems, ...), formal methods/models in practice, etc.
- October 01-03 **2024 International Conference on Software Engineering Research & Development** (SERD'2024), Oklahoma City, Oklahoma, USA & Online. Topics include: general and social aspects of software engineering (SE); software design, testing, evolution, and maintenance; formal methods and theoretical foundations; programming languages (PLs), systems, and environments; object-oriented (OO) design and analysis; emerging SE technologies and dependability; distribution, componentization, and collaboration; concurrent, parallel and distributed systems; etc. Deadline for submissions: July 15, 2024.
- October 07-08 **24th IEEE International Working Conference on Source Code Analysis and Manipulation** (SCAM'2024), Flagstaff, Arizona, USA. Topics include: abstract interpretation, bad smell detection, clone detection, program comprehension, program slicing, program transformation and refactoring, security vulnerability analysis, source level metrics, source level optimization, source-level testing and verification, static and dynamic analysis, etc. Deadline for submissions: July 1, 2024 (abstracts Engineering and New Ideas and Emerging Results (NIER) tracks), July 5, 2024 (papers Engineering and New Ideas and Emerging Results (NIER) tracks).
- ☺ October 13-16 **33rd International Conference on Parallel Architectures and Compilation Techniques** (PACT'2024), Long Beach, California, USA. Topics include: parallel architectures; compilers and tools for parallel architectures; 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; distributed computing architectures and systems; etc.
- October 15-18 **24th International Conference on Runtime Verification** (RV'2024), Istanbul, Türkiye. Topics include: monitoring and analysis of runtime behavior of software, hardware, and cyber-physical systems; program instrumentation; combination of static and dynamic analysis; monitoring techniques for concurrent and distributed systems; fault localization, containment, resilience, recovery and repair; etc.
- October 15-18 **24th International Conference on Formal Methods in Computer-Aided Design** (FMCAD'2024), Prague, Czech Republic. Topics include: methods, technologies, theoretical results, and tools for reasoning formally about computing systems; formal aspects of computer-aided system design including verification, specification, synthesis, and testing; etc.
- October 16-18 **17th International Conference on Verification and Evaluation of Computer and Communication Systems** (VECoS'2024), Djerba, Tunisia. Topics include: analysis of computer and communication systems, where functional and extra-functional properties are inter-related; cross-fertilization between various formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems. Deadline for submissions: July 5, 2024.

- October 20-22 31st **Static Analysis Symposium** (SAS'2024), Pasadena, USA. Co-located with SPLASH'2024. Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.
- ☉ October 20-25 **ACM Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2024), Pasadena, California, USA. Deadline for submissions: July 7, 2024 (workshop papers), July 8, 2024 (student research competition), July 30, 2024 (volunteer applications).
- October 20-21 17th **ACM SIGPLAN International Conference on Software Language Engineering** (SLE'2024). Topics include: software language engineering in general rather than engineering a specific software language; software language design and implementation; validation of software language tools and implementations (verification and formal methods, testing techniques, simulation techniques); software language maintenance (software language reuse; language evolution; language families and variability, language and software product lines); software language integration and composition domain-specific approaches for any aspects of SLE; (analysis, design, implementation, validation, maintenance); empirical studies and experience reports of tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.
- ☉ 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.
- October 21-24 21st **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2024), Kyoto, Japan. 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; verification in industrial practice; applications and case studies; automated tool support; etc.
- October 22 **High Integrity Software Conference** (HISC'2024), Newport, South Wales, UK. Topics include: advanced software development for high-integrity and high-assurance systems, including programming languages, verifiable code generation; verification and testing of high-integrity systems; assurance of high-integrity systems; infrastructure and ecosystem for high-integrity software; etc.
- October 22-24 22nd **Asian Symposium on Programming Languages and Systems** (APLAS'2024), Kyoto, Japan. Topics include: all areas of programming languages and systems; programming paradigms and styles; methods and tools to specify and reason about programs and languages; programming language foundations; methods and tools for implementation; concurrency and distribution; applications, case studies and emerging topics.
- Oct 27 – Nov 01 39th **IEEE/ACM International Conference on Automated Software Engineering** (ASE'2024), Sacramento, California, USA. Topics include: foundations, techniques, and tools for automating analysis, design, implementation, testing, and maintenance of large software systems. Deadline for submissions: July 2, 2024 (journal-first papers), July 12, 2024 (industry showcase).
- October 28-31 35th **IEEE International Symposium on Software Reliability Engineering** (ISSRE'2024), Tsukuba, Japan. 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); software testing and formal methods; software fault localization, debugging, root-cause analysis; reliability of AI-based systems; reliability of model-based and auto-generated software; reliability of open-source software; normative/regulatory/ethical spaces about software reliability; societal aspects of software reliability; etc. Deadline for submissions: July 23, 2024 (Doctoral Symposium), July 28, 2024 (workshop papers).
- November 04-08 22nd **International Conference on Software Engineering and Formal Methods** (SEFM'2024), Aveiro, Portugal. 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 (testing and runtime 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; etc.

- ☺ Nov 07-08 **32nd International Conference on Real-Time Networks and Systems (RTNS'2024)**, Porto, Portugal. Deadline for submissions: August 14, 2024 (abstracts 3rd round), August 16, 2024 (papers 3rd round).
- November 13-15 **19th International Conference on integrated Formal Methods (iFM'2024)**, Manchester, UK. 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.
- November 13-15 **29th IEEE Pacific Rim International Conference on Dependable Computing (PRDC'2024)**, Osaka, Japan. 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. Deadline for submissions: July 24, 2024 (abstracts), July 31, 2024 (papers).
- December 03-06 **31st Asia-Pacific Software Engineering Conference (APSEC'2024)**, Chongqing, China. Topics include: requirements and design; component-based software engineering; software architecture, modeling and design; middleware, frameworks, and APIs; software product-line engineering; testing and analysis; testing, verification, and validation; program analysis; program repairs; formal aspects of software engineering; formal methods; model-driven and domain-specific engineering; software comprehension and traceability; dependability, safety, and reliability; software maintenance and evolution; refactoring; reverse engineering; software reuse; debugging and fault localization; software repository mining; etc. Deadline for submissions: July 6, 2024 (abstracts), July 13, 2024 (papers).
- ☺ December 10-13 **45th IEEE Real-Time Systems Symposium (RTSS'2024)**, York, UK. Topics include: addressing some form of real-time requirements/constraints, such as deadlines, response time, or delay/latency. Deadline for submissions: July 1, 2024 (Hot Topics Day proposals), September 30, 2024 (TCRTS award nominations).
- December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

2025

- ☺ January 19-25 **52nd ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'2025)**, Denver, Colorado, USA. Topics include: all aspects of programming languages and programming systems, both theoretical and practical; 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: July 11, 2024 (papers), July 26, 2024 (workshops, co-located events).
- January 20-21 **International Conference on Certified Programs and Proofs (CPP'2025)**. Topics include: research areas related to formal certification of programs and proofs; new languages and tools for certified programming; program analysis, program verification, and program synthesis; program logics, type systems, and semantics for certified code; teaching mathematics and computer science with proof assistants; etc.
- March 01-05 **ACM/IEEE International Symposium on Code Generation and Optimization (CGO'2025)**, Las Vegas, USA. Deadlines for paper submissions: September 12, 2024 (2nd round).
- Mar 30 – Apr 03 **20th European Conference on Computer Systems (EuroSys'2025)**, Rotterdam, the Netherlands. Topics include: all areas of computer systems research, such as distributed systems, language support and runtime systems, systems security and privacy, dependable systems, analysis, testing and verification of systems,

parallelism, concurrency, and multicore systems, real-time, embedded, and cyber-physical systems, etc. Fall deadline for submissions: October 15, 2024 (abstracts), October 22, 2024 (submissions).

- Apr 26 – May 04 47th **International Conference on Software Engineering** (ICSE'2025), Ottawa, Ontario, Canada. Topics include: the full spectrum of Software Engineering (SE), trustworthy AI for SE; AI-assisted software design and model driven engineering; mining software repositories; software metrics (and measurements); software design methodologies, principles, and strategies; architecture quality attributes, such as security, privacy, performance, reliability; modularity and reusability; dependency and complexity analysis; patterns and anti-patterns; technical debt in design and architecture; formal methods and model checking; reliability, availability, and safety; resilience and antifragility; design for dependability and security; vulnerability detection to enhance software security; dependability and security for embedded and cyber-physical systems; evolution and maintenance; API design and evolution; software reuse; refactoring and program differencing; program comprehension; reverse engineering; environments and software development tools; human and social aspects (focusing on programming languages, environments, and tools supporting individuals, teams, communities, and companies; focusing on software development processes; ...); modeling and model-driven engineering; variability and product lines; modeling languages, techniques, and tools; empirical studies on the application of model-based engineering; software testing; automated test generation techniques such as fuzzing, search-based approaches, and symbolic execution; testing and analysis of non-functional properties; program analysis; debugging and fault localization; runtime analysis and/or error recovery; etc. Deadline for submissions: July 24, 2024 (showdown PC self-nomination), July 26, 2024 (2nd cycle abstracts), August 2, 2024 (2nd cycle submissions), November 11, 2024 (workshop papers).
- May 03-08 28th ETAPS **International Joint Conferences on Theory and Practice of Software** (ETAPS'2025), Hamilton, Canada. Deadline for submissions: July 18, 2024 (satellite events), October 10, 2024 (TACAS, FoSSaCS, FASE, ESOP)
- ♦ June 10-13 29th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC'2025), Paris, France. Organized by Ada-Europe and Ada-France. #AEiC2025 #AdaEurope #AdaProgramming
- December 10 Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!



Looking forward to AEiC 2025

Paris, France

The 29th Ada-Europe International Conference on Reliable Software Technologies (**AEiC 2025**) will take place on **June 10-13**, 2025, in **Paris, France**.

It will be hosted by the prestigious “École des Mines de Paris”, located next to the *Quartier Latin*.

Did you enjoy the superb views of Paris during the Olympics? Now is the time to visit by yourself and enjoy famous monuments, museums, or just walk along the streets of the beautiful city of light!

And of course, attend the conference, which will offer all the usual features that you liked on previous occasions: keynotes, presentations, posters, vendor booths, and interactions with other participants.

We'll offer the logistics for a great event, with enjoyable social, cultural, and touristic opportunities. Start now preparing submissions for an interesting scientific and industrial program!

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/>

A Framework for Improving Portability and Ensuring Correctness of Operating System Kernels

Vignesh Manjunath

Pro2Future GmbH, Graz, Austria; email: vignesh.manjunath@pro2future.at

Marcel Baunach

Graz University of Technology, Graz, Austria; email: baunach@tugraz.at

Abstract

Traditional embedded Real-Time Operating Systems (RTOS) or Basic Software (BSW) implementations typically require manual porting to new hardware platforms. However, this approach can be time-consuming and error-prone, especially given the frequent introduction of new or upgraded hardware architectures. In addition, traditional testing methods may not fully capture the complexity and nuances of the system, making it difficult to ensure correctness and dependability. To address these challenges, we propose a comprehensive methodology that integrates formal methods, a WCET tool, and a code generation technique. We use formal methods to create models and verify their correctness against functional and non-functional specifications or properties such as safety, liveness, and timing. We use the WCET tool as a microarchitecture analyzer to analyze the low-level binary code. The intermediate results of the tool are used to verify the correctness of the software implementation against the runtime effects of the hardware, such as data/memory race conditions. Finally, the code is generated from the formal models. Our proposed framework simplifies the maintenance of RTOS or BSW implementations while ensuring their correctness and partially automating portability to new hardware architectures.

Keywords: RTOS, formal methods, dependability

1 Introduction

Conventional implementations of embedded RTOS and BSW modules, such as device drivers, typically require manual porting to new hardware (HW) architectures. This is because the low-level implementations (e.g., system initialization, context switch) of these modules are hardware-dependent and do not compile for different targets. In contrast, high-level implementations (e.g., scheduler, task management) that are hardware-independent can be reused more easily. The manual porting process can be tedious, time-consuming, and error-prone, especially given the frequent introduction of new or upgraded hardware. Developers must repeat the porting process for each new hardware platform they encounter.

In addition, traditional software testing methods may not fully capture the complexity and nuances of the system, which can lead to issues of correctness and dependability. Testing might only cover specific scenarios or functionalities, potentially leaving out corner cases or other system behaviors that may impact the overall correctness of the system. This could be a major challenge for critical systems, e.g., automotive and avionics, where high levels of dependability are required.

While approaches such as [1, 2] attempt to model and verify OS kernels using formal methods, the low-level implementations still need to be ported manually. The process even requires additional effort to implement a formal specification of the low-level code, eventually increasing the overall time required to port the RTOS. Some approaches such as [3, 4] use formal methods to model, verify, and generate low-level code. They provide the basis for automatic portability and correctness of RTOS implementations. However, they do not have a hardware model to consider other internal hardware effects such as pipeline and cache behavior, data and instruction synchronization dependencies, and memory race conditions during verification, which limits their applicability.

Our framework addresses the portability and correctness challenges mentioned earlier by integrating three main concepts: First, we use UPPAAL [5] and Event-B [6] as formal methods to model an RTOS, and then verify its correctness with respect to functional specifications and non-functional properties, i.e., safety, liveness, and timing. Second, we use the WCET analyzer OTAWA [7] as a microarchitecture analyzer for the low-level implementations of the RTOS. The intermediate results of OTAWA are used to verify correctness against runtime effects of the hardware architecture, i.e., the data/memory race conditions and synchronization dependencies. Third, we use a code generation technique based on the LLVM backend [8] to generate assembly code from the models.

The primary focus of this paper is on the development of the proposed framework. Section 2 provides some design decisions on the key components and their integration within our framework. Section 3 describes the underlying methodology, and Section 4 presents the preliminary results of a part of our framework. Section 5 discusses the expected benefits and limitations. Section 6 compares related work,

and Section 7 summarizes the paper with an outlook to future work.

2 Design Decisions

This section presents a brief overview of the main components of our framework and discusses the design decisions that lead to their integration into the framework.

UPPAAL [5] is a formal modeling and verification tool based on timed automata. Since RTOS must meet strict timing constraints, it is essential to verify their correctness using timing verification. Therefore, UPPAAL is an appropriate choice for our framework to model and verify the correctness by checking both functional, temporal (safety and liveness), and timing properties. In addition, UPPAAL is a symbolic model-checking tool that facilitates a seamless transition from code-based development to formal model-based development.

Event-B [6] is a formal method based on set theory and refinements. First, an abstract model of the system is created and then incrementally refined to a concrete model. The refinement approach makes Event-B suitable for formally modeling the low-level implementations of RTOS. Similar to [3], the idea is to include hardware details (e.g., registers) only in the final refinement so that the previous refinements can be reused for different hardware architectures. For example, the existing work reuses 10 out of 14 model refinements across MSP430 and RISC-V architectures. This way, we reduce the manual porting effort and partially automate portability. Our framework uses Event-B to verify safety and liveness properties.

LLVM [9] is a compiler infrastructure that consists of a frontend for high-level languages such as C and C++, and generates an Intermediate Representation (IR). Its backend uses the LLVM IR to generate assembly code for a specific target architecture. Due to its modular structure, the LLVM backend can be used as a standalone tool in our framework to generate assembly code for a target architecture.

OTAWA [7] is a static analysis framework to estimate WCET. It first generates a Control Flow Graph (CFG) of a program from the ELF executable. The CFG consists of a set of connected basic blocks based on the program flow. The basic blocks contain a sequence of instructions with one entry and exit. OTAWA then performs a timing analysis for each basic block and its predecessor blocks to calculate the possible execution times for each execution path while considering hardware effects (e.g., cache hit/miss, pipeline behavior). By extracting the timing analysis results into eXecution Graphs (XGraphs) [10], we obtain pipeline-level (e.g., fetch, decode, and execute) timing information in CPU cycles while covering all possible execution paths and hardware behavioral aspects (pipeline, memory and cache access). Although we can use cycle-accurate simulators such as gem5 [11] to analyze the programs, we cannot simultaneously analyze different execution paths and consider all hardware effects. Therefore, in our framework, we use OTAWA as a microarchitecture analyzer to analyze the low-level implementations and extract the intermediate (program and timing analysis) results of the WCET analysis. Furthermore, OTAWA supports major

embedded hardware architectures such as ARM7TDMI, TriCore, and RISC-V.

3 Framework and Methodology

Figure 1 illustrates the proposed methodology which integrates formal methods, a WCET analyzer, and a code generation technique. The process involves a sequence of steps, referred to by **No.** in the figure. To modularize our approach, we begin by manually splitting the RTOS specifications into two categories: hardware dependent and independent. While the hardware-independent specifications (e.g., scheduler, task management) are reusable across different hardware platforms, the hardware-dependent (e.g., context switch, system initialization) ones require manual porting. In step **1**, we use the formal method UPPAAL to model the hardware-independent specifications. Next, in steps **2** through **6b**, we model low-level specifications and verify them for correctness against functional specifications and runtime hardware effects. In steps **7** and **8**, we use the results of the verification and hardware analysis to formally verify functional and non-functional correctness. In step **9**, we generate code, which is then subject to post-generation verification in step **10**. Finally, if the verification is successful, we deploy the final code in step **11**. Let's take a closer look at each of these steps and their specific tasks:

- 1** We use UPPAAL to model the hardware-independent specifications of an RTOS, such as the scheduler or resource manager. We also model the syscalls, since these are typically implemented in a high-level programming language. By formalizing the related specifications, we aim to create a set of models that can be used for different hardware platforms, making them automatically portable.
- 2** We use the refinement approach in Event-B to model and functionally verify the hardware-dependent RTOS specifications, such as context switch or system initialization. The hardware details are included only in the final refinement so that the previous refinements can be reused for different hardware platforms. We then translate the Event-B models to LLVM IR and use the LLVM backend to generate the corresponding assembly code. This approach builds upon the works presented in [12] and [13].
- 3** After generating the low-level code in the previous step, we compile it into an ELF executable for analysis by OTAWA. We ensure the correctness of the low-level code w.r.t. the runtime effects of the hardware architecture, we analyze the ELF instead of the low-level code. This is because we need to consider assembly-level optimizations that may affect the behavior of the code.
- 4** To analyze the input ELF with OTAWA, we need to provide additional information, such as hardware architecture details (ISA, memory, and pipeline). OTAWA then analyzes the input ELF, and we extract the intermediate results of the analysis, such as CFGs and XGraphs.
- 5** We use the CFGs and XGraphs to verify low-level code for correctness against runtime hardware effects, i.e., data hazards, memory race conditions, and synchronization

4 Preliminary Results

As a first step in realizing the proposed methodology, we used our framework for verifying low-level RTOS code against runtime effects of the AURIX TriCore architecture (i.e., steps 3 to 6a in Figure 1). In our implementation, we considered the following runtime effects: (1) data and instruction synchronization dependencies, (2) data hazards (read-after-write, write-after-read, and write-after-write), and (3) memory race conditions.

To evaluate our implementation, we analyzed and verified the context switch assembly code of FreeRTOS [14] on the AURIX TC375 architecture [15]. As a result, we identified (1) two constraint check fails with respect to data synchronization dependency, (2) nine potential memory race conditions that need further investigation, and (3) 68 data hazards that can be resolved at compile time to save execution time. Additionally, we extracted the program flow information and computed the execution times of instructions and basic blocks. For reference, we uploaded the results of our analysis at <https://dx.doi.org/10.6084/m9.figshare.25288825>.

5 Benefits and Limitations

Our proposed methodology for developing and maintaining RTOS or BSW implementations is still under construction but already shows some benefits and limitations. It is important to understand these aspects to improve or extend the software, evaluate the suitability of our methodology for specific use cases, and determine how to use its strengths effectively while overcoming its limitations.

Expected benefits:

- Improved software correctness and dependability since we use formal methods and consider hardware effects for verification.
- Improved accuracy of timing verification as we analyze all possible execution scenarios with a WCET tool.
- Improved software portability and maintenance by using the refinement approach in Event-B with a code generation technique.

Current limitations:

- High initial cost and effort to train developers in formal methods and implement formal models of the software.
- Overhead due to manual effort for partitioning RTOS specifications and formalizing system requirements into timing constraints or UPPAAL verification formulas.

6 Related Work

There are a variety of approaches that focus on formal modeling and verification of RTOS. In [3], an Event-B-based approach for formal modeling, verification, and generation of low-level RTOS code is proposed. However, the internal hardware effects such as pipeline and cache behavior, synchronization dependencies, and memory race conditions were not considered during the verification.

Similar to our methodology, the work [4] also proposes a comprehensive methodology that integrates formal methods

with other tools to verify their OS. However, they exclude hardware architecture information such as memory, cache, and pipeline and only verify correctness at the ISA level.

In [2], the seL4 kernel is proposed as a formally verified microkernel to ensure correctness, security, and reliability. The hardware-independent specifications of the kernel are verified using Isabelle/HOL [16]. While the high-level implementations are done in C, their formal specifications developed and verified in parallel. This approach requires a high implementation and maintenance effort and may lead to difficulties in maintaining consistency between the formal specifications and the C code. In addition, low-level verification of seL4 is performed using simulators or emulators, which leaves room for errors similar to traditional testing methods.

The architecture proposed in [1], called CertiKOS, aims to develop certified concurrent OS kernels using a Coq [17] proof assistant for formal verification. Their approach follows a refinement strategy, starting from an abstract model to a concrete C implementation for hardware-independent specifications. However, the hardware-dependent specifications, such as context switching, are implemented manually in assembly language and linked to the C implementation at assembly level. Unlike our approach, this requires an intensive manual porting effort when targeting new hardware platforms.

As far as we know, other works focus primarily on achieving correctness rather than portability or maintainability, as we do in our framework. Moreover, low-level verification in these works often depends on testing methods that involve simulators or emulators, which can easily miss errors. It appears that the research community needs to further investigate the portability and maintainability of RTOS.

7 Conclusion and Future Work

In this work, we proposed a novel framework that integrates formal methods, a WCET tool, and a code generation technique to verify and automatically port low-level code. We minimized the manual porting effort by keeping most of the low-level models free of hardware-specific information. In addition, we used code generation to transform the models into code and partially automate the porting process for new hardware architectures. The use of formal methods in conjunction with the WCET tool supports rigorous verification of RTOS or BSW implementations against functional specifications and non-functional properties such as safety, liveness, and timing while taking hardware aspects into account. In this way, our framework improves portability while ensuring the correctness of RTOS kernels.

As future work we plan to improve and implement missing parts of our framework, and empirically evaluate it using existing RTOS implementations. We also intend to automate the formalization of system requirements into verification formulas. This will enable or improve traceability and consistency between requirements and the formal specification.

Acknowledgement

This work has been supported by the FFG under contract No. 881844: "Pro2Future" (Products and Production Systems of the Future), Graz University of Technology (TU Graz), and Elektrobit Automotive GmbH in the joint research project CompEAS.

References

- [1] R. Gu, Z. Shao, H. Chen, X. N. Wu, J. Kim, V. Sjöberg, and D. Costanzo, "Certikos: An extensible architecture for building certified concurrent os kernels.," in *OSDI*, vol. 16, pp. 653–669, 2016.
- [2] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, *et al.*, "sel4: Formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 207–220, 2009.
- [3] R. M. Gomes, B. Aichernig, and M. Baunach, "A Formal Modeling Approach for Portable Low-Level OS Functionality," in *Software Engineering and Formal Methods* (F. de Boer and A. Cerone, eds.), (Cham), pp. 155–174, Springer International Publishing, 2020. doi: 10.1007/978-3-030-58768-0_9.
- [4] J. Shi, J. He, H. Zhu, H. Fang, Y. Huang, and X. Zhang, "ORIENTAIS: Formal verified OSEK/VDX real-time operating system," in *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pp. 293–301, IEEE, 2012. doi: 10.1109/ICECCS20050.2012.6299224.
- [5] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," *Formal Methods for the Design of Real-Time Systems: International School on Formal Methods for the Design of Computer, Communication, and Software Systems, Bertinora, Italy, September 13-18, 2004, Revised Lectures*, pp. 200–236, 2004.
- [6] T. S. Hoang, "An introduction to the event-b modelling method," *Industrial Deployment of System Engineering Methods*, pp. 211–236, 2013.
- [7] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "OTAWA: An Open Toolbox for Adaptive WCET Analysis," in *IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, pp. 35–46, Springer, 2010. doi: 10.1007/978-3-642-16256-5_6.
- [8] C. Chung-Shu, "Tutorial: Creating an llvm backend for the cpu0 architecture," 2021.
- [9] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *International symposium on code generation and optimization, 2004. CGO 2004.*, pp. 75–86, IEEE, 2004.
- [10] X. Li, A. Roychoudhury, and T. Mitra, "Modeling out-of-order processors for software timing analysis," in *25th IEEE International Real-Time Systems Symposium*, pp. 92–103, IEEE, 2004. doi: 10.1109/REAL.2004.33.
- [11] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj, *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.
- [12] R. Martins Gomes, B. Aichernig, and M. Baunach, "A framework for embedded software portability and verification: from formal models to low-level code," *Software and Systems Modeling*, pp. 1–27, 2024. doi: 10.1007/s10270-023-01144-y.
- [13] R. M. Gomes and M. Baunach, "Code generation from formal models for automatic rtos portability," in *2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 271–272, IEEE, 2019.
- [14] "FreeRTOS." <https://freertos.org/> (accessed Dec. 10, 2023).
- [15] Infineon Technologies AG, *AURIX TC3xx Architecture Volume 1, VI.2.2*, 01 2020. <https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/aurix-family-tc37xtp/#!?fileId=5546d46276fb756a01771bc4c2e33bdd>.
- [16] T. Nipkow, M. Wenzel, and L. C. Paulson, *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.
- [17] G. Huet, G. Kahn, and C. Paulin-Mohring, "The coq proof assistant a tutorial," *Rapport Technique*, vol. 178, 1997.

Algebraic Effects and Static Analysis for Safety-Critical Applications in Fuzion

Fridtjof Siebert, Michael Lill, Max Teufel

Tokiwa Software GmbH, Karlsruhe, Germany; email: {siebert, michael.lill, max.teufel}@tokiwa.software

Abstract

This work-in-progress paper presents the introduction of algebraic effects to the Fuzion language and how algebraic effects can be used in the context of safety-critical systems.

Fuzion is a modern, general purpose programming language that unifies functional and object-oriented paradigms into a pure functional language. Algebraic effects are used to represent and manage non-functional aspects like I/O operations or mutable state. Static analysis is used extensively at several stages in the Fuzion toolchain to verify different correctness aspects of the application.

We start with a condensed overview of the Fuzion language to then describe how algebraic effects are used to represent non-functional aspects. The Fuzion toolchain will be explained and how static analysis is used to build and validate applications. Finally, it will be shown how algebraic effects can be used to model aspects relevant to safety-critical systems.

Keywords: programming languages, algebraic effects, static analysis, safety, security

1 Introduction

Fuzion is a work-in-progress project to design a new high-level language targeting safety critical applications [1]. It provides a novel way that combines aspects of object-oriented and functional programming paradigms in a coherent and simple way. Fuzion is *pure* in the sense that no *public routine* will have any non-functional side-effects unless it is explicitly declared to do so.

2 Fuzion Language Introduction

This section gives a condensed and recursive overview of the Fuzion language and an introduction to its terminology, a tutorial is available online [2].

2.1 Building Block: Feature

The building blocks of Fuzion applications are *feature* declarations. There are different kinds of *features*, the most common kinds are *field* and *routine*. A *field* is an immutable variable of

a statically fixed *type* that is initialized with a value calculated from an *expression*. A *routine* is a callable *feature* with formal *arguments* and code given as an *expression*. A *routine* may be one of two kinds: a *function* that results in the value produced by evaluation of its *expression* or a *constructor* that defines a product type consisting of its inner *fields*. The *arguments* of a *routine* themselves are *features*, they may be *type parameters* or *fields*.

2.2 Types

Fuzion's *types* fall into two categories, product types that are defined by a *constructor* and *sum types* that are defined by *choice features*. A *choice feature* defines a tagged union type (*choice type*) of its *type parameters*, complementary to a *constructor feature* that defines a product type of its inner *fields*. Unlike *constructor features*, a *choice feature* may not be called in an *expression*. An instance of a *choice type* is created by an assignment of a value whose type is one of the *choice's type parameters* to a field of the *choice type*.

2.3 Expressions

An *expression* is code that can be evaluated to produce a result value of the expression's *type*. Any code that is executed must therefore produce a result value when it returns, but there are types like *unit* for values that do not contain any information or *void* to indicate that the expression does not return¹.

The most important expression is a *feature call* to a *routine*. On a *call*, actual values are assigned to the *routine's* formal *arguments*: For *arguments* that are *type parameters*, the actual values must be *types*, while for *argument fields* corresponding *expressions* must be given. The result of a call to a *function* is the value of the *function's expression*, while the result of a call to a *constructor* is the instance of the product type defined by that *constructor* with *arguments* set to the actual *types* and values and inner fields initialized to their *initial values*.

The only expression that permits conditional code is a *match* that takes an expression that evaluates to an instance of a *choice type*. Depending on the original type stored in the choice, evaluation proceeds with one of several expressions.

Finally, Fuzion expressions allow nested declarations described in the next sub-section:

¹e.g., as the result type of a call to *panic* that aborts with an error

2.4 Nested Features

A feature declaration itself is a Fuzion expression that results in a unit type result value. This permits fields to be nested within *routines*, but also permits the nesting of *routines*. An inner *routine* may access features declared in all of its outer *routines*. On a call to a *routine*, a reference to the outer instance is passed as an implicit argument.

2.5 Inheritance and Dynamic Binding

Constructors may inherit from other *constructor* features by adding *calls* to these *parents* in the declaration. As a result, the *child* inherits the *parents*' inner features, which the child may *redefine*.

Since Fuzion uses value semantics, using inheritance and redefinition does not require dynamic binding. However, a constructor may be defined as a *reference* type. If this is the case for the parent, the child becomes assignable to fields of the parent type and dynamic binding will be used.

2.6 Algebraic Effects

Fuzion *routines* are pure, i.e., their result depends only on the values of the actual arguments including the implicit outer instance. The only means to perform state changes or to interact with the outside world is via algebraic effects. These are features that inherit from a base feature *effect* and add effect *operations* as inner features. Effects can be *installed* to run code that can access the effect's operations in its environment. Static analysis is used to verify that all effects required for certain code are actually installed in the code's environment.

2.7 Syntactic Sugar

Fuzion uses extensive syntactic sugar to provide a more human-readable syntax for common code patterns.

Conditionals of the form *if-then-else* are internally handled like *match* expressions. This is possible since type *bool* in Fuzion's base library is defined as a *choice type* of unit types *FALSE* and *TRUE*.

Loops are supported via a powerful syntax that is, internally, mapped to tail-recursive calls² and *match* expressions.

Type inference is used extensively in the frontend such that—even though Fuzion is statically typed—types can be omitted in most cases.

2.8 Information Hiding

Visibility of features and types can be restricted as *private* (same source file, default), *module* (same module) and *public*.

²that will be optimized by the backends

3 Algebraic Effects

Purely functional code brings a number of advantages for the correctness of a complex software system: The result of every call depending only on the call's arguments simplifies (automatic) reasoning about the code, the absence of mutable state results in thread-safety and the absence of side-effects permits optimizations since there is no observable effect if code is not executed or executed repeatedly. Also, if purity of code from an untrusted source can be verified, it can safely be used without compromising a system's security.

However, for code to interact with the outside world or for mere performance reasons, actual systems must be able to perform non-functional operations. Examples include i/o operations, mutation of data, inter-thread communication, access to hw timers, sensors, actuators, aborting an operation, and many more.

Algebraic Effect handlers [3,4,5,6] are used in recent programming languages as a means to handle operations that would break the purity by having non-functional side-effects. An effect defines a set of such operations, while an effect handler provides a concrete implementation of these operations. Code that calls these operations is then said to require an instance of the given effect in its environment. The environment is essentially a stack of effects that is used to find the innermost handler for each operation to be used.

3.1 Effects in Fuzion

Fuzion uses algebraic effects to wrap non-functional operations. An effect in Fuzion is a *constructor feature* that inherits from a base library *feature* called *effect* and that defines a set of operations as inner features. An effect can be instantiated and installed to run code that uses the effect's operations. Effects are identified by their type [7].

Feature declarations in Fuzion include an optional section to list all the effect types that are required to call that feature. Features that are marked as *public* must include this information, which will be verified by static analysis.

3.2 Effects in dynamic code

A major difficulty in specifying the effects of a feature originates in the presence of dynamic code: functions that are passed as arguments to routines may require additional effects that are unknown to the called routine. E.g., one might want to log calls to a function passed to a library routine, where the effect that performs this logging is unknown to the library. The same problem may occur if a redefinition of an inherited feature uses effects that were unexpected by the original routine.

To solve this, languages like Koka introduce effect polymorphism to declare required effects explicitly [8] while the Effekt language [9] simplifies effect polymorphism by viewing effects as capabilities [10].

In Fuzion, static analysis is used both at module level and at whole application level. At module level, effect polymorphism is analyzed only to the extent that control flow reaches

the use of a given effect, while additional effects introduced by users of that module through function arguments or redefinition are ignored. At the application level, whole program data-flow analysis verifies that all uses of effects occur in environments that provide corresponding effect instances.

3.3 Effect Example

We will present a small example using an effect *temp* to model a temperature sensor that can read a temperature in degrees Centigrade:

```
temp (hdlr ()->f64) : simple_effect is
  read => hdlr()
```

Here, *temp* is the effect constructor and also its type, and *read* is the only operation, which gives a temperature reading. *read* is implemented by calling a handler function *hdlr*, which permits different implementations for different instances of this effect.

Our application main loop now requires this *temp* effect to repeatedly perform a temperature reading and printing the result unless it is larger than 41°C, when it should call *panic*:

```
main ! temp =>
do
  t := temp.env.read
  if t > 41
    panic "***_get_doctor_***"
  say "ok:_$t°C."
  time.nano.sleep (time.durations.ms 500)
```

In a deployed system, this would run with an instance of *temp* using a handler that reads the temperature from a thermometer. In a test setup, we can simulate the hardware using a handler *test_temp* that reads and modifies a mutable field *cur_temp*:

```
cur_temp := mut 37.0
test_temp =>
  t := cur_temp.get
  cur_temp <- t+0.3
  t
(temp test_temp).use main
```

This illustrates that, when using effects, we can separate the program logic, *main* in this example, from the implementation of non-functional aspects like reading a sensor.

4 Compilation Phases

The Fuzion toolchain (Fig 1) starts by compiling a set of Fuzion source files **.fz* into a Fuzion module *name.fum*. Modules may depend on other modules and compilation is done against pre-compiled modules. The frontend phase checks that the source code respects the language validity rules that include type checks, visibility rules, etc.

Module files have unique version numbers, any change or recompilation of one module requires recompilation of all modules that depend on that module. There is hence no need for mechanisms to detect incompatible changes at link or load time as in other languages³.

³Java produces an *IncompatibleClassChangeError* in some cases, *C* could fail during linking or crash at runtime in this case.

The middle end then builds an application *name.fuir* from a main module that defines a main feature plus all the modules the main module depends on. The middle end performs monomorphization, i.e., all type parameters are replaced by actual types, features called with different type parameters are specialized for all combinations of type parameters that are used in the application.

Consequently, the intermediate code used for the application is fairly simple, all types except runtime types of reference values are known. Whole program static analyzers can now process the application that is represented using the Fuzion intermediate representation.

Finally, the intermediate representation is used by one of the Fuzion backends to produce executable code. Currently, three backends are implemented: a JVM bytecode generator, one backend that create C source code to be processed by clang/llvm or other C toolchains and an interpreter that directly executes the intermediate code.

5 Static Analysis in Fuzion

The use of static code analysis is essential in all compilation phases of Fuzion: During the frontend phase, static data-flow analysis is used to verify that dependencies on effects are declared for public features, while whole application analysis can be used for a variety of applications.

5.1 Intermediate Representation

The basis of the whole-program analysis is the Fuzion Intermediate Representation *FUIR*. This representation consists of a collection of features that were monomorphized, i.e., all type parameters are replaced by actual *runtime types* while features are duplicated for every combination of runtime types found by the middle end.

Expressions that provide the code of routines are encoded using a stack-based bytecode format, comparable to Java bytecode [11]. However, there are currently only ten different bytecode instructions:

- *AdrOf* — used for call-by-references for outer instances
- *Assign* — assign a value to a field
- *Box* — create a reference instance from a value instance
- *Call* — perform a call to a routine
- *Comment* — only used for debugging
- *Const* — create an instance from serialized byte data

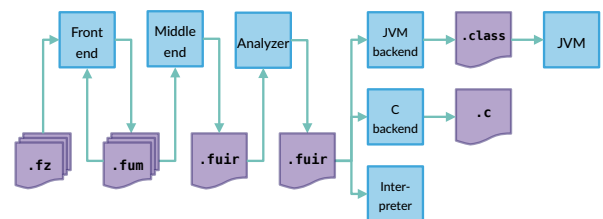


Figure 1: Fuzion toolchain and intermediate data.

- *Current* — return the instance of the current feature
- *Env* — obtain current instance of an algebraic effect type
- *Pop* — discard a value from the stack
- *Match* — extract the original value from a tagged union type value and branch to code that processes that value
- *Tag* — create tagged union type value from a value of one of the choice types

The low number of instructions helps to simplify the implementation of static analyzers and code generators.

There are no instructions for basic arithmetic operations. Instead, fundamental features like addition of values of type *i32* are calls to features of kind *intrinsic* that must be provided by the backends and handled by static analyzers correctly⁴.

5.2 Whole Application Analysis

The whole application analysis performs a data-flow analysis over an application until a fix-point is reached. As a result, upper-bound sets of possible values for all fields and expressions in different call contexts are found.

This analysis can be used for a number of purposes, e.g.

- Elimination of deactivated code
- proof of absence of errors⁵
- verification of pre- and post-conditions
- specialization of code for actual values
- determination of life-spans of instances, e.g., for automatic stack or static allocation
- user feedback on heap allocation

Furthermore, we expect the application wide data-flow analysis to be useful during the verification and validation process by producing evidence for deactivated code, developing test cases for better code coverage, or even correctness proofs by verification of pre- and post-conditions.

6 Algebraic Effects for Safety and Security

Algebraic effects can be used to manage non-functional aspects related to the safety and security of the system:

6.1 Security along SW Supply Chain

Static, program-wide analysis will find all effects required by code, including all effects required by third-party libraries. This could be used to increase the security by providing harmless handlers to suppress undesired functionality. An example is the log4shell vulnerability [12] that enabled downloading and execution of arbitrary code in a widely used logging library for Java. Static analysis would first help to detect that such effects are used. Then, the library code could be sandboxed by applications using effect handlers that prohibit operations like network access or execution of arbitrary code.

⁴which, depending on the analysis, often permits ignoring them.

⁵which essentially means error handling. like calls to *panic*. is deactivated

6.2 Safety and Real-Time Aspects

With the presence of algebraic effects as a powerful means to describe behavior that is not purely functional, we expect to use these effects to address aspects that are of particular importance to many safety-critical systems, e.g., to describe and verify real-time behavior.

The following gives a list of possible effects that we want to implement and apply in Fuzion:

- *constant time* — code contains no conditionals
- *bounded time* — code contains no unbounded recursion⁶
- *non blocking* — code performs no blocking operations
- *no heap* — code performs no heap allocation⁷
- *interruptible* — code may be aborted asynchronously
- *worst-case execution time* — execution time limit, enforced by analysis or at runtime

The flexible way that algebraic effects permit to introduce scopes of arbitrary types into the program in conjunction with the ability of static analysis to attach semantics to these effects and verify these semantics appears to give a powerful tool to handle safety and real-time aspects.

7 Conclusions and Future Work

We have presented the Fuzion project that develops a pure functional language using algebraic effects and a corresponding toolchain and showed how we expect the use of Fuzion to help enhance security and help during safety verification and validation.

The project is still in an early prototype state, we have basic implementations of the frontend, middle-end, first versions of static analysis using data-flow analysis and three different backends.

The current focus of our work is on improving the base library, in particular to model a variety of non-functional aspects using algebraic effects. The Fuzion intermediate representation is kept simple to encourage the integration with different powerful tools, an integration with proof assistants like Rocq/Coq [13] or Isabelle [14] might increase the power of correctness proofs significantly.

Fuzion has a powerful interface to call Java code, but we will need other foreign function interfaces, in particular for C, to inter-operate with legacy code. The C backend currently uses the garbage collector by Hans Boehm [15], for use of Fuzion in real-time systems, we will need to ensure that all allocation can be performed statically or use a real-time GC [16].

⁶this implies bounded loops since loops use recursion

⁷and, with a suitable GC, is hence never preempted by GC work

References

- [1] F. Siebert, “Fuzion - safety through simplicity,” *Ada Lett.*, vol. 41, p. 83–86, oct 2022.
- [2] “Fuzion Portal Website.” <https://fuzion-lang.dev>, 2024.
- [3] G. Plotkin and J. Power, “Algebraic operations and generic effects,” *Applied categorical structures*, vol. 11, pp. 69–94, 2003.
- [4] G. Plotkin and M. Pretnar, “Handlers of algebraic effects,” in *Proceedings of the 18th European Symposium on Programming Languages and Systems: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, ESOP ’09*, (Berlin, Heidelberg), p. 80–94, Springer-Verlag, 2009.
- [5] G. D. Plotkin and M. Pretnar, “Handling algebraic effects,” *Logical methods in computer science*, vol. 9, 2013.
- [6] M. Pretnar, “An introduction to algebraic effects and handlers. invited tutorial paper,” *Electron. Notes Theor. Comput. Sci.*, vol. 319, p. 19–35, dec 2015.
- [7] F. Siebert, “Types as first-class values in fuzion.” Talk at TyDe 2023: 8th ACM SIGPLAN International Workshop on Type-Driven Development, <https://fuzion-lang.dev/talks/tyde23types>, sep 2023.
- [8] D. Leijen, “Koka: Programming with row polymorphic effect types,” *arXiv preprint arXiv:1406.2061*, 2014.
- [9] The Effekt research team, “Effekt Language — Effekt Safety.” <https://effekt-lang.org/docs/concepts/effect-safety>, 2023.
- [10] J. I. Brachthäuser, P. Schuster, and K. Ostermann, “Effekt: Lightweight effect polymorphism for handlers (technical report),” tech. rep., University of Tübingen, Germany, 2020.
- [11] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java Virtual Machine Specification, Java SE 21 Edition*. Oracle America, Inc., sep 2023.
- [12] “Cve-2021-44228 apache log4j2 jndi features do not protect against attacker controlled ldap and other jndi related endpoints.” <https://www.cve.org/CVERecord?id=CVE-2021-44228>, dec 2021.
- [13] Coq Development Team, “The coq proof assistant.” <https://coq.inria.fr/>.
- [14] Isabelle Contributors, “Isabelle proof assistant.” <https://isabelle.in.tum.de/>.
- [15] H.-J. Boehm, “Space efficient conservative garbage collection,” in *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation, PLDI ’93*, (New York, NY, USA), p. 197–206, Association for Computing Machinery, 1993.
- [16] F. Siebert, “Concurrent, parallel, real-time garbage-collection,” in *Proceedings of the 2010 International Symposium on Memory Management, ISMM ’10*, (New York, NY, USA), p. 11–20, Association for Computing Machinery, 2010.

An Iterative Benchmark Configuration Method for Quantifying Multi-core Interference

Sébastien Levieux, Frank Singhoff, Stéphane Rubini

Lab-STICC UMR CNRS 6285, University of Brest, 29200 Brest, France; email: firstname.lastname@univ-brest.fr

Philippe Plasson, Pierre-Vincent Gouel, Lee-Roy Malac-Allain

LESIA, Observatoire de Paris, Université PSL, CNRS, Sorbonne Université, Université Paris Cité, 5 place Jules Janssen, 92195 Meudon, France; email: firstname.lastname@obspm.fr

Lucas Miné, Gabriel Brusq

Centre National d'Etudes Spatiales (C.N.E.S.), 18 av. Edouard Belin, 31401 Toulouse, France; email: firstname.lastname@cnes.fr

Abstract

Interference within a multi-core architecture may have several origins. Understanding where interference comes from is mandatory for verification and certification purposes. Unfortunately, the complexity of current architectures makes it difficult to quantify such interference. In this article, a new approach is introduced that enables benchmark configurations to isolate and quantify interference. An experiment with DMA interference is presented and shows a WCET overhead of up to 0.26% at 25 Mbit/s. This experiment was also able to discover and identify interference related to DMA, such as interruptive flow overhead, around 3% for 25 Mbit/s, or packet transmission memory access overhead, around 9% for 25 Mbit/s.

1 Introduction

Predicting the temporal behavior of tasks running in a multi-core system is hard due to the number of various interference that tasks may suffer [1]. Many approaches have attempted to quantify interference. Several are model-based [2], which involves modeling the system and, for example, simulating its execution. Other methods involve measurement on the system itself [3] by running it under different scenarios to obtain the interference and its impact.

Problem Statement The context of this article is flight software for space missions. Such mission-critical software undergoes a cycle of reviews during which schedulability analysis must be performed to justify the designed real-time architectures. For example, PLATO [4] flight software architecture was justified using an AADL model and simulations with Cheddar [5] as early as the Preliminary Definition Review (PDR).

When schedulability is investigated in such a context, and when a multi-core architecture is used, it may be difficult to model all interference that could occur because of the hardware mechanisms and their interactions.

Contribution In this article, a new approach is introduced to understand the interference that an application may suffer when running on a multi-core architecture. The proposed approach enables the production of a set of configurations, which are combinations of different viewpoints of the target platform and the application. These configurations can then be used to associate metrics (collected at execution time) and interference (produced by one or more components).

The rest of the article is organized as follows. Section 2 presents background. Section 3 introduces the method proposed to quantify interference, following in section 4 by an example and preliminary results obtained from it. Finally, related works and the conclusion complete the article, respectively in sections 5 and 6.

2 Background

This section introduces the notions and terms required to understand the proposed benchmarking method.

2.1 Multi-core Architecture and Interference

In this article, interference is a delay in the task execution time caused by the action of another component in the system.

A multi-core architecture is composed at least of two cores and, generally, of one or further cache units. In addition, each architecture provides a set of hardware mechanisms to ensure robustness, reliability, performance...

With this type of architecture, the interference suffered by a task may come from different sources [1]. From a hardware point of view, DMA transfers, for example, generate hardware interrupts for each packet of sent data. Another example is Compare and Swap (CASA) instructions that lock the memory bus for execution. When two tasks use the same cache unit, the miss rate may increase, and accesses to the main memory may cause additional delays due to such interference. From a software point of view, the activity of other tasks in the system may be another source of interference. The use of a memory bus may be concurrent with the activity of tasks producing memory accesses on other cores. Shared

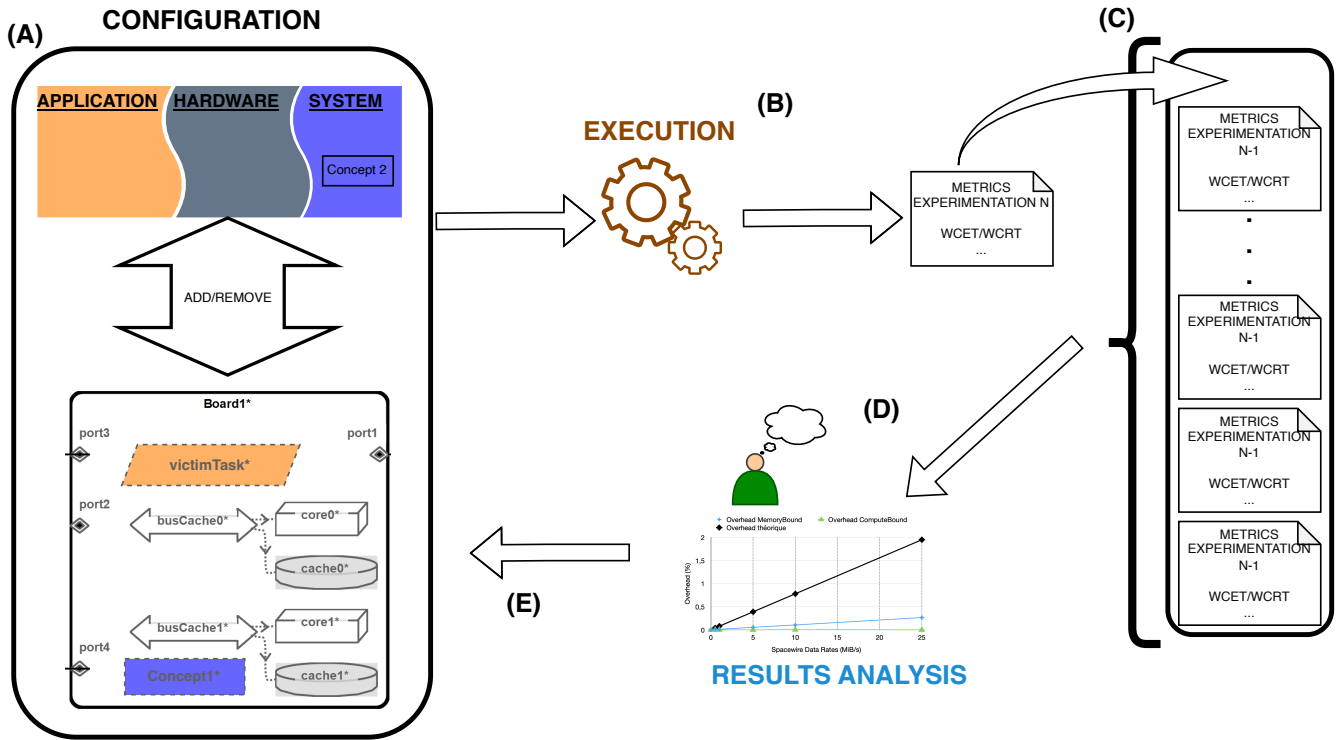


Figure 1: Interference Analysis Process

memory locked by tasks, or any preemption are also sources of interference.

In this work, we consider all sources of interference.

2.2 Benchmark

A benchmark is an application designed to be run on a target platform to evaluate its performance. For example, the Mälardalen benchmark suite [6] aims to evaluate WCET (Worst Case Execution Time). PapaBench [7] provides a set of applications useful for scheduling analysis. From the point of view of interference in a multi-core architecture, Rodinia [8] offers a representative set of multi-core applications. Splash-2 [9], and its extension Splash-4 [10], provide a set of applications characterized in particular by their memory traffic. The latter allows us to show interference related to memory bus.

2.3 GERICOS

GERICOS [11] is a C++ framework developed by LESIA for space applications. This framework offers an integrated solution for rapidly developing multi-core applications using an AMP (Asymmetric MultiProcessing) approach. The framework allows the developer to define application architectures and the assignment of each component to each core. The C++ classes implementing the application are defined independently of the core on which they will run. GERICOS also provides services to measure task WCET and WCRT (Worst Case Response Time). However, in GERICOS, there is no means to quantify the delays for each interference that contributes to WCET.

3 Proposed Approach

In a multi-core architecture, it may be difficult to obtain the accurate delay resulting of a specific interference. The WCET,

even though it is supposed to be measured in isolation, may suffer interference from the hardware resources that interact with the system in the background.

Our work aims to quantify all interference through metrics, but in this article we illustrate with an example concerning only hardware resources. In this article, two metrics are considered: the WCET and WCRT. A method is proposed to configure a benchmark, and, with a set of experiments, deduce the delay caused by a hardware resource. First, we explain what we mean by benchmark configuration. Second, we present the proposed analysis process. Finally, we introduce examples of benchmark configurations that will be used in the next section of this article.

3.1 Benchmark Configuration

A benchmark configuration is produced from 3 viewpoints represented by (A) in figure 1. A configuration is the assignment of the various components of the 3 viewpoints on the target platform. In addition, one or more victim tasks are also defined in each benchmark configuration. Victim tasks are tasks from which the metrics are retrieved after benchmark execution to measure delays related to interference.

The first viewpoint is related to the kind of application the benchmark will run. The proposal is to design and run a set of applications that stress the system in a particular way. For example, in this article, as [12], two types of applications are investigated: *MemoryBound* applications that are composed of a task making memory accesses during its execution, and *ComputeBound* which runs a task taking CPU time only, i.e. without any memory access.

The second viewpoint models any hardware entities and mechanisms that may raise a potential interference. For example,

	APPLICATION					HARDWARE					SYSTEM													
Configuration Name	Compute Bound	Memory Bound	SpaceWire Sender	SpaceWire Receiver	Interrupt Bound	Core	Instruction Cache	Data Cache	Interrupt (IRQ)	DMA	Spinlock	Circular Buffer	Single Buffer	InterCore Manager	Interrupt Handler	Shared Object	Single Object	Synchronized Object	Task	Timer	OS	Scheduling Policy	Allocation Resource Policy	
Configuration 1.1	1	0	1	1	0	2	True	True	True	True	0	4	4	0	2	0	1	0	3	3	RTEMS	HPF	PIP	
Configuration 1.2	0	1	1	1	0	2	True	True	True	True	0	4	4	0	2	0	1	0	3	3	RTEMS	HPF	PIP	
Configuration 2.1	1	0	1	1	0	2	True	True	True	True	0	4	4	0	2	0	1	0	3	3	RTEMS	HPF	PIP	
Configuration 2.2	0	1	1	1	0	2	True	True	True	True	0	4	4	0	2	0	1	0	3	3	RTEMS	HPF	PIP	
Configuration 3.1	1	0	1	1	0	3	True	True	True	True	0	4	4	0	2	0	1	0	2	2	RTEMS	HPF	PIP	
Configuration 3.2	0	1	1	1	0	3	True	True	True	True	0	4	4	0	2	0	1	0	2	2	RTEMS	HPF	PIP	

Figure 2: The 3 viewpoints for the GR712RC board

most of the current multi-core architectures have different levels of cache or hardware interruption types. Each of these entities may be modeled since they may cause interference.

The last viewpoint models system artifacts that may change the scheduling or the synchronization of the task composing the benchmark. For example, GERICOS provides the concept of *GscSharedResource* that enforces critical section on shared data with spin locks and mutexes. In this viewpoint are also specified the scheduling policy, the operating system and the allocation resource policies.

Figure 2 is a 3-viewpoint model of the benchmark for the GR712RC board used in the section 4. Each line represents a configuration. Each column stores either a quantity or a boolean indicating whether the element is activated or not. For example, configuration 1.1 uses 2 cores and the instruction cache is enabled. In the Application view, *MemoryBound* is a task that constantly performs memory accesses, and *ComputeBound* which only takes CPU time without memory any accesses. *SpwSender* and *SpwReceiver* are tasks that respectively send and receive data packets on the SpaceWire ports of the board. Finally, *InterruptBound* is a task that generates hardware interrupts during its execution.

At hardware viewpoint, we have to specify, the number of cores used during execution, data and instruction cache units, IRQ interrupts and DMA transfers from SpaceWire ports (which can be enabled or not).

The last viewpoint, i.e. the system viewpoint, contains GERICOS concepts, such as the *InterCoreManager*, a task in charge of the communications between cores. Notice that several concepts in this viewpoint, such as spinlock or the operating system, may be not specific to GERICOS.

3.2 Analysis Process

We now describe how benchmark configurations are expected to be used iteratively to understand how interference occurs.

The analysis process is shown in figure 1. The process consists of iteratively running several benchmark configurations.

In the benchmark configuration phase (A), two benchmark configurations are produced at least. The first, named the victim configuration, suffers desired interference. The second configuration, named reference configuration, does not suffer any interference. The reference configuration execution is compared to the victim configuration execution to discover interference.

Benchmark configurations are executed on the platform in (B), and from the victim tasks, a set of metrics is retrieved (i.e. WCET and WCRT).

In (C) and (D), metrics retrieved from configuration execution helps the user to understand whether components contribute or not to interference on the victim tasks. At those steps, two outcomes are possible: either the number of executed configurations is sufficient to understand interference, or interference is not identified and the analysis process is repeated.

To sum up the method, we derive benchmark configurations to progressively eliminate undesired interference until the interference created by a specific hardware resource is isolated, or at least quantified. The interest of this method is to be able to characterize, using a configuration cycle, interference caused by a specific hardware resource.

In this article, we focus on WCET and WCRT to quantify interference. Notice that other metrics could be mandatory to understand the system behavior. For example, *CPU load* or *DMA transfers* can be the indicator of a background activity. Furthermore, *Cache information* such as the L1 or L2 hit/miss rates may reveal precious information on memory usage and can explain interference origin. Hardware counters would be used also for such a purpose.

3.3 Example

We now illustrate the proposed method with a use case in which we expect to quantify interference due to bus contention during DMA transfers.

The hardware viewpoint in figure 4 models two GR712RC boards. Each board is a Dual-Core LEON3FT SPARC V8 processor with a 32KiB L1 cache for each core and 4 SpaceWire ports. The GR712RC block diagram is represented in figure 3.

We model this system as 6 configurations. Each of them represents different steps to isolate interference and will be used to run the benchmark. To investigate the impact of DMA transfers on bus contention different measurements are done with different transfer rates. All the results are discussed in the next section.

The first configuration, named 1.1, is the reference configuration. In this configuration, a task named *SpwEmitter*, which periodically sends data packets on the SpaceWire network, is assigned and run to *core0*. On the other core, called *core1*, runs the task *SpwReceiver* which is receiving packets emitted from the *core0*. Connection (A) in the figure 4

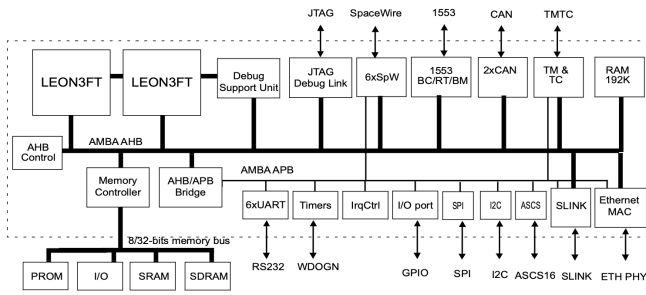


Figure 3: GR712RC Block Diagram

models such communication. The victim task, *ComputeBound*, is placed on *core1*. The victim task, by its nature, cannot be subject to interference from memory accesses and is therefore used as a reference. The second configuration, named 1.2, is the 1.1 configuration when *ComputeBound* is replaced by *MemoryBound*.

The third configuration is the reference configuration, noted 2.1, is similar to the first configuration but *SpwReceiver* is moved on the *core0*. The objective is to isolate the victim task on the *core1*, i.e. not suffer interference from the presence of *SpwReceiver* on the same core. The fourth configuration, named 2.2, is the 2.1 configuration when *ComputeBound* is replaced by *MemoryBound*.

The fifth configuration is the reference configuration, noted 3.1, *SpwEmitter* is assigned on *core0* of *Board2*. *SpwRmapConfigurator* is run on *core1* of *Board1*. Communications between the two boards are modeled by the connection (B) of figure 4. The task *SpwReceiver* is replaced by *SpwRmapConfigurator* in this benchmark configuration. The victim task is assigned on the *core0* of *Board1* to fully isolate it of all perturbations, except the bus contention created by the SpaceWire network during DMA operations. The sixth configuration, named 3.2, is the 3.1 configuration when *ComputeBound* is replaced by *MemoryBound*.

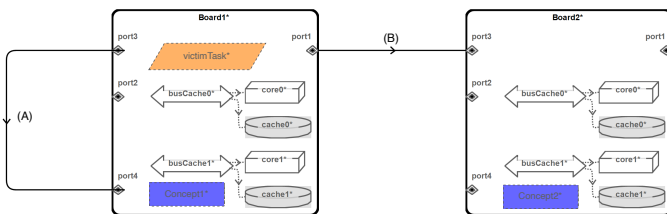


Figure 4: Example of a benchmark configuration

The next section shows the results when running the benchmark configurations above.

4 Evaluation

In this section, we illustrate the proposed analysis process with an experiment. This section presents the goal, the details

of the experiment process, and the results obtained.

The aim of this experiment is to characterize the interference of bus contention in a DMA transfer context. To quantify this interference, a communication is generated on a SpaceWire network which creates DMA transfers and then generates contention on the memory bus.

The system is evaluated with various data rates on the SpaceWire network: 0.5 Mbit/s, 1 Mbit/s, 5 Mbit/s, 10 Mbit/s and 25 Mbit/s. In the configurations, in order to ensure packet transmission, the tasks responsible for sending and receiving data have higher priority levels than the victim tasks. The size of a packet is constant. The period of the sending task defines the data rate. The victim tasks are executed 10 times, with an execution time of 1 second and a period of 10 seconds. Statistics related to tasks, such as WCET, are measured through the *GscMethodReport* class of *GERICOS*. *GscMethodReport* provides a set of functions that will be called by the task itself to measure and record data.

We apply the process proposed in section 3 by configuring and running the benchmark with 6 configurations. The results, shown below, compare the overhead due to interference on the WCET of the victim tasks. Each graph shows the results of two comparable configurations for a different type of victim task.

First, we configure the benchmark according to the configurations 1.1 and 1.2 described in section 3, run it, and get results of figure 5. In this figure, the overhead of *ComputeBound* and *MemoryBound* are closer, respectively of 11,41% and 11,33% for 25 Mbit/s, which means suffered interference is the same. However, *ComputeBound* does not use the memory bus, so interference should not be found for this task with these configurations. In fact, the first configuration creates interference than even a task that does not access memory suffer. It is caused by *SpwReceiver*, which is on the same core, and has a higher priority than the victim task. Such results lead us to investigate the system performance with the configurations 2.1 and 2.2.

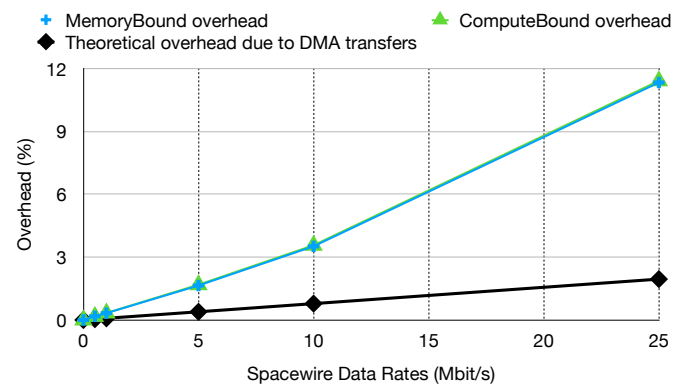


Figure 5: Throughput for configurations 1.1 and 1.2

Figure 6 shows the results for the configuration 2.1 and 2.2, described in section 3. In this figure, we show the *ComputeBound* overhead curve rises more slowly than *MemoryBound* overhead curve, reaching a gap of 0.86% at 25 Mbit/s. This indicates a difference in suffered interference which cannot

be related to the contention on the memory bus, as 89.69% of the interference perceived by *MemoryBound* is also perceived by *ComputeBound*. This may be due to the number of interruptions caused by DMA transfers, which are performed on the same board as the task being analyzed. To investigate this assumption, we run the next configurations.

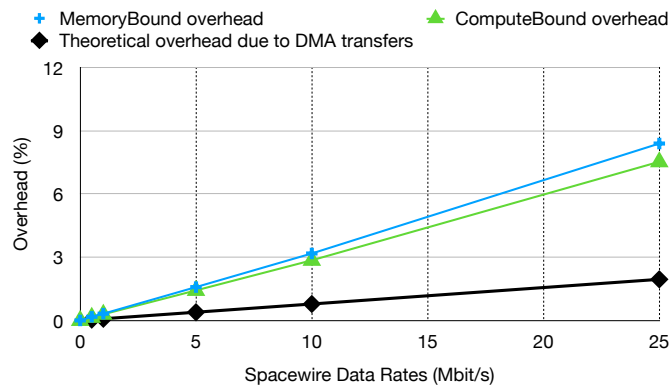


Figure 6: Throughput for configurations 2.1 and 2.2

To complete the analysis, we run the benchmark with the configurations 3.1 and 3.2, and get the results of the figure 7. Now, *ComputeBound* suffers no interference with an overhead of 0.1 ms between 10 Mbit/s and 25 Mbit/s; whereas *MemoryBound* suffers interference of up to 0.26% overhead at 25 Mbit/s. Assuming that the performance gap between *ComputeBound* and *MemoryBound* is due to memory accesses, it means that we succeeded to isolate interference created by DMA accesses on the memory bus with this last experiment.

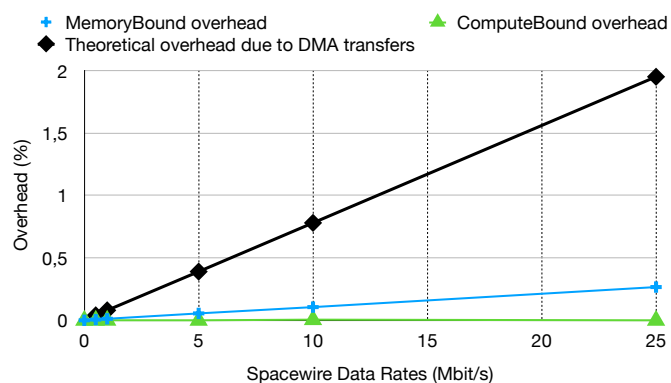


Figure 7: Throughput for configurations 3.1 and 3.2

In the 3 figures presenting the results, a curve called theoretical overhead is displayed. In the case of the PDR of PLATO project, interference of DMA transfers was estimated by a theoretical computation. Our experiments showed that this theoretical computation predicted an interference overhead of 7.33 times higher than those measured for 25 Mbit/s with configuration 3.2. This is explained because theoretical computation was carried out with the assumption that all DMA accesses are concurrent and have the highest priority, which is a pessimistic case that does not occur with configurations 3.1 and 3.2. In figure 5 and 6, the measurements show higher values than the theoretical curve because the measures are composed of different interference origins and not only from

DMA memory bus contention, this motivated configurations 3.1 and 3.2 to better isolate DMA interference.

Furthermore, DMA transfers using SpaceWire also cause interrupts on reception, and memory accesses on transmission. Figures 5, 6 and 7 show the overhead caused by these two mechanisms. Switching the *SpwReceiver* to *core1* in configurations 2.1 and 2.2, shows that interrupts related to reception no longer interfere with the victim task. This leads to a loss of about 3% overhead for 25 Mbit/s between the curves in figure 5 and 6. Similarly, for configurations 3.1 and 3.2, the packet emission is no longer on the same board as the victim task, so it no longer competes for access to the memory bus. This reduces overhead by about 9% for 25 Mbit/s between the curves on the figure 6 and 7.

In conclusion, by applying the proposed method, the set of configurations that was produced enabled us to characterize the interference of bus contention in a DMA context. In addition, within the PLATO context, the measurements demonstrated the pessimism of the theoretical evaluation. Finally, all the curves enable us to make assumptions about the overhead caused by other mechanisms resulting from DMA transfers.

5 Related Works

A lot of works deals with interference management in multi-core architectures. Interference can be either predicted, measured or mitigated by proposing software and/or hardware designs that reduce them [13].

In [12], the author introduces a method to quantify interference by measurements. He focuses on several hardware components such as shared L1 and L2 caches by experiments with memory bound and compute bound benchmarks. Applied benchmarks are similar to the one we have used in our configuration process.

In contrary, the multi-phases task model [14] avoids interference with the PREM model [15]. Tasks are designed as set of different phases that limit memory bus interference. The main drawback of this approach is to not support legacy programs. Supporting legacy programs is mandatory for systems targeted by the configuration process we propose.

Several benchmarks provide means to understand interference. Rodinia [8] and Splash-2 [9] are examples of them. Interference analysis with these benchmarks focuses on memory interference and does not consider many others such as interruption, DMA or operating system interference. Our approach expects to quantify any type of interference.

6 Conclusion

In this article, a method to isolate and evaluate interference in multi-core architectures is proposed. The approach consists of running several configurations of a benchmark and measuring task metrics to understand how interference contributes to the metrics. An experiment has shown that the proposed method can characterize, from a set of configurations, interference of a shared hardware resource through the WCET. For the PLATO mission, the theoretical computations for such interference, for 25 Mbit/s, were 3.25 times higher than our measurements which estimated it to 0.26% of the WCET. This demonstrates

the pessimism of the theoretical results and the accuracy of the benchmarking method we proposed. Finally, the experiment was also able to identify interference related to DMA, such as interruptive flow overhead, around 3% or packet transmission memory access overhead, around 9%.

In the future, we expect to evaluate the method on more complex architectures such as GR740RC, a naked quad-core with different cache levels and interference reduction mechanisms. Furthermore, this article has presented a method that gives users the ability to design benchmark configurations according to their understanding of the system behavior, which can be difficult to achieve. In the next steps, we expect to help users by defining configuration design patterns for specific hardware resource patterns. Finally, the hardware architecture of the GR712RC does not provide statistics on system interactions. More and more architectures are introducing performance counters [16] [17] [18] [19], notably the GR740RC. Performance counters may contribute to quantify/identify interference. We plan to integrate these counters in our benchmark configuration process.

References

- [1] T. Lugo, S. Lozano, J. Fernández, and J. Carretero, “A survey of techniques for reducing interference in real-time applications on multicore platforms,” *IEEE Access*, vol. 10, pp. 21853–21882, 2022.
- [2] V. A. Nguyen, E. Jenn, W. Serwe, F. Lang, and R. Ma-teescu, “Using model checking to identify timing interferences on multicore processors,” in *ERTS 2020-10th European Congress on Embedded Real Time Software and Systems*, pp. 1–10, 2020.
- [3] J. Stärner and L. Asplund, “Measuring the cache interference cost in preemptive real-time systems,” in *Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pp. 146–154, 2004.
- [4] P. Plasson, G. Brusq, F. Singhoff, H. N. Tran, S. Rubini, and P. Dissaux, “Plato n-dpu on-board software: an ideal candidate for multicore scheduling analysis,” in *10th European Congress ERTSS Embedded Real Time Software and System*, 2022.
- [5] 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.
- [6] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, “The mälardalen wcet benchmarks: Past, present and future,” in *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.
- [7] F. Nemer, H. Cassé, P. Sainrat, J.-P. Bahsoun, and M. De Michiel, “Papabench: a free real-time benchmark,” in *6th International Workshop on Worst-Case Execution Time Analysis (WCET’06)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.
- [8] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, “Rodinia: A benchmark suite for heterogeneous computing,” in *2009 IEEE international symposium on workload characterization (IISWC)*, pp. 44–54, Ieee, 2009.
- [9] J. M. Arnold, D. A. Buell, and E. G. Davis, “Splash 2,” in *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pp. 316–322, 1992.
- [10] E. J. Gómez-Hernández, J. M. Cebrian, S. Kaxiras, and A. Ros, “Splash-4: A modern benchmark suite with lock-free constructs,” in *2022 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 51–64, IEEE, 2022.
- [11] P. Plasson, C. Cuomo, G. Gabriel, N. Gauthier, L. Gueguen, and L. Malac-Allain, “Gericos: A generic framework for the development of on-board software,” *DASIA 2016-Data Systems In Aerospace*, vol. 736, p. 39, 2016.
- [12] T. Beck, *Evaluation and analysis of Linux applications on multi-core processors in a space environment*. PhD thesis, Toulouse, ISAE, 2023.
- [13] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis, “A survey of timing verification techniques for multi-core real-time systems,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–38, 2019.
- [14] C. Maia, L. Nogueira, L. M. Pinho, and D. G. Pérez, “A closer look into the aer model,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, IEEE, 2016.
- [15] R. Pellizzoni, E. Betti, S. Bak, G. Yao, J. Criswell, M. Caccamo, and R. Kegley, “A predictable execution model for cots-based embedded systems,” in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 269–279, IEEE, 2011.
- [16] G. Cabo, S. Alcaide, C. Hernández, P. Benedicte, F. Bas, F. Mazzocchi, and J. Abella, “Safesu-2: a safe statistics unit for space mpsoes,” in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1085–1086, IEEE, 2022.
- [17] T. Scheipel, F. Mauroner, and M. Baunach, “System-aware performance monitoring unit for risc-v architectures,” in *2017 Euromicro Conference on Digital System Design (DSD)*, pp. 86–93, IEEE, 2017.
- [18] F. Cosimi, F. Tronci, S. Saponara, and P. Gai, “Analysis, hardware specification and design of a programmable performance monitoring unit (ppmu) for risc-v ecus,” in *2022 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 213–218, IEEE, 2022.
- [19] M. Lei, T.-Y. Yin, Y.-C. Zhou, and J. Han, “Highly reconfigurable performance monitoring unit on risc-v,” in *2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, pp. 1–3, IEEE, 2020.

Improving Availability in a Robotic Application without Loss of Safety

Gema Rincon, Carlos-F. Nicolas

Ikerlan Research Center, J. M. Arizmendiarrieta 2, 20500 Arrasate-Mondragon (Spain), Spain; Tel: +34 943 712 400; email: {grincon, cfnicolas}@ikerlan.es

Tomaso Poggi

Mondragon University - Faculty of Engineering, Loramendi 4, 20500 Arrasate-Mondragon (Spain); Tel: +44 123 987 6543; email: {tpoggi}@mondragon.edu

Abstract

In our automated and industrialized world, ensuring safety in human-robot interaction is essential, a complex engineering task especially in dynamic environments. The widespread adoption of collaborative and autonomous robots across various sectors underscores the critical need for robust safety measures.

This article examines the current state of safety in collaborative robotics and proposes a strategy for assessing the safety of the robot task against the indications of existing standards. If the task is not considered safe in the current environment, a new task is sought for the robot, which increases its availability. These addresses dynamic environments where robots and humans coexist, allowing the autonomous robot to make task decisions based on safety considerations.

Keywords: Safety, Autonomous Robots, Collaborative robots, Availability.

1 Introduction

Industry 4.0 integrates intelligent elements into industries, enabling real-time decision-making, increased productivity, and flexibility in manufacturing and distribution processes [1]. This necessitates autonomous and adaptive industrial systems, with Digital Twins, Industrial Internet of things (IIoT), and Cyber-Physical Systems (CPS) playing key roles. However, current systems find it difficult to adapt to unforeseen events, thus showing the need for greater flexibility. With advancements in Industry 4.0, human-robot collaboration (HRC) gains significance in industrial processes.

Introducing cobots in industrial settings highlights the need to prioritize worker safety due to their potential economic benefits and versatility, despite the new safety challenges posed by human interaction [2]. Therefore, adapting verification techniques to effectively tackle these challenges and ensure safety in robotics becomes even more critical [3]. Additionally, the implementation of robotic systems without physical barriers requires thorough risk assessment and the establishment of robust safety measures to safeguard workers and optimize operational efficiency [4].

In general, safety in collaborative robotic environments is ensured by stopping or reducing the speed or torque of the robot, thus causing an interruption of the task and a consequent momentary loss of performance. The objective of this research is to enhance the availability of robotic applications while keeping the safety of the system uncompromised. To achieve this goal, automated metrics and evaluators will be deployed in ROS 2, to assess the safety of the environment, allowing the robot to make optimal decisions. This approach seeks to ensure both effectiveness and safety of robotic applications in dynamic and evolving environments. We will seek to minimize unnecessary stops by allowing the robot to adjust its trajectory in real time by means of motion predictions [5]. In this way automatic stopping will be kept only as a last resource safety measure in situations where the high level risk demands it, thus preventing unwarranted interruptions in robot operation.

The article provides a critical analysis of current collaborative robotics safety practices, identifying areas that require enhancements. The main contribution is the introduction of a high-level software framework to obtain a safety assessment of the environment that can be used to increase the availability. Section 2 reviews existing standards, while Section 3 proposes a solution, followed by a conclusion.

2 State of the art of robotics safety

Safety in robotics remains a challenge, as it lacks a general validation standard such as ASAM in the automotive industry [6]. The European project (COVR) attempts to solve this problem by proposing a set of tools that provide safety validation protocols [7]. However, the project is limited to the requirements for the design of a safe robotic application. In addition, other research, which also discusses safety in robotics and the need for regulatory standards, emphasizes the importance of integrating safety from the very beginning in the design phase [8, 9].

To date, there is still a lack of a software tool that assesses the safety of an online application, enabling it to make secure decisions.

2.1 Safety standards





Safety functions are crucial in coexistent robotics, where robots share workspace with humans. Advanced technologies such as force feedback, low-inertia servo motors, elastic actuators, and collision detection systems facilitate this coexistence.

In robotics, key standards include ISO 13482 for healthcare robots, ISO 10218 for industrial robot safety, and ISO 15066 for safe human-robot collaboration. For this study, safety functions will be aligned with ISO 10218 and ISO 15066, which describe the use of safety features. ISO 10218-1 describes the required safety functions, while ISO 15066 details their application in robotic collaboration scenarios. These standards identify specific hazards and enable the selection of risk reduction measures (RRMs). Operation methods, such as SRMS (Stop and Restart), SSM (Speed and Separation Monitoring), PFL (Power and Force Limiting), HG (Hand Guiding), defined in ISO 10218, can be implemented with various tools and technologies [10, 11].

ISO 15066 classifies human-robot interactions into two main types: quasi-static and transient. Quasi-static interactions involve slow or minimal motion between the robot and the human, while transient interactions are characterized by brief and dynamic contact. This study specifically examines transient interactions, emphasizing the need for safety measures even when autonomous robots and humans share space without direct interaction.

Sections 5.10 of ISO 10218-1 and 5.5 of ISO 15066 detail the functionalities that must be limited or monitored in collaborative operation applications; see Table 1. In addition, these points are essential for the implementation of safety functions in robotic applications.

Table 1: Functionalities for Collaborative Operation Applications: ISO 10218-1 and ISO 15066 Sections 5.10 and 5.5 [10].

Safety function	Image	Description
Safety-rated monitored stop		The robot must stop if a person enters its workspace
Hand guidance		Specifies manual mode activation and its requirements
Speed and separation monitoring		Specifies robot speed and safety distance requirements
Power and force limiting		Specifies torque and force limits for the robot

ISO 15066 specifies that not all the functionalities should be included. For instance, the Safety-rated Monitored Stop function, which halts the robot upon detecting a person in the shared workspace, may be omitted to prioritize continuous operation.

While manual guidance is assessed during commissioning tests, it does not impact the proposed software framework's

design. Safety functions such as speed and separation monitoring, together with force and torque limits, are an integral part of the robot, which is detailed in section 5.5 of ISO 15066. [12].

In collaborative operations with power and force limitations, contact between robot and operator can occur in a variety of situations: as part of the application, in accidents or application failures such as hardware failures. Due to the different types of interaction, both passive (increased contact surface, energy absorption, limitation of moving masses) and active (control of forces, velocities and energy) measures are applied to reduce risks [13]. Power and force control limits should be designed not to exceed the applicable threshold values, and robots should be able to set these limits. The procedures for establishing and maintaining these thresholds are described in detail in Annex A of ISO 15066 [12].

Safe speed monitoring and automatic stop functions on the robot are essential, being activated if the separation distance with a human falls below the evaluated threshold. The dynamic determination of the protective separation distance considers factors such as operator speed, robot reaction and stopping, and position uncertainty. This distance can vary depending on the application conditions. During automatic operation, hazardous parts of the robotic system should never approach the operator beyond the protective separation distance. When the distance decreases, the robot stops automatically, applying measures such as speed reduction or an alternative route to avoid violating the safety distance. Continuous speed and separation monitoring is applied to all people in the collaborative space, adjusting to the number of people present. A safety speed limitation can also be implemented to avoid exceeding the limits set by the risk assessment [12, 14].

2.2 Safe software in collaborative robotics

Proposing software as a response to enhancing robotics safety introduces a new aspect to consider, as the safety of the software to be used must be verified. ISO 31010 details common conflicts in software analysis. It demonstrates that it is not only application risks that affect safety, but that it is relevant especially in safety to know the fidelity of each element that makes up the system. Therefore, it is necessary to understand the critical aspects of the software, knowing what faults it can commit, that is to say, its weakest parts [15]. In our case, we will be using ROS 2, so it will be crucial to understand the fidelity it provides and what elements are essential.

ROS 2 is a robotic middleware that is making additional efforts to enhance safety within the ROS 2 ecosystem. It is important to mention that safety functions will not be integrated within ROS 2, but will be separated into safe software and hardware. This is exemplified by the ROS 2 Critical Security Working Group. The goal of the working group is to support the development of safe robots and systems with ROS 2, focusing on the creation of tools, libraries and documentation. Its future projects are the enhancement of the open source requirements management tool Doorstop and the development of reusable nodes such as watchdog nodes [16].

3 Problem formulation and proposed approach

This research proposes a ROS 2 software module to evaluate the environment and calculate real-time risk associated with robot actions, enhancing risk prevention and system availability. Figure 1 shows two rectangles differentiated by colour: the green rectangle symbolizes elements of the environment, while the yellow rectangle represents software components. Within the yellow rectangle, two essential elements stand out: The "Evaluator" evaluates the safety information in the environment, while the "Task List" directs the control of the robot based on this evaluation, determining the execution of the task or switching to a new safe task.

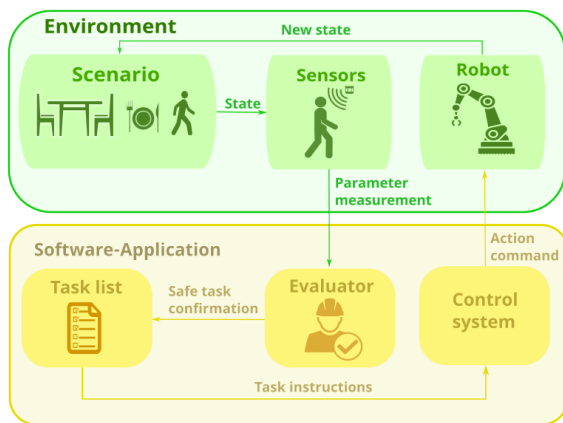


Figure 1: Diagram of the general logical flow of the proposal, showing the relationships between the established blocks.

3.1 Evaluator

This software makes use of data from the external world environment to generate a spatial map around the robot, displaying the level of risk (see figure 2). In this way, it enhances the decision-making process of the software application by enabling real-time risk assessment, thus improving overall availability in dynamic environments. The main contribution is that, through the use of standards, it is possible to set specific requirements and to determine which parameters are necessary for risk calculation.

The evaluator module determines the degree of risk using a Hazard Rating Number (HRN) equation [17], in addition to the use of the ISO 12100 standard to understand each parameter of the equation [18].

$$\text{HRN} = \text{LO} \times \text{FE} \times \text{DPH} \times \text{NP} \quad (1)$$

- Likelihood of Occurrence (LO)
- Frequency of Exposure (FE)
- Degree of Possible Harm (DPH)
- Number of Persons at risk (NP)

To understand the parameters of the HRN equation, Reference is made to ISO 12100 (identification and assessment of risks in industry).

Following ISO 12100 guidelines, a list of system environment hazards is compiled. In particular, section 5.5.2 of this standard deals with risk assessment, which is based on the severity of the damage and the probability of occurrence of the hazard. This section also evaluates human exposure to hazards, potential hazardous events, and preventive measures [18], aligning with components of the HRN equation. Parameters like damage severity will be determined based on this standard.

The output will be a spatial risk map (figure 2), divided into four zones around the robot, each indicating a specific risk level. The strategy entails enveloping the robot with two concentric layers: an inner danger zone, where strict limits are applied, and an outer attention zone, where the robot adjusts its trajectory and tasks based on risk levels.

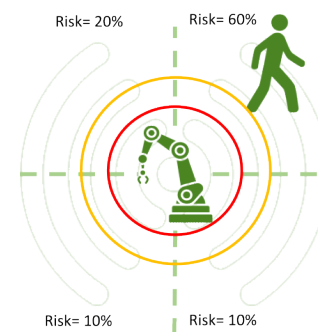


Figure 2: The figure shows the robot and its environment divided into four areas for task selection according to the level of risk

The parameters will be selected by reviewing the ISO 15066, identifying safety functions related to distance, speed, torque and force [12]. Thus, factors such as object location (i.e. the distance between the robot and other objects), robot speed, torque and force are crucial considerations. The occupancy parameter is relevant too, as the safe availability of the robot could be affected if the space is crowded. Therefore, the parameters to consider are:

- Occupancy: Number of people in a specific area, measured in people per square meter.
- Localization: The distance at which the objects are placed from the robot and where they are positioned.
- Limit speed: At lower speeds, tasks are safer; velocity should not exceed 250 mm/s when humans are in the attention zone [10].
- Force and torque limit: By limiting the robot torque, the possibility to cause hazards will be reduced.

In Figure 3, the diagram outlines the Evaluator block's phases, featuring conditional (if) questions in elliptical blocks and corresponding actions in rectangular ones. It is split into two sections: one for initial dynamic parameter analysis and another for subsequent risk calculation. Safe speed and force values for each zone (envelope) of the robot can be determined using the equations provided in ISO 15066 [12].

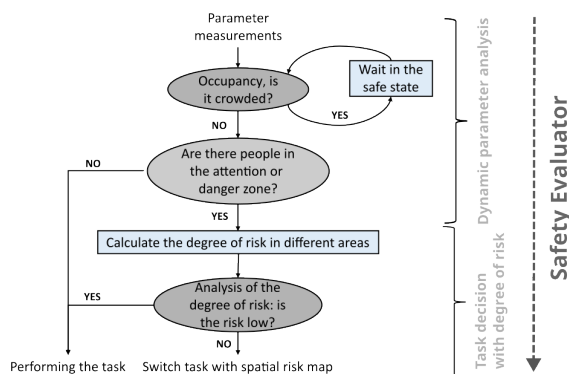


Figure 3: Evaluator block operation process. The logic is divided into two parts: analysis of dynamic parameters and risk calculation.

3.2 Task list: ROS 2 Behaviour Trees Integration

After the Evaluator block diagnoses the task's safety and deems it unsafe, another block will communicate a new task to the control system to enhance availability. This will be accomplished using ROS 2 behaviour trees, a set of tools integrated into the robotic system to define its behaviour and schedule the tasks to achieve objectives. Behaviour trees offer a hierarchical structure for organizing robot actions logically, simplifying complex and adaptive behaviours. They include nodes representing actions like moving, avoiding obstacles, or interacting with objects, dynamically adapting to environmental conditions and goals for effective responses [19].

4 Conclusion

This work aims to improve robotic system availability while ensuring safety through software design. By evaluating the environment in real-time, assessing risk associated with robot actions, and integrating with ROS 2, it enhances decision-making and defines robot tasks, ultimately addressing safety concerns and improving availability in collaborative robotics.

Acknowledgment

This research has been funded by the Basque Government through the ELKARTEK programme within the framework of the AUTOTRUST project (grant number KK-2023/00019) and by the CERVERA programme within the framework of the MEDUSA project (grant number CER-20231011).

References

- [1] IBM, "What is Industry 4.0 and how does it work?," Accessed: March 26, 2024.
- [2] D. Rodriguez-Guerrera, G. Sorrosal, I. Cabanes, and C. Calleja, "Human-robot interaction review: Challenges and solutions for modern industrial environments," *IEEE User Journal*, 2021.
- [3] J. Guiochet, "Safety-critical advanced robots: A survey," *Robotics and Autonomous Systems*, 2017.
- [4] A. Giallanza, G. La Scalia, R. Micale, and C. M. La Fata, "Occupational health and safety issues in human-robot collaboration: state of the art and open challenges," *Safety science*, vol. 169, p. 106313, 2024.
- [5] Z. Jiang, B. Jin, and Y. Song, "A novel pet trajectory prediction method for intelligent plant cultivation robot," *IEEE Sensors Letters*, vol. 7, no. 2, pp. 1–4, 2023.
- [6] D. Bassermann, "Asam-standarization for automotive development." [Accessed 20-11-2023].
- [7] J. Saenz, I. Fassi, G. B. Prange-Lasonder, M. Valori, C. Bidard, A. B. Lassen, and J. Bessler-Etten, "Covr toolkit—supporting safety of interactive robotics applications," in *2021 IEEE 2nd International Conference on Human-Machine Systems (ICHMS)*, pp. 1–6, IEEE, 2021.
- [8] J. Saenz, R. Behrens, E. Schulenburg, H. Petersen, O. Gibaru, P. Neto, and N. Elkmann, "Methods for considering safety in design of robotics applications featuring human-robot collaboration," *The International Journal of Advanced Manufacturing Technology*, vol. 107, pp. 2313–2331, 2020.
- [9] P. Chemweno, L. Pintelon, and W. Decre, "Orienting safety assurance with outcomes of hazard analysis and risk assessment: A review of the iso 15066 standard for collaborative robot systems," *Safety Science*, vol. 129, p. 104832, 2020.
- [10] <https://www.iso.org/standard/51330.html>, "iso 10218-1.org." [Accessed 17-11-2023].
- [11] <https://www.iso.org/standard/41571.html>, "iso 10218-2.org." [Accessed 17-11-2023].
- [12] <https://www.iso.org/standard/62996.html>, "iso 15066.org." [Accessed 17-11-2023].
- [13] W. D. Peter Chemweno, Liliane Pintelon, "Orienting safety assurance with outcomes of hazard analysis and risk assessment: A review of the iso 15066 standard for collaborative robot systems," 2020.
- [14] A. Pupa, M. Minelli, and C. Secchi, "A dynamic planner for safe and predictable human-robot collaboration," *IEEE Robotics and Automation Letters*, vol. 9, no. 1, pp. 507–514, 2023.
- [15] <https://www.iso.org/standard/72140.html>, "Iec 31010:2019 iso.org.", 2019. [Accessed 14-12-2023].
- [16] "Github - ros 2 safety-critical working group." https://github.com/ros-safety/safety_working_group. [Accessed 19-12-2023].
- [17] M. P. H. et al., "Towards safety level definition for industrial robots in collaborative activities," *Procedia Manufacturing*, vol. 38, pp. 1481–1490, 2019.
- [18] <https://www.iso.org/standard/51528.html>, "iso 12100.org." [Accessed 17-11-2023].
- [19] J. A. Segura-Muros and J. Fernández-Olivares, "Integration of an automated hierarchical task planner in ros using behaviour trees," in *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 20–25, 2017.

Reconfigurable and Scalable Honeynet for Cyber-physical Systems

Luís Sousa, José Cecílio, Pedro M. Ferreira, Alan Oliveira de Sá*

LASIGE, Faculdade de Ciências da Universidade de Lisboa, Portugal; email: fc60428@alunos.fc.ul.pt,{jmcecilio, pmf, aodsa}@ciencias.ulisboa.pt

Abstract

Industrial Control Systems (ICS) constitute the backbone of contemporary industrial operations, ranging from modest heating, ventilation, and air conditioning systems to expansive national power grids. Given their pivotal role in critical infrastructure, there has been a concerted effort to enhance security measures and deepen our comprehension of potential cyber threats within this domain. To address these challenges, numerous implementations of Honeypots and Honeynets intended to detect and understand attacks have been employed for ICS. This approach diverges from conventional methods by focusing on making a scalable and reconfigurable honeynet for cyber-physical systems. It will also automatically generate attacks on the honeynet to test and validate it. With the development of a scalable and reconfigurable Honeynet and automatic attack generation tools, it is also expected that the system will serve as a basis for producing datasets for training algorithms for detecting and classifying attacks in cyber-physical honeynets.

Keywords: Industrial Control Systems, Cyber-Physical Systems, Honeypot/Honeynet, Dataset, Attack Capturing.

1 Introduction

Honeypots serve as tools designed to mislead attackers by creating an illusion that they are targeting genuine infrastructure. In reality, these systems gather valuable information about the ongoing attack, enabling the enhancement of both the actual infrastructure and the honeypot. Additionally, they provide insights into the attackers' intentions and the origin of the attack [1]. Honeynets are multiple Honeypots working together in the same system [1] which is the intended implementation of this work. A system like a CPS is typically considered a honeynet because each component can be considered a honeypot.

Honeynets are important for Industrial Control Systems (ICS), as the impact of a cyber-attack on these systems can cause significant harm to countries and society. Examples of some of the most prolific attacks on CPS include the Stuxnet [2], which attacked the nuclear enrichment facilities of Iran, or the 2015 attacks on the Ukrainian power grid [3]. Note that

the difficulty of also identifying the culprits and plausible deniability of the attacks makes it attractive for governments to influence foreign opinion and cause unrest without many consequences [4].

A ICS is a type of Cyber-physical System (CPS) that is typically composed of actuators and sensors that interact with the physical world (called the Plant), the control is composed of Programmable Logic Controllers (PLC) or Remote Terminal Units (RTU) that control the Plant and Monitoring typically done through a Human Machine Interface (HMI). This system is typically called a Supervisory Control and Data Acquisition (SCADA). To develop a CPS honeynet it is necessary to simulate all the components of a typical CPS so that it provides realistic data to fool the attacker into thinking it is a real system. For this, the use of real-time simulations to provide realistic data and information about a physical system is required. For that, there needs to be a complete simulation of the entire system from monitoring to the physical world.

In this context, this work aims to implement a software-based scalable and reconfigurable honeynet. This means that it should be able to add components and reconfigure the existing components. This work also focuses on the generation of dynamic and orchestrated attacks in the honeynet. The attack generation capability is used to validate the system. For this, the work includes the creation of a component called the Attack coordinator, which using attack modelling algorithms can attack the CPS dynamically. After having a good implementation of the honeynet and the attack generator, an additional goal is to use this setup to produce a data set for Machine Learning (ML)-based intrusion detection systems (IDS) in cyber-physical honeynets.

With this, the main objectives of this work are:

1. Develop a honeynet capable of being scalable and reconfigurable.
2. Produce an automatic attack generation tool to validate and test the honeynet.
3. Using the results of objectives 1 and 2 create datasets for ML-based IDSs in cyber-physical honeynets.

The rest of this work is organized as follows: Section 2 shows some of the work related to the topics of honeynets. Section 3 describes the proposed system design. In Section 4 discusses the results achieved so far and the work that has to be done. Section 5 brings the conclusions.

*Corresponding author.

Work	Year	Scalability	LoI	Attack Simulation	Physical Interaction
Hilt et al. [5]	2020	None	High	-	Real Hardware
HoneyVP [6]	2021	Limited	High	-	PLC level
HoneyPLC [7]	2020	Good	Medium	-	None
Pliatsios et al. [8]	2019	Limited	Medium	-	Recorded data
ICSpot [9]	2022	Good	Medium	-	MiniCPS [10]
PLCHoney [11]	2023	Limited	High	Data injection	Matlab Simulation
MimePot [12]	2019	Limited	High	Integrity Attack	Python Simulation

Table 1: Table representing the honeypots analyzed (LoI: Level of Interaction)

2 Related Work

The related works focus on similar implementations of honeynets to the honeynet this work is going to implement.

Hilt et al. [5] goes into detail on the implementation of a honeynet using real hardware in a real factory. Hilt's implementation focuses on having a realistic system so that an attacker will be fooled, but it also shows the costs that are required for a honeynet with real hardware, and that's why this work implementation diverges from having real hardware and instead uses a simulated environment.

HoneyVP [6] is a honeypot that uses a real PLC and routes the attack traffic to it. This approach is capable of handling multiple requests and attacks simultaneously by sending them to the same PLC. It uses Memoization [13] to reduce the required interactions with the PLC, this implementation reduces the cost compared to a traditional full hardware implementation but doesn't offer much realism when looking at the complete environment of a CPS as it only targets the PLC.

HoneyPLC [7] is a honeynet for simulating PLCs. It starts with real physical PLCs and tries to copy all relevant information from the PLC. After this HoneyPLC can simulate the different PLCs that it reads allowing for scalability and diversity in what an attacker sees. This work limits its scope to the PLCs thus still lacking when it comes to the other components of a CPS.

The study [8] (Pliatsios et al.) specifies a procedure to create a simple honeynet from an already existing CPS. It uses Conpot [14] a honeypot framework and uses real traffic from an RTU with two sensors. Afterwards, Conpot uses that recorded traffic to generate fake traffic by replaying it. From the attacker's perspective, there seems to be real traffic thus looking like a real system.

In [9] the authors build on top of HoneyPLC to add a physical component. In this article, they add MiniCPS [10] which is a CPS simulation. The use of a physical simulation improves the realism of the system compared to a system without it.

The work presented in [11] creates a honeypot called PLCHoney. It uses OpenPLC [15] and uses data from a recorded Simulink simulation to generate the Plant to PLC traffic. This implementation reduces the cost of running the simulation but by only using the data from the simulation the environment the honeypot tries to simulate is less dynamic.

MimePot is the honeypot conceived in [12]. It uses a real-time simulation for the Plant that feeds data to an HMI to form its CPS. This CPS is simple as it only includes a PLC and an

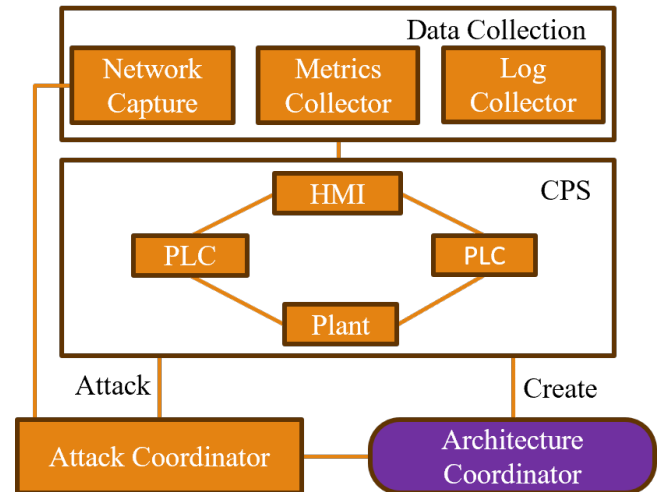


Figure 1: Architecture

HMI but is similar to the work we intend to make by having a real-time simulation for the plant.

The implementation that this work aims to achieve diverges from the other by having good scalability, a high level of interaction, attack simulation and a simulation for the physical interactions. Some of the works focus on different aspects similar to this work from the scalability or simulation of the plant, but they either only focus on a single component of a CPS or they don't achieve real scalability. This work will also focus on the attack simulation that PLCHoney and MimePot have but will try to have multiple attacks working together instead of just one.

3 System Design

Figure 1 shows the architecture of the system. It's divided into four modules: Coordinators, Attacks, CPS and Data Collection. The Coordinators are responsible for managing the system. This module is composed of the Architecture Coordinator and the Attack Coordinator. They communicate among themselves so that when the Architecture Coordinator generates the structure of the system the Attack Coordinator knows it as well.

There is the CPS composed of the elements that the attacker will have access to. They are a mix of HMIs, PLCs and Plants. The number of components in this group can vary as the architecture is scalable and controlled by the Architecture Coordinator. They are networked between them in a realistic way that is meant to be similar to implementations of CPS in the industry.

Then there is the Data Collection module which is composed of the tools necessary to collect metrics, logs and network

data. It will interact with all the other components so they can garner the information necessary to validate and later generate the dataset from that data.

In the implementation, all the components will be docker [16] containers as this will help isolate every running component and make their implementation dynamic to expand and replace components. This will also help the implementation of attacks, as attacks can be docker components that initialize in runtime.

3.1 Architecture Coordinator

The Architecture Coordinator will only run in the setup phase so it can generate the necessary extra files to configure each component on the architecture and provide the required architectural information to the Attack Coordinator. With this information, the Attack Coordinator can orchestrate attacks that make sense on the instantiated network.

3.2 Attack Coordinator

The Attack Coordinator will use the architecture of the system to plan and organize attacks, this should be done dynamically for any structure of the system. This helps improve the diversity of attacks that are performed. Having diverse attacks will lead to a higher level of confidence when validating this solution. With the improved variability, we also have a more diverse and complete input to feed into a dataset.

To create realistic attacks, the attack coordinator needs to have a good idea of the architecture of the system and for this, it needs to keep track of all the useful system information. The information of the network is in a graph containing the information of each service running in each component and the possible corresponding attacks for each service. The attacks will depend on the architecture that was generated by the Architecture Coordinator. The realism of the attacks will emerge from a combination between the network graph and a set of cataloged attack graphs that represent existing (and well-known) attack chains.

The coordination of multiple attacks will require information about what each attack does and what changes they do to the system. This is important to make a good attack sequence where the attacks need to make changes to the system to achieve the attacker's goal.

The attacks were chosen from [17] which specifies common attacks in CPSs. From there, the following attacks were chosen to be implemented based on how common and the complexity of implementation:

- Man-in-the-middle
- Modbus-Register Reading
- Denial of service
- Modbus-Register spoofing
- Replay attack

More attacks can be added as we progress with the implementation.

3.3 CPS

This section describes the components of the CPS module. The Plant is described in the Section 3.3.1, the SCADA/HMI in Section 3.3.2, and the PLC in Section 3.3.3.

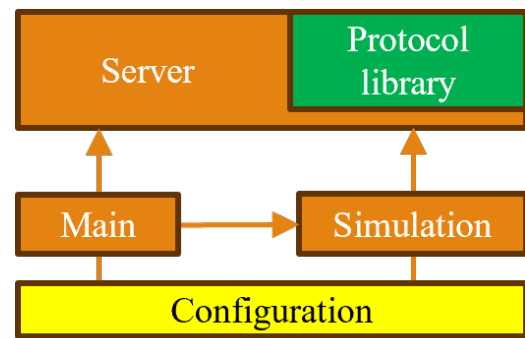


Figure 2: Plant

3.3.1 Plant

The Plant is implemented as a container simulating the physical process of the system and interacts and communicates with the PLCs using different protocols – currently implemented is the Modbus/TCP. The details of the software-based Plant simulator are presented in Figure 2, which represents an overview of how it works. A simulation feeds the data to the server and can be configured with different parameters from a configuration file. The aim is to include multiple simulations and be able to choose between the different simulations with the configuration file. Currently, the implemented simulation is the heating up of a gas container following Boyle's law.

3.3.2 SCADA/HMI

The SCADA is implemented with an open-source solution. The one used is FUXA which is a simple HMI that is easy to configure and use.

This solution also allows some further expansion. By allowing more protocols that can add additional attacks making a more diverse and realistic environment, the diverse protocols also serve to create background network traffic.

3.3.3 PLC

For the PLC we used OpenPLC which is also an open-source implementation of a PLC that implements our used protocols. In this case, it can use Modbus to communicate with the Plant and with the SCADA.

This solution will allow us to control the Plant with a realistic control structure. The code implemented on the PLC for the simulation follows the simple instructions of heating the container if the pressure and the temperature are below those specified on the HMI.

3.4 Data collection

To collect the data, the network traffic of all the components running on containers will be collected and run through a pre-processing step that will add information from the type of data in the packet.

The network capture will be implemented to allow the Attack Coordinator to inform which packets are attacks, and add information to the traffic about what is passing through the network. Beyond the data about the attacks, there will also be some of the data added about the kinds of protocols being used as can be seen in Figure 3.

There will be data collected beyond the network traffic. The system will also collect the metrics of the running system, from the number of network connections to processes running on the containers. The collection of logs will also be

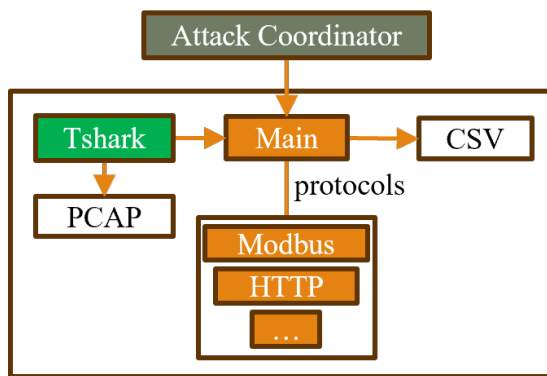


Figure 3: Network Capture

important and can be useful in feeding into the dataset to train a machine-learning algorithm, to fulfil Objective 3.

4 Current Status and Forthcoming Work

Currently, the Plant, PLC and HMI are implemented in the CPS thus being able to run a simple CPS. The initial work in the Architecture Coordinator and Network Capture is being done. After the completion of those two components, the focus will be on the Attack Coordinator to start the realization of attacks on the system to validate and generate the data set. The work will be expanded with more features, and will be made available (open source) as a tool to produce datasets of cybeattacks in ICSs.

5 Conclusion

This work aims to improve the current state of the art in honeynets by creating a honeynet that can fit any ICS, it also tries to provide better training data to create better IDSs so they can be integrated with machine learning and artificial intelligence by providing controlled environment to generate and execute attacks on a system that can follow the structure of a real CPS.

6 Acknowledgement

This work was supported by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020 (<https://doi.org/10.54499/UIDB/00408/2020>) and ref. UIDP/00408/2020 (<https://doi.org/10.54499/UIDP/00408/2020>).

References

- [1] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, "A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2351–2383, 2021.
- [2] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.
- [3] G. Liang, S. R. Weller, J. Zhao, F. Luo, and Z. Y. Dong, "The 2015 ukraine blackout: Implications for false data injection attacks," *IEEE Transactions on Power Systems*, vol. 32, no. 4, pp. 3317–3318, 2017.
- [4] J. K. Canfil, "The illogic of plausible deniability: why proxy conflict in cyberspace may no longer pay," *Journal of Cybersecurity*, vol. 8, p. tyac007, 09 2022.
- [5] S. Hilt, F. Maggi, C. Perine, L. Remorin, M. Rösler, and R. Vosseler, "Caught in the act: Running a realistic factory honeypot to capture real threats," tech. rep., Trend Micro, Inc.
- [6] J. You, S. Lv, Y. Sun, H. Wen, and L. Sun, "Honeyvp: A cost-effective hybrid honeypot architecture for industrial control systems," in *ICC 2021 - IEEE International Conference on Communications*, pp. 1–6, 2021.
- [7] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G.-J. Ahn, "Honeyplc: A next-generation honeypot for industrial control systems," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20*, (New York, NY, USA), p. 279–291, Association for Computing Machinery, 2020.
- [8] D. Pliatsios, P. Sarigiannidis, T. Liatifis, K. Rompolos, and I. Siniosoglou, "A novel and interactive industrial control system honeypot for critical smart grid infrastructure," in *2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pp. 1–6, 2019.
- [9] M. Conti, F. Trolese, and F. Turrin, "Icspot: A high-interaction honeypot for industrial control systems," in *2022 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–4, 2022.
- [10] D. Antonioli and N. O. Tippenhauer, "Minicps: A toolkit for security research on cps networks," in *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or Privacy, CPS-SPC '15*, (New York, NY, USA), p. 91–100, Association for Computing Machinery, 2015.
- [11] S. Y. Chowdhury, B. Dudley, and R. Sun, "The case for virtual plc-enabled honeypot design," in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 351–357, 2023.
- [12] G. Bernieri, M. Conti, and F. Pascucci, "Mimepot: a model-based honeypot for industrial control networks," in *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 433–438, 2019.
- [13] D. Michie, "'memo' functions and machine learning," *Nature*, vol. 218, pp. 19–22, Apr. 1968.
- [14] L. Rist, "Conpot." <http://conpot.org/>. Accessed: 2024-01-5.
- [15] "Openplc." <https://autonomylogic.com/>. Accessed: 2023-12-9.
- [16] "Docker." <https://www.docker.com/get-started/>. Accessed: 2023-12-8.
- [17] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-physical systems security—a survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.

Software-Based Security Framework for Edge and Mobile IoT

José Cecílio, Alan Oliveira de Sá, André Souto

LASIGE, Faculdade de Ciências da Universidade Lisboa, Portugal; email: {jmcecilio, aodsa, ansouto}@ciencias.ulisboa.pt

Abstract

With the proliferation of Internet of Things (IoT) devices, ensuring secure communications has become imperative. Due to their low cost and embedded nature, many of these devices operate with computational and energy constraints, neglecting the potential security vulnerabilities that they may bring. This work-in-progress is focused on designing secure communication among remote servers and embedded IoT devices to balance security robustness and energy efficiency. The proposed approach uses lightweight cryptography, optimizing device performance and security without overburdening their limited resources. Our architecture stands out for integrating Edge servers and a central Name Server, allowing secure and decentralized authentication and efficient connection transitions between different Edge servers. This architecture enhances the scalability of the IoT network and reduces the load on each server, distributing the responsibility for authentication and key management.

Keywords: *IoT Security, Edge, Device Protection, Data reliability.*

1 Introduction

Given the increasing ubiquity of Internet of Things (IoT) devices in our daily lives, understanding the variety and complexity of cyberattacks these devices face has become imperative. Johnston, in [1], highlights a broad spectrum of vulnerabilities in IoT devices, leading to a large range of attack techniques that compromise data integrity and confidentiality, significantly threatening IoT systems' functionality and security.

One of the primary challenges in the IoT landscape is balancing the affordability of devices with the implementation of robust security measures. For instance, jamming and adversarial attacks, which aim to disrupt wireless communication, are particularly disruptive in resource-limited devices [2]. Measures such as signal strength analysis and packet delivery rate verification have been proposed as effective solutions to detect such activities [3]. Additionally, the insecure setup of IoT devices exposes them to various vulnerabilities. Researchers have suggested introducing artificial noise and robust authentication processes to secure devices during this critical phase [4]. Similarly, for low-level Sybil and spoofing

attacks, channel-based detection and RSS consistency emerge as fundamental preventive measures [5].

Insecure physical interfaces also represent a considerable risk. Strategies for preventing physical tampering and secure access service management are essential to mitigate these risks [6]. For sleep deprivation attacks, which aim to deplete devices' energy, cluster-based techniques and anomaly detection offer a potential solution [7]. Fragmentation in IoT devices creates additional vulnerabilities, where replay attacks can be particularly damaging. Content chaining schemes and split buffer approaches are presented as viable solutions [8]. Within the scope of IoT attacks, there is a need for a deeper analysis of middleware security. Robust authentication, access control, and secure protocols like HTTPS and XMPP are proposed measures to bolster security at this layer [9].

Given the wide range of potential attacks and solutions, it is clear that IoT device security is continuously changing, requiring a comprehensive approach that considers effectiveness, cost, and practicality.

This work-in-progress delves into the intricate challenge of having a balanced approach to security while maintaining operational efficiency and safeguarding against potential threats. We propose an architectural framework for secure IoT communications designed to address the emerging challenges of the mobile Internet of Things (MIoT). This architecture is focused on enhancing security through decentralized authentication, which is achieved by integrating a combination of Edge servers and a central Name Server, aiming to provide robust and decentralized authentication mechanisms across the network.

This work extends the software-based security approach proposed in [10] by adding new features for mobile IoT devices and multiple Edge servers, allowing mobile-embedded IoT devices to change their Edge server. Considering the typical constraints of IoT devices, our architecture optimizes resource and energy consumption using a strategic cache system implemented at each Edge server. This minimizes the need for repetitive and resource-intensive re-authentication processes, thereby conserving energy. Scalability and flexibility is another feature of the proposed architecture. The design is intended to be highly scalable, easily accommodating an increasing number of IoT devices without compromising performance. Central to this scalability is the architecture's flexibility in seamlessly integrating new Edge

servers into the network and allowing devices to switch connections between these servers without friction. Balancing security with resource usage (efficiency) and energy consumption is vital. The architecture is tailored to balance robust security measures and efficient performance, particularly on resource-constrained IoT devices. Given the IoT context of the proposed solution, the encryption mechanism must be lightweight, ensuring minimal resource consumption while preserving data privacy. In alignment with this requirement, the encryption strategy leverages NIST's lightweight encryption standards¹ to handle data encryption and integrity.

2 Threat Model and Assumptions

Developing a threat model and establishing assumptions are crucial to designing a secure architecture. Next, we outline the threat model considered in the design of our architecture, as well as the adopted countermeasures:

- **Unauthorized Access:** Malicious actors are attempting to access IoT devices or communication channels illegally. This threat is addressed by considering robust decentralized authentication through Edge servers and a central Name Server, leveraging secure protocols.
- **Data Tampering:** Attackers try to manipulate or tamper with data during communication. The proposed architecture uses lightweight encryption standards following NIST guidelines for data encryption and integrity.
- **Physical Tampering:** Physical access to IoT devices for tampering or unauthorized manipulation. Secure access service management and regular code integrity verification are applied.
- **Jamming and Adversarial Attacks:** Deliberate disruptions to wireless communication to compromise IoT devices. Signal strength analysis and packet delivery rate verification are used to detect and counter jamming and adversarial attacks.
- **Replay Attacks:** Replay attacks exploiting fragmentation vulnerabilities in IoT devices. Challenge approach, data integrity, and application integrity verifications are used to prevent and detect replay attacks.

Considering this threat model, the following assumptions are made:

- **Communication Channels:** The architecture assumes the availability of reliable communication channels, and the threat model focuses on securing the data transmitted over these channels.
- **Adequate Resource Allocation:** The proposed architecture assumes that sufficient resources, such as bandwidth and processing power, are allocated to Edge servers to handle authentication and encryption processes effectively.
- **Cooperation Among Edge Servers:** The architecture assumes a cooperative environment among Edge servers to facilitate seamless authentication and communication between devices switching connections.

¹<https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices>

These threat scenarios and assumptions provide a basis for evaluating the security features of the proposed architecture and guide the implementation of relevant security measures.

3 Edge and IoT mobile management architecture

In [10], we introduced software-driven protection and encryption mechanisms tailored for embedded devices. Our design incorporates an Agent specifically designed to operate seamlessly with low-cost and low-end devices, eliminating the need for any alterations to the underlying hardware. Complementing this, we introduce a Computing Module designed for devices with slightly greater computational capabilities. The Computing Module empowers devices to write data in secure memory, ensuring ongoing verification of its integrity for sustained protection. Moreover, it leverages the Agents present on the device to strengthen device applications against potential attacks. This is achieved by instructing the Agent to generate a code signature for the application and subsequently validating it, enhancing the overall security posture of the embedded system.

This work extends the previous software-based security approach for mobile IoT devices and multiple Edge servers, allowing mobile-embedded IoT devices to change their Edge server. It aims to balance robust security measures and energy efficiency harmoniously, avoiding new authentication procedures every time a new device changes the Edge server. Our proposed methodology uses lightweight cryptography, strategically optimizing device performance and ensuring security without imposing undue strain on their constrained resources. The architectural framework incorporates Edge servers and a central Name Server, fostering secure and decentralized authentication processes.

This integration also provides seamless and efficient transitions between different Edge servers, which allows to increase the number of nodes. Our architecture reduces the burden on individual servers by distributing authentication and key management responsibilities, leading to a more robust and resource-efficient system.

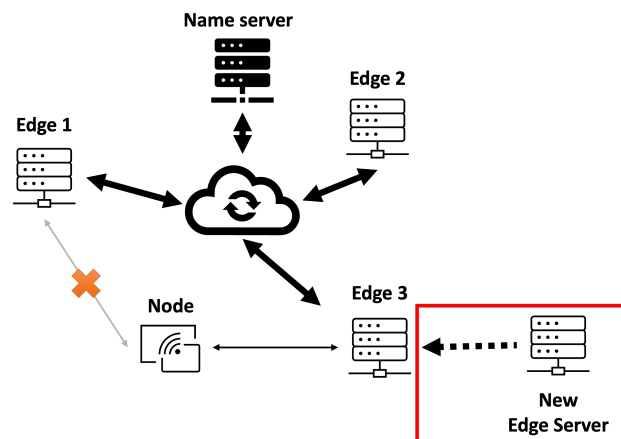


Figure 1: General architecture for Edge and IoT mobile management service.

Figure 1 shows the proposed architecture where we have envisioned a dynamic and mobile network of Nodes capable of disconnecting from one server and seamlessly connecting to another. To facilitate this mobility without needing re-authentication each time a Node switches servers, we propose a system where the authentication credentials, specifically the key and the hashed application data, are transferred alongside the Node to the new server. To support this seamless transfer, each edge in the network will maintain a secure cache to store authentication information and keys associated with each Node. These keys are subject to renewal, triggered by the Edge servers or when a specific time interval has lapsed, ensuring that authentication remains updated and secure.

Moreover, the key renewal protocol is designed to respect the mobility of the Nodes. If a Node transitions to a new Edge server, the key migrates, and subsequent key renewals are exclusively negotiated with the new server. Upon recognizing that a key is due for renewal and is marked as shared, the initial server will remove that key and associated Node from its registry, effectively transferring full authentication responsibility to the new server.

The system is also designed to achieve high scalability, with provisions for new Edge servers to integrate into the existing system using the protocol proposed in Rosa et al. [11]. When a Node wishing to join sends an encrypted authentication message to nearby authenticated Nodes (any Edge server). This message includes a unique identifier and a random key from a pre-established set, ensuring the Node's authenticity from the outset. Upon receipt, an authenticated Node decrypts the message using a default key. If the random key is recognized, a challenge is issued to the new Node to confirm its identity. It uses encryption functions with a mix of the Node's characteristics and randomly generated values. Only upon a successful response to this challenge the process proceed. This authentication is not isolated. It is part of a broader protocol that seamlessly adds new servers to the network.

When a new Edge server joins, it undergoes a similar authentication process and must be registered with the name server by the authenticating server. This ensures that all entities within the network, whether Nodes or servers, are authenticated and authorized, maintaining a secure network environment.

A Server's authentication message includes a random key and the Server's ID, encrypted with a default key known as DK, pre-loaded by an administrator. When an existing authenticated Server receives this message, it decrypts and verifies the random key against its own set. A failure to recognize the key results in the Server's ID being placed on a denylist. However, if the key is validated, the authenticated Server issues a cryptographic challenge to the new Server. The challenge and subsequent response ensure that only Servers with the correct credentials can join the network, preventing unauthorized accesses.

Once authenticated, Nodes can freely move within the network, connecting to different servers without needing to re-authenticate, as their credentials and keys are securely transferred between servers. This procedure aligns with the earlier

detailed method for authenticating new servers within the network. It emphasizes the reuse of established authentication, minimizing overhead and enhancing network fluidity.

When a new Edge server joins and is authenticated, the authenticating Server is responsible for registering the newcomer in the Name Server.

The Name Server holds essential metadata about the network. This includes mapping addresses to names, the physical locations of Edge servers –potentially using GPS coordinates – the resources available on each Edge server, and records of which entities have authenticated which.

Each Edge Server and Node implements the software-based security solution outlined in [10], ensuring that each component contributes to the network's security posture.

When a Node moves from the coverage of one server to another, it relies on the initial server to guide its connection to the optimal Edge server. After consulting the metadata from the name server, the initial server directs the Node to the most suitable server based on a set of criteria derived from the available metadata. This decision-making process is informed by a comprehensive understanding of the network's topology, resources, and authentication relationships, which are meticulously maintained and updated in the Name Server's database.

4 Device-Edge Communication

The proposed architecture defines an innovative protocol for handling Edge servers within the network, allowing it to shift from the traditional central Gateway model to a more flexible and scalable system with multiple Edge servers and a Name Server.

In this system, the authentication of a new Edge server is conducted by an already authenticated Edge server, thus transferring the computational burden and responsibility of the authentication process from a central server to the Edge servers themselves. This decentralized methodology allows each Edge server to authenticate new network participants, simplifying network management and reducing the load on a single central point.

The authentication process begins when an authenticated Edge server sends detailed information about a new Edge server to the Name Server following a successful challenge process. The Name Server then maintains a comprehensive registry of Edge servers in the network, including their addresses, names, locations, resources, and authentication history.

Figure 2 illustrates the message exchange process when a node disconnects from a previous Edge server and seeks to connect to another Edge server recommended by the initial one. This connection transition in the proposed architecture is conducted with a particular focus on resource and energy efficiency, thus avoiding unnecessary consumption of both energy and computational resources.

To ensure this efficiency, the previous Edge Server passes the encrypted key to the new Edge server, which was used to

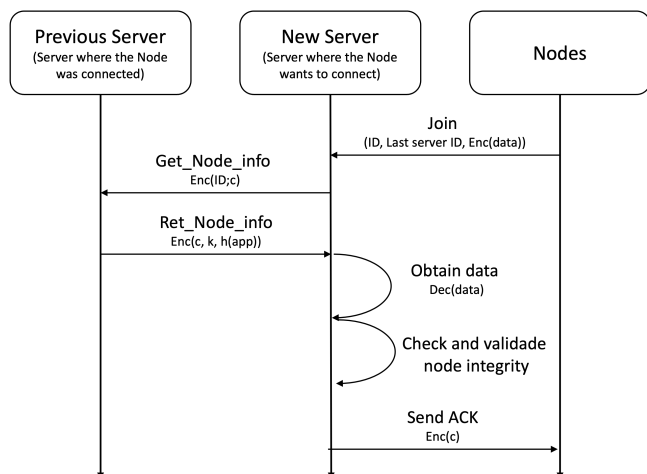


Figure 2: Overview of node connection with a new edge.

establish the initial connection with the Node. It also provides the encrypted hash of the application. As a result, the new Edge Server that will now establish a new connection with the Node does not need to expend energy and computational resources to create a new shared key for authentication.

Upon receiving the key and the application's hash, the new server will decrypt the data sent by the Node and understand its application's hash, which is then compared with the Initial Server's. If the hashes coincide, it indicates that the data integrity has been maintained, and therefore, the Edge server and the Node can securely establish the connection.

This process not only ensures the security and integrity of the data during connection transitions between Edge servers and nodes but also optimizes the use of resources and energy, a relevant aspect in IoT networks, especially for devices with limited capabilities.

5 Conclusion

This work-in-progress is focused on designing secure communication among remote servers and mobile-embedded IoT devices to balance security robustness and energy efficiency. Our focus on addressing several threat vectors led us to propose a comprehensive architectural framework. The integration of Edge servers and a central Name Server, coupled with the deployment of lightweight cryptography, allows for improved security while respecting resource-constrained IoT devices' limitations.

The strategic use of a secure cache system at each Edge server minimizes the need for resource-intensive re-authentication processes, contributing to energy preservation. The defined threat model allows to establish a robust security framework addressing unauthorized access, data tampering, physical tampering, and other potential threats. We balance security robustness, efficiency, and feasibility by adhering to NIST's lightweight encryption standards and enforcing device participation policies.

This work-in-progress represents a significant advance towards establishing a nuanced and balanced approach to IoT device security. By intertwining decentralized authentication,

lightweight cryptography, and thoughtful design principles, our proposed architecture sets the stage for a more secure, scalable, and energy-efficient future for the Mobile Internet of Things (MIoT).

Acknowledgments

This work was supported by FCT through the LASIGE Research Unit, ref. UIDB/00408/2020 (<https://doi.org/10.54499/UIDB/00408/2020>) and ref. UIDP/00408/2020 (<https://doi.org/10.54499/UIDP/00408/2020>).

References

- [1] N. Johnston, "A gentle introduction and an exploration of root causes," 2019.
- [2] Q. Dong, D. Liu, and M. Wright, "Mitigating jamming attacks in wireless broadcast systems," *Wireless networks*, vol. 19, pp. 1867–1880, 2013.
- [3] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," 2005.
- [4] S. H. Chae, W. Choi, J. H. Lee, and T. Q. Quek, "Enhanced secrecy in stochastic wireless networks: Artificial noise with secrecy protected zone," *IEEE Transactions on Information Forensics and Security*, vol. 9, pp. 1617–1628, 10 2014.
- [5] L. Xiao, L. J. Greenstein, N. B. Mandayam, and W. Trappe, "Channel-based detection of sybil attacks in wireless networks," *IEEE Transactions on Information Forensics and Security*, vol. 4, pp. 492–503, 9 2009.
- [6] S. K. Khatri, A. U. Dubai, A. U. D. A. D. of Engineering & Technology, I. of Electrical, E. E. U. A. E. Section, I. of Electrical, and E. Engineers, *2017 International Conference on Infocom Technologies and Unmanned Systems (ICTUS) (Trends and Future Directions): December 18-20, 2017*.
- [7] T. Bhattachali and R. Chaki, "A survey of recent intrusion detection systems for wireless sensor network," in *Advances in Network Security and Applications: 4th International Conference, CNSA 2011, Chennai, India, July 15-17, 2011*, pp. 268–280, Springer, 2011.
- [8] *WiSec '13: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, (New York, NY, USA), Association for Computing Machinery, 2013.
- [9] D. Conzon, T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M. A. Spirito, "The virtus middleware: An xmpp based architecture for secure iot communications," in *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–6, IEEE, 2012.
- [10] J. Ferreira, A. Oliveira, A. Souto, and J. Cecílio, "Software-based security approach for networked embedded devices," *Ada Lett.*, vol. 43, p. 73–77, oct 2023.

- [11] P. Rosa, A. Souto, and J. Cecílio, “Light-sae: A lightweight authentication protocol for large-scale iot environments made with constrained devices,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 2428–2441, 2023.

Supporting Ada in the ROSE Compiler

Peter Pirkelbauer, Chunhua Liao, Pei-Hung Lin, David Wright, Charles Reynolds, Daniel Quinlan

Lawrence Livermore National Laboratory, 7000 East Ave, Livermore, CA 94550, USA;

email: {pirkelbauer2, lin32, liao6, wright97, reynolds12, quinlan1}@llnl.gov

Abstract

Manual code maintenance of large code bases is tedious, time-consuming, and error-prone. To enable the engineering of source code maintenance tools for Ada, a mature infrastructure that provides capabilities for parsing, unparsing, semantic analysis, and transformations is needed.

This work discusses our progress of adding Ada support to ROSE, a mature source-to-source translation infrastructure. The paper discusses the design of ROSE, the extensions required for adding Ada, difficulties we encountered with processing existing code bases, and several prototype analysis and translation tools enabled by the new Ada support in ROSE.

Keywords: source-to-source translation; ROSE

1 Introduction

Manual code maintenance of large code bases is tedious, time-consuming, and error-prone. Tool support for the maintenance of large-code bases is needed. To be accepted by software engineers, such tools need to be able to soundly reason about the source code and produce output that is understandable and maintainable. With understandable and maintainable we mean that the output file needs to preserve much of the original code formatting and it needs to contain all elements that were part of the original code, including comments and preprocessor directives. To meet these requirements, tools need to operate on an intermediate representation (IR) that is similar to compilers. In addition tools need to preserve many details present in source code that compilers typically discard in early processing stages, such as comments and preprocessor directives.

To enable the development of such tools, this work adds Ada support to the ROSE source-to-source translation infrastructure [1]. We chose the ROSE framework, because we expect that existing analysis and transformation tools that are available in ROSE would be portable to Ada with minimal effort.

The contributions of this work are: (1) the extension of the ROSE infrastructure to support Ada 95, (2) an initial set of tools enabled by the Ada support in ROSE.

The remainder of the paper is organized as follows: §2 provides background on ROSE, §3 describes our extension to ROSE, §4 discusses a set of prototype tools, §5 compares this work to related work, and §6 offers a brief summary and outlook on future work.

2 Background - ROSE infrastructure

The ROSE source-to-source compiler infrastructure [1] offers unique capabilities for building software maintenance tools.

ROSE is a source-to-source compiler infrastructure that enables static analysis and transformation tools. ROSE is a mature open-source project that provides frontends and backends for many popular languages and parallel programming models, including C, C++, UPC, Fortran, OpenMP, and CUDA. ROSE represents program code at a high-level that is close to the source code. ROSE preserves comments, pragmas, and preprocessor directives so that code can be unparsed with all elements that were in the original source file. ROSE's goal is to produce source code that is understandable and maintainable by humans. This makes ROSE very suitable for building source code maintenance tools.

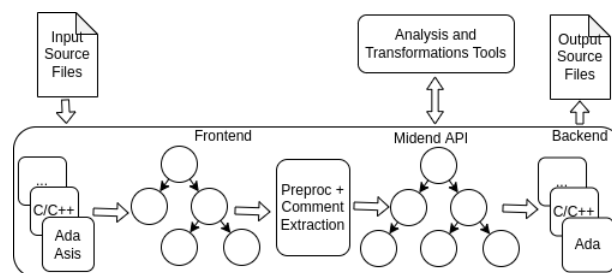


Fig. 1: The Rose source-to-source infrastructure

Fig. 1 shows ROSE's system architecture. To process input files, ROSE uses industrial strength compiler frontends. The frontend parses the input files and ROSE generates its intermediate representation (IR). Typically compiler frontends discard important information, such as preprocessor directives and comments. Thus, ROSE post-processes the input file to extract information that was discarded. In this phase, ROSE creates representations for comments and preprocessor directives, and attaches them to the IR based on source code location information.

ROSE's IR is designed as object-oriented class hierarchy, implemented in C++. Fig. 1 in §3 shows a small subset of classes that represent source code. At the root is a class `Node`, and derived from it are various subclasses representing different concepts, such as types, nodes with source code locations (i.e., `LocatedNode`), including expressions, statements, and declarations, and nodes for symbols representing declarations in symbol tables.

ROSE offers application programming interfaces (API) for the traversal and manipulation of its IR. Custom analysis and

transformation tools use the API to access and modify the program representation.

3 Implementation for Ada

We extend the ROSE source-to-source compiler infrastructure with support for the Ada programming language.

Frontend - Parser: For Ada, we follow the ROSE frontend workflow. First we use the Ada Semantic Interface Specification (ASIS) [2] implementation by AdaCore [3] to process Ada source files. Then, post-processing attaches comments and non-standard preprocessor directives¹. Currently, ROSE supports Ada 95 [4] and additional selected language features from more recent standards (e.g., aspects, if expressions, null procedures).

Like ROSE, ASIS represents the source code in the form of an abstract syntax tree (AST). In ASIS, each node has an ID. Edges in the AST are represented by using these IDs to link source to the target nodes. The translation traverses the ASIS AST and generates a ROSE AST from it. Declarations from ASIS are mapped to ROSE AST nodes through their IDs. The translation follows three guiding principles: (1) represent the Ada source code accurately; (2) preserve ROSE Intermediate Representation (IR) invariants; (3) reuse existing IR nodes, if the Ada concepts are close to already existing IR nodes.

The translation faced the following challenges:

Standard package: We were unable to extract the content of the Standard package from ASIS. Therefore, any ASIS node with references to Standard cannot be resolved (e.g., types, operators). To solve this problem, ROSE builds a representation for the Standard package internally. Names that the translator cannot resolve through ASIS IDs get looked up by name in the context, including Standard.

Scope qualification: ASIS preserves scope qualification in source code, for example the prefix `Ada.Text_IO` is preserved in `Ada.Text_IO.Put_Line("Hello")`. To simplify the development of transformation tools that move declarations from one scope to another, ROSE does not represent the prefix currently. Instead, the ROSE unparser uses a scope-qualification pass that computes a valid prefix based on the scope of the declaration and the scope of its use. The scope qualifier tracks `with` clauses, renaming declarations, and shadowing declarations to generate an accessible prefix.

Symbol overload resolution: Ada's derived types inherit primitive subprograms from the base type. While ASIS represents inherited subprograms, it does not use them at call sites. Instead, the call links to the original subprogram. Since the original subprogram and the inherited subprogram may reside in different scopes, scope-qualification based on the original subprogram declaration would generate an invalid scope prefix.

Consider the following example, where package A introduces a new type `Num` and a primitive operation `Init`. Package B creates a new type `Number` derived from `Num`. `Number` also inherits the primitive operations.

```
1 package A is
2     type Num is new Integer;
3     function Init return Num;
4 end A;
5
6 package B is
7     type Number is new A.Num;
8 end B;
```

and the initialization of some variable `val` of type `B.Number`.

```
9 val : B.Number := B.Init;
```

Here, in package B ASIS provides information about the inherited subprogram, whereas the callee in the call of `B.Init` links to the declaration in package A. In addition, ASIS preserves the package prefix `B`. ROSE, on the other hand, does not preserve the prefix in its IR, but represents the callee through the inherited subprogram symbol that was created in package B.

To resolve these calls correctly to an inherited subprogram symbol, ROSE post-processes the initial AST created from ASIS. For any expression that involves subprogram calls, a overload resolution pass generates all candidate callees, consisting of a subprogram symbol and its inherited subprogram symbols. Then, we use argument and return types derived from the context to identify the called target symbol. Since such calls can be subexpressions of other expressions, overload resolution iterates over the expression tree until no more ambiguity can be resolved.

The scope qualifier relies on the correct symbol to generate the correct prefix.

Operator declarations: Built-in operator declarations (e.g., `+`, `*`) for types declared in the Standard package are also declared in Standard. For types that derive from built-in types, we are unable to find the corresponding operator declarations. In such cases, ROSE generates implicit operator declarations and places them into the corresponding scope, usually the scope of the derived type declaration. This enables the correct generation of a subprogram's scope qualification prefix.

Uniform forward and defining declarations: ROSE requires forward declarations and defining declarations to use the same IR node type. This is not the case for ASIS, where a package may forward declare a type name and define it later, for example in the private part of a package. ROSE resolves this by using the type definition to inform what kind of IR node needs to be used for the forward declaration (e.g., record, enumeration, scalar).

Placement by source location information: To our knowledge, ASIS does not preserve the source location information of certain block structure related keywords, such as `declare`, `begin`, `exception`. This makes it hard to attach additional IR nodes for comments, pragmas, and preprocessor directives to the relevant AST nodes, because it is unclear whether the extra information should be part of the declarative section or the code section (the source location is after the last declaration, but before the first statement). Currently, pragmas get placed based on their kind, while comments and preprocessor nodes are attached to the last declaration (if available).

¹While the Ada language standard does not define preprocessing, many compilers support it.

Midend - Abstract Syntax Tree (AST): ROSE's IR represents the source code at a high-level in the form of an abstract syntax tree. Tools that are based on ROSE, will access and manipulate the IR.

The IR for representing source-level information across all languages comprises 684 classes (including Ada). Thus far 72 classes have been added to represent distinct features in Ada. Most of the new classes are related to the language's rich type system, generics, and concurrency.

The classes are organized in an object-oriented hierarchy. Fig. 2 shows the top levels of the hierarchy. At the root are `Node`, the foundational root class, and `LocatedNode`, a base class for all nodes that carry source location information. `Type`, `Statement`, `DeclarationStatement`, and `Expression` are common base classes for source-level information in source files. In addition, ROSE also maintains declaration symbols (`Symbol`) and symbol tables for scopes. The nodes with a blue bold frame are examples for classes that have been added to represent Ada.

A complete list of ROSE IR classes and their hierarchy is available at http://rosecompiler.org/ROSE_HTML_Reference/classSgNode.html.

Each class contains additional information and edges to other nodes as needed. This includes back links to types and symbols, which makes the program representation a directed graph.

Consider the following recursive implementation of a factorial function, `Fact`.

```
1 function Fact(n: Natural) return Natural is
2 begin
3   return (if n < 2 then 1 else n*Fact(n-1));
4 end Fact;
```

Fig. 3 shows the AST that ROSE generates for `Fact`. The top node is the global scope. Purple nodes underneath Global refer to declarations, statements, and expressions. Operators on the type `Natural` are represented as function calls to declarations in package `Standard`. The figure shows a forward declaration of `Fact` (called first nondefining declaration in ROSE) that is not part of the original tree, but that is required by ROSE. The ROSE IR is fully typed. The red edges indicate the type information attached to declaration and computed for expressions. Note, the type `LongLong` (located at the bottom) currently serves as the universal integer type from which other integer types are derived (e.g., `Natural`). Some nodes, such as unreferenced nodes in `Standard`, symbol tables, and symbols are omitted.

Backend: ROSE supports three ways to write out the AST or parts of the AST.

Standard unparsing generates code for an input file. Currently, the unparser has been developed to produce compilable code. Improving code formatting is work in progress.

Token-based unparsing uses the token sequence and token location in the original source code to print unmodified AST

subtrees. Only manipulated AST nodes are pretty-printed. This unparsing technique is not yet supported for Ada.

Patch generation uses the source location information in modified subtrees to generate patch files that can be applied using the patch utility in Unix².

Evaluation: We have used ROSE to process a range of Ada codes, including ACATS 2 to test for completeness. Our tests ingest source files and unparse the IR to source files. Then the produced files are compiled with GNAT and executed. ROSE+GNAT currently passes 2,151 of the ACATS 2 tests, while GNAT alone passes 2,170 successful tests. Most of the 19 failing tests are caused by the scope qualification pass producing an invalid scope prefix, by missing or misplaced pragmas, and by issues with querying the ASIS representation.

We have also tested on open-source software and other large code bases. Our tests involve compiling the original code with ROSE, generating output with the default unparser, and recompiling the output with GNAT.

Some projects were written for different architectures and compilers. Here, we noticed the following portability issues.

Source code portability: Some code that is accepted by some compilers is rejected by the GNAT/ASIS frontend due to compiler implementation choices. For example, different numeric base types enable code to compile on some systems that fail with GNAT/ASIS. The following code snippet illustrates the problem.

```
1 type Msp is range 0 .. 3;
2 b : Msp := 1;
3 x : Integer := Integer(b * 2**9); -- ERROR
```

Here, Line 3 fails to compile, because GNAT uses the range `-128..127` as `Msp`'s base type. Since the result of `2**9` is outside that range the compilation fails. Other compilers may choose a larger base type for `MSP`. This can be worked around by converting `b` into an `Integer` before the multiplication, `Integer(b) * 2**9`.

Another case we saw was that one compiler allows passing the same variable to multiple `out` parameters in the same function call, whereas GNAT flags that usage as an error. The reason for this is that GNAT community edition 2019 uses the Ada 2012 language standard, whereas the code base was written for Ada 95. Ada 2012 rejects the code [5, §4.2], while earlier compilers may have produced warnings against that coding practice, as the order of write backs to the aliased variable is unspecified.

To solve these portability issues and differences in language standards, we have manually rewritten the source code to make them compilable with GNAT.

Different Target Hardware and Architectures: Some of the code that we have used to evaluate our Ada support has been coded for different target architectures than the GNAT/ASIS frontend. One specific example is Ada code that has been

²[https://en.wikipedia.org/wiki/Patch_\(Unix\)](https://en.wikipedia.org/wiki/Patch_(Unix))

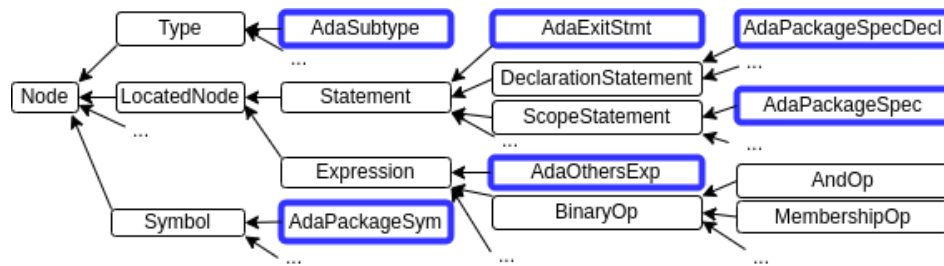


Fig. 2: ROSE AST class hierarchy, showing a subset of nodes that were used or added to support Ada.

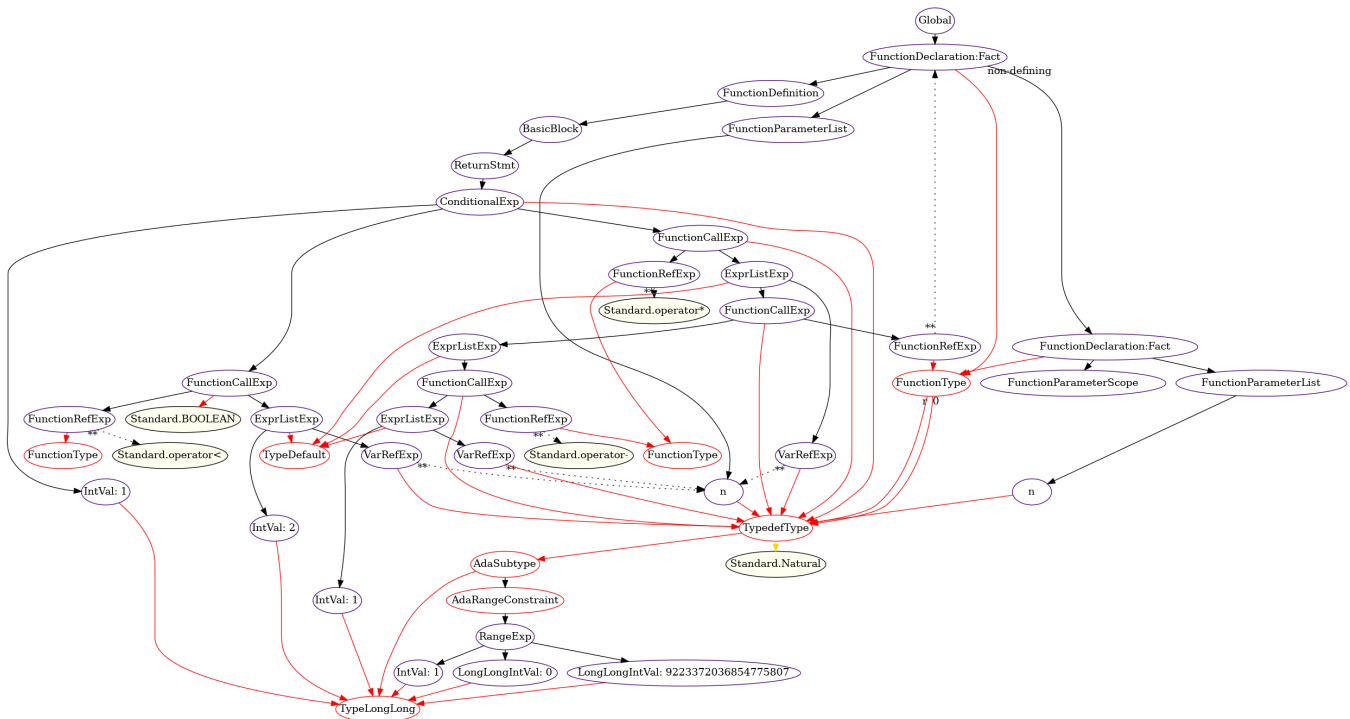


Fig. 3: Abstract Syntax Tree for Factorial

targeted for a 32-bit architecture, whereas the GNAT/ASIS frontend is running on a 64-bit architecture. That is only a problem when the evaluation software does not encapsulate architecture-specific code with this portability in mind.

Another issue we had to address when dealing with different target hardware was the different embedded assembly languages and support (ASM calls) with the GNAT/ASIS frontend. In addition to the actual different assembly statements, there were even some style differences that caused issues with the GNAT/ASIS frontend. Most of the embedded ASM from the sample Ada code was just commented out, or changed to NOP statements.

Preprocessor: Processing preprocessor directives poses unique challenges, similar to C++. Preprocessor directives may be used to maintain multiple configurations of a program. The problem is that the preprocessor constitutes its own language that modifies the source code before the compiler runs, yet the preservation of preprocessor directives is important for a truthful representation of the original source file. ROSE frontend post-processing adds the preprocessor directives, but the AST only represents code enabled by the preprocessor.

4 Applications

Several software maintenance tools were prototyped.

LCOM Code quality metrics: Code quality metrics play an important role in software engineering. They indicate code that may be difficult to understand, modify, and test. Tracking code quality metrics during code maintenance is particularly important. In contrast to a clean design, adding features or making changes to the code may make the code architecture deteriorate over time. Loss of Cohesion of Methods (LCOM) describe a set of metrics that assess the cohesion of object-oriented designs by analyzing how methods and data defined in a class are related. Our adaptation of the LCOM metrics for Ada is described separately [6].

Duplicate with clause removal Code refactoring is an important software development practice to improve the quality of source code, while still keeping its desired behavior. For some large scale Ada codes, duplicated `with` clauses may be introduced by developers as the codes evolve. While technically not harmful, extra `with` clauses clutter the code. Thus

it is desirable to identify and remove such duplicated with clauses in the source code.

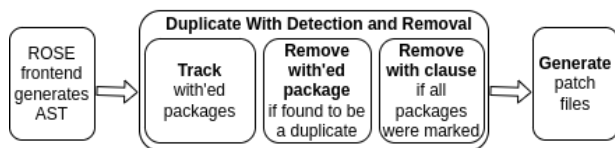


Fig. 4: Internal Components of refactorAda

Using ROSE, we have developed a source-to-source code refactoring tool named refactorAda (shown in Fig. 4). It currently supports the code transformation to remove duplicated `with` clauses. The tool works in the following steps:

1. The ROSE frontend creates an AST from Ada sources
2. *Track*: By traversing the AST, the tool finds all `with` clauses and stores package declarations included by `with` clauses into a dictionary.
3. *Remove with'd package*: For each new `with` clause, it checks if the included packages have been seen before. If so, the included package is marked for removal.
4. *Remove with clause*: If all packages in a `with` clause were marked, the entire clause is marked for removal.
5. The tool creates a patch file that specifies the lines to be modified or deleted to eliminate the duplicates.

Note that in Step 3, a unit referenced by a `with` clause can be a package, a function, or a package renaming declaration. The tool has to differentiate them. For multiple levels of renamed packages originating from the same package, the tool does not trace all the way to their root package. Instead, any packages with different names are treated as not duplicated, since the code may later refer to the package using the changed name.

We currently generate patch files to represent the transformation. The reason is that the default unparser for Ada does not preserve the original code indentation. Using patch files allows users to inspect changes to their codes before applying the patches.

Null exclusion: The semantics of anonymous access type parameters differs slightly between the Ada 95 and the Ada 05 standards. While the former standard implicitly excludes null values, the latter requires null exclusion to be explicitly specified [7, §3.2]. With Ada 95, passing null as argument fails at the caller. With Ada 05, the same code would fail in the callee, only if a value is accessed through the access-type parameter.

To facilitate code migration of Ada 95 code bases to newer Ada standards, we have used ROSE to prototype a migration tool that adds null exclusion to access type parameters. The tool traverses the ROSE IR and finds all function, procedures, and subprogram access type declarations that use anonymous access types. Then it adds the null exclusion specification as declaration modifier to the parameter declaration.

Translating Ada to C++: Some code teams might want to translate their legacy software written in Ada to C++ versions, since C++ has a large community of developers which makes it easier to hire trained software developers.

Given the largely shared AST among Ada and C++ in ROSE, it is convenient to build a cross-language translator using ROSE. We have explored ways of translating a subset of sequential Ada code to C++. Specifically, we focused on exploring ways to model the rich type system in Ada as domain specific C++ template libraries. Our prototype translator (named Ada2Cpp) works as follows:

1. Parsing Ada files to create an Ada Abstract Syntax Tree (AST) using ROSE's frontend.
2. Collecting type information, primarily array types, to facilitate later processing.
3. Conducting AST Normalization such as loop normalization, adding explicit loop labels, and constant folding.
4. Converting Ada symbol tables to a C++ style, ensuring case sensitivity. Ada operator declarations from the Standard packages are converted to operator notation.
5. Top-down traversal of the AST (using preorder) to translate types to C++ types and type constants. This step also maps loop exits to loop labels and translates Ada comments into C++ style comments.
6. Bottom-up traversal of the AST (using the reversed preorder) to rewrite different types of Ada nodes into equivalent C++ nodes, including Ada operators, constants, expressions, and statements. The reversed preorder traversal in this step ensures the walking of the AST is still valid when being modified, avoiding issues similar to C++ iterator invalidation.
7. Running AST consistency tests after all transformation is done. This helps catch any translation errors.
8. Running ROSE's Backend generating the final C++ source code from the translated AST.

The design of Ada2Cpp follows some best practices we have accumulated when developing ROSE-based tools. For example, AST normalization is applied first to simplify and unify the AST as much as possible. This step makes later translation easier to implement. The major translation work is done through a two-pass: 1) a top-down tree walk to translate and populate types followed by 2) a bottom-up walk to translate operators, expressions and statements.

In the end, the original Ada AST is changed to C++ AST in place, preserving shared input AST nodes as many as possible. An alternative translation choice is to keep the original Ada AST while generating a complete new C++ AST. We did not choose this option to be more memory efficient.

Given the specific semantics of Ada arrays and ranges, we created a C++ library to mimic Ada array and range types. The translator generates C++ code utilizing the special library to preserve Ada-specific semantics.

Consider the procedure `Demo_range1` containing a loop computing the sum of elements in an integer range type.

```

1  -- example using range
2  procedure Demo_range1 is
3    subtype Index_Range is Integer range 0..9;
4    Sum : Integer := 0;
5  begin
6    for I in Index_Range loop
7      Sum := Sum + I;
8    end loop;
9  end Demo_range1;

```

The output C++ code generates a `main` function, which in turn calls the `Demo_range1` function which contains the for loop.

```

1  #include "Ada/Range.hxx"
2  #include "Ada/Array.hxx"
3  //example using range
4  void Demo_range1 () {
5    typedef struct Ada::Range<int, 0, 9>
6      Index_Range;
7    int Sum = 0;
8    for (auto I: Index_Range())
9    {
10      Sum = Sum + I;
11    }
12  }

```

Note that static Ada range types are translated into C++ template types in the form of `Ada::Range<type, start, end>` provided by our C++ array library mimicking semantics of Ada types. The corresponding header files are also included so that the generated C++ code has access to the type declarations. Ada comments are also translated to C++ style comments.

The Ada style loop using a range is translated into a C++ for loop using the `Ada::Range<>` type. The `Index_Range()` expression creates an instance of the `Index_Range` type (the range from 0 to 9), and `auto I` deduces the type of `I` automatically from the range's elements, which are integers in this case.

5 Related Work

Several other source-to-source translation frameworks exist. The Clang [8] and LLVM [9] frameworks form the high-level and low-level components of a compiler toolchain. In contrast to ROSE, Clang uses an immutable internal representation and transforming the code involves updating the source text followed by reparsing the code. LLVM represents code more closely to machine level and it is difficult to generate code that is maintainable by humans from a low-level representation.

The libadalang framework offers capabilities for parsing, analysis, and instrumentation [10] of Ada sources. We are planning to adopt libadalang as frontend for ROSE.

Coccinelle is a transformation system for C code [11]. Coccinelle uses a semantic patching language (SmPL) to describe transformations and has been applied to evolve the Linux kernel.

6 Conclusion

This paper has discussed the Ada support within the ROSE compiler. ROSE enables software maintenance tools that target source code. Four prototype tools were presented.

We plan to continue enhancing the Ada support in ROSE by adopting libadalang as the frontend parser for long-term support and for being more permissive with respect to accepted source code. Also, with libadalang we will strive to support modern Ada standards fully. Lastly, we plan on implementing token-based unparsing for Ada, which promises to fully generate files from the IR while preserving code formatting in the original file.

ROSE is available as open source from <https://github.com/rose-compiler/rose>. How to install ROSE with Ada is detailed in the LCOM tool repository <https://github.com/LLNL/ROSE-LCOM-Tools>.

Acknowledgments

We thank the anonymous referees for their feedback and suggestions for improvement.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-861134.

References

- [1] D. Quinlan and C. Liao, “The ROSE source-to-source compiler infrastructure,” in *Cetus users and compiler infrastructure workshop, in conjunction with PACT*, 2011.
- [2] ISO/IEC 15291 International Standard, *Ada Semantic Interface Specification (ASIS)*. International Organization for Standardization, Apr. 1999.
- [3] AdaCore, “GNAT ASIS library,” 2019.
- [4] ISO/IEC 8652 International Standard, *Ada*. International Organization for Standardization, Feb. 1995.
- [5] J. Barnes, *Rationale for Ada 2012*. John Barnes Informatics, 2013.
- [6] K. Lamar, P. Pirkelbauer, and D. Dechev, “LCOM metric analyzer: A case study,” under review. available upon request.
- [7] J. Barnes, *Rationale for Ada 2005*. John Barnes Informatics, 2005.
- [8] “Clang transformation tutorial.” <https://clang.llvm.org/docs/ClangTransformerTutorial.html>, 2024. accessed on June 15, 2024.
- [9] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis and transformation,” (San Jose, CA, USA), pp. 75–88, Mar 2004.
- [10] AdaCore, “Libadalang User Manual 25.0w.” https://docs.adacore.com/live/wave/libadalang/html/libadalang_uug/introduction.html, 2023. accessed on June 20, 2024.
- [11] J. Lawall, “On the origins of coccinelle,” in *Eelco Visser Commemorative Symposium (EVCS 2023)*, pp. 18:1–18:10, 2023.

Task-to-Thread Mapping in OpenMP Using Fuzzy Decision Making

Mohammad Samadi*, Tiago Carvalho, Luís Miguel Pinho

Polytechnic Institute of Porto & INESC TEC, Porto, Portugal; email: {mmas, tdc, lmp}@isep.ipp.pt

Sara Royuela

Barcelona Supercomputing Center, Barcelona, Spain; email: sara.royuela@bsc.es

Abstract

The performance of shared-resource multi-core hardware platforms in complex cyber-physical systems (CPSs), e.g., automotive industry, can be improved using task-based parallelism through OpenMP. However, most CPS require certain level of predictability, which challenges the efficient implementation of the task-to-thread mapping process. This exploratory work build on the fact that existing mapping methods mostly use elementary or heuristic algorithms, and the idea that artificial intelligence (AI) algorithms can be used to enhance the efficiency of such processes. Accordingly, this paper (1) evaluates the suitability of AI-based techniques in improving the performance of task-to-thread mapping in the OpenMP framework, and (2) proposes a hypothesis to perform an intelligent mapping using fuzzy logic for multi-queue schedulers to improve the predictability of the system.

Keywords: OpenMP, predictability, task-to-thread mapping, artificial intelligence (AI), fuzzy logic.

1 Introduction

OpenMP is a parallel framework for heterogeneous shared-memory architectures, which can be used to leverage high-performance computing (HPC) capabilities. It has been increasingly used over the past decade to improve the performance of CPSs, e.g., robotics, automation, and satellites. Although OpenMP applications can be executed on multi-core hardware platforms at high speed, the predictability (e.g., real-time analysis) of the parallel system is still a challenge.

Early versions of OpenMP support a thread-centric model for exploiting massively loop-intensive and data-parallel programs, while later versions support a task-centric model that allows complex, fine-grained, and irregular parallelism [1]. Focusing on the latter, task-to-thread mapping can be performed using centralized or distributed queues. In the former case, ready tasks are added to a single queue accessed by all threads to execute tasks (which increases contention in the queue) [2]. In the latter model, instead, each thread includes an individual allocation queue [3, 4], possibly reducing the contention of the system.

In parallel frameworks, task-to-thread mapping can be performed according to the timing requirements of the system. However, the main restriction of most of the existing algorithms is that they mainly use elementary or heuristic techniques that may not be able to make more efficient decisions under unforeseen conditions, which is the target of knowledge-based system [1, 2, 3, 4]. This paper explores an AI-based mechanism for task-to-thread mapping in OpenMP using fuzzy theory. A multi-queue system model is considered, where OpenMP tasks are allocated to threads' queues in the allocation phase, while they are deallocated from the queues and dispatched to threads in the dispatching phase. A fuzzy controller is proposed for the allocation phase to select the most appropriate queue for each OpenMP task. It is worth noticing that the First In, First Out (FIFO) algorithm is applied in the dispatching phase to select the task from the queue to execute by its idle thread. The main objective is to enhance the predictability of critical systems.

2 Background and Related Work

This section provides an overview of the terms needed to understand this article and introduces the most relevant state-of-the-art works related to the proposed technique.

2.1 Task-to-thread mapping

Task-to-thread mapping is a key process in parallel applications (e.g., OpenMP). It can be carried out through a single-queue (e.g., GCC) or multi-queue (e.g., LLVM) system. In the former model, after a task meets its input data dependencies, it is allocated to a queue, from which it is later deallocated and dispatched to an idle thread. In the latter model, which is considered in this paper, tasks are allocated to one of the allocations queues (belonging to threads) and then deallocated from the queue and dispatched to the thread when idle. The main problem with the former system is that all threads access the same queue for allocating and deallocating tasks, so the contention in the queue can potentially increase.

Mainstream implementations for OpenMP task-to-thread mapping use breadth-first scheduler (BFS). This algorithm creates all child tasks before executing them. Some implementations might also support work-first scheduler (WFS). This algorithm executes new tasks immediately after they are created, causing the suspension of the encountering (i.e., parent) task [1]. However, WFS causes in many cases the sequentialization of the execution and it is typically not used.

*PhD Candidate, Universitat Politècnica de Catalunya, Barcelona, Spain

Melani et al. [2] have introduced two schedulers for OpenMP, including optimal and sub-optimal approaches. The optimal (but expensive) approach uses an integer linear programming (ILP) formulation to optimally allocate task-parts (i.e., the chunks of sequential code in-between task scheduling points, or points in which the scheduler might decide to suspend a task for resuming it later) to threads and minimize the task-set makespan. For the cases where the optimal approach may be too computationally expensive, the sub-optimal approach uses several tractable and less computational expensive heuristics to select tasks from a single queue and execute them by idle threads.

Samadi et al. [4] have presented a heuristic task-to-thread mapping through a multi-queue tasking system, which consists of allocation and dispatching phases. Multiple heuristics are provided in the allocation phase to select allocation queues for OpenMP tasks, as well as several heuristics are employed in the dispatching phase to select tasks from the queues and execute them by idle threads. Some heuristics are proved to reduce the application response time. However, in general, this method is still not be able to perform better than basic ones in uncertain conditions.

2.2 Artificial intelligence

AI can be used in engineering applications to solve the problems in unforeseen conditions. Some of the AI algorithms (e.g., machine learning and fuzzy logic) use predefined information (i.e., training set), while others (e.g., A* and ant colony) employ intelligent procedures (e.g., heuristic path finding) to find near-optimal solutions to complex problems [5]. AI techniques can be used in different applications (e.g., cyber-physical systems [6]) to improve the performance of the system.

Fuzzy logic is one of the most prominent AI techniques. It uses a knowledge-based system capable of making suitable decisions even in unforeseen conditions. This mechanism uses *if-then* rules composed of linguistic terms (e.g., high) as a knowledge base to first train a model based on the rules and then estimate unforeseen situations based on the model. Instead of using only boolean values, as boolean logic does, fuzzy logic uses all values between 0 and 1 [7]. All operations in this logic are performed using the fuzzy inference system (FIS), which is composed of four main components: fuzzification, rule making, inference engine, and defuzzification. Fuzzification converts a crisp value (e.g., 20°C) to a fuzzy set (e.g., [0, 0.2, 1, 0.5, 0]). Note that a universal set (e.g., [-40, -5, 5, 20, 50] °C for temperature) is defined to represent linguistic terms and crisp values as fuzzy sets. Rule making creates a fuzzy rule based on inputs and output(s) through the *if-then* rule and linguistic terms. The total fuzzy rule (i.e., model) can be built using the aggregation operation between all fuzzy rules. The inference engine predicts uncertain conditions (i.e., new output) based on new inputs and the model. Finally, defuzzification converts a fuzzy set into a crisp value [8].

3 System model

The system model considers real-time tasks implemented using graphs of OpenMP parallel computation. Each real-time

```

1 #pragma omp parallel
2 #pragma omp single
3 {
4     #pragma omp task depend(out: A,B,C)
5     Task1();
6
7     #pragma omp task depend(in: A) depend(out: D,F)
8     Task2();
9
10    #pragma omp task depend(in: C) depend(out: E)
11    Task3();
12
13    #pragma omp task depend(in: B,D,E)
14    Task4();
15
16    #pragma omp task depend(in: F)
17    Task5();
18 }

```

Figure 1: OpenMP data dependencies.

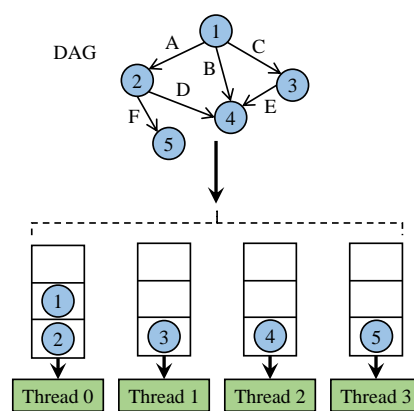


Figure 2: Multi-queue tasking systems.

task is represented as one OpenMP task dependency graph (TDG). OpenMP allows us to specify data dependencies between OpenMP tasks (Figure 1), where data dependent tasks can be executed if and only if the input data dependencies are met.¹

In this model, a real-time task is therefore represented as a directed acyclic graph (DAG), i.e. the task dependency graph (TDG), nominated with $G = (V, E)$, where V is the set of vertices (i.e., OpenMP tasks) and E is the set of edges (i.e., the data dependencies between tasks). Furthermore, a real-time task in multi-threaded applications can be executed with multiple threads; Figure 2 exemplifies a possible deployment of the OpenMP DAG created by the code in Figure 1 onto a multi-threaded systems, where the number of OpenMP threads is assumed to be equal to the number of processor cores.

¹Note that in some of the OpenMP-based benchmarks (available at <https://gitlab.bsc.es/ampere-sw/wp2/general-information/>), e.g., the AXPY application, OpenMP tasks do not include any data dependencies, so OpenMP tasks can be allocated to the queues at the beginning of the execution process. Other cases, e.g., the Heat application, tasks can only be allocated after the data they depend upon is available. Accordingly, the complexity of an OpenMP program depends on the number of tasks and their data dependencies.

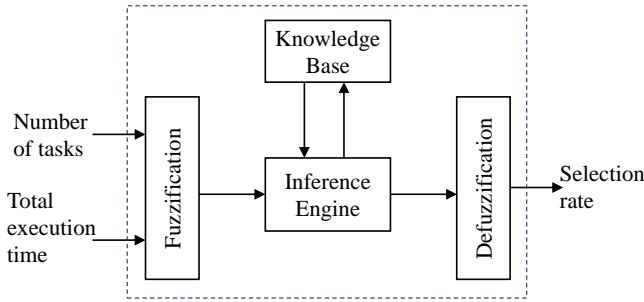


Figure 3: Structure of the suggested fuzzy controller.

4 Methodology

With the aim of achieving predictability, this work proposes a new methodology based on FIS for the allocation phase of the task-to-thread mapping process of OpenMP applications deployed on a multi-queue system. The methodology uses a fuzzy controller to select the most appropriate allocation queue for each OpenMP task when it does not include any data dependencies or they have been met. The controller, shown in Figure 3, consists of two input parameters, nominated as number of tasks and total execution time, and one output parameter, nominated as selection rate. The queue with the highest selection rate will be chosen for the task allocation process. As this controller is used for predictable OpenMP applications, the parameters have been selected in a way to improve the performance and the predictability of the system, i.e., the application's response time.

The linguistic terms used in the proposed controller are {feeble, few, normal, many, lots} for the *number of tasks*, {very small, small, mean, large, very large} for the *total execution time*, and {very low, low, medium, high, very high} for the *selection rate*. The number of (maximum) rules for this controller is based on the number of inputs, which are 5 for each possible input, hence $5^2 = 25$ in total. Then, the rules are defined in such a way that the queue with the minimum number of tasks and the least total execution time is selected in the task allocation process. For example, if the *number of tasks* is *feeble* and the *total execution time* is *very small*, then the selection rate should be *very high*. Table 1 contains some of the *if-then* rules defined in the controller.

Rule #	Number of tasks	Total execution time	Selection rate
1	feeble	very small	very high
2	few	small	high
3	normal	mean	medium
4	many	large	low
5	lots	very large	very low

Table 1: Some of the *if-then* rules used in the controller

A fuzzy set is used in FIS based on the universal set to represent a linguistic term or a crisp value. In the suggested controller, the lower and upper bounds in the universal set are {0, 4} for the *number of tasks*, {0, 100,000,000} *ns* for the *total execution time*, and {0, 1} for the *selection rate*.

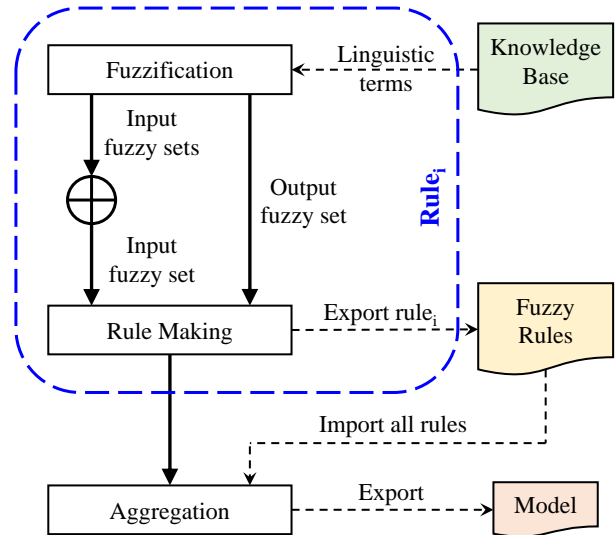


Figure 4: Workflow that builds the model using the controller.

Consequently, the universal set for each parameter can be defined with continuous numbers between the lower and upper bounds. Note that these bounds are collected based on recording the tasks-to-thread mapping using two heuristics recently proposed in the state-of-the-art [4]: (i) MNTTP (selecting the most appropriate allocation queue based on the minimum number of task-parts) for the *number of tasks* parameter, and (ii) MTET (selecting the most suitable queue based on the least total execution time of tasks in the queues) for the *total execution time* parameter.

In the fuzzification process, the *triangular membership function* is used to fuzzify the linguistic terms and crisp values for the input parameters, while the *bell-shaped membership function* is used to fuzzify them for the output parameter. The *Mamdani rule-making type* (i.e., the *Maximum-Minimum function*) is used to make the fuzzy rules based on the input and output fuzzy sets for each *if-then* rule, as well as the *Maximum function* is applied to build the total fuzzy rule (i.e., the model) based on the aggregation of all the fuzzy rules. Additionally, in the defuzzification process, the *Center of Gravity* (also known as Center of Area or Centroid) method is used to defuzzify fuzzy set of the output parameter. More details about these processes can be found in recent works [8].

Figure 4 shows the process of building the model using the proposed controller. All the processes within the blue dashed region are separately performed for each fuzzy rule (i.e., *Rule_i*). First, linguistic terms belonging to the *if-then* rule in the knowledge base are converted to fuzzy sets during the fuzzification process. The input fuzzy sets are then converted to a fuzzy set to specify a unified input fuzzy set using the *Minimum function*. Finally, the fuzzy rule is created using the rule making process based on the input and output fuzzy sets. After performing this process for each *if-then* rule, all the fuzzy rules are aggregated using the aggregation process to build the final set of fuzzy rules, namely the model. This model will be used in the execution process to predict a selection rate for each queue based on new inputs.


```

1  N = Number of threads
2  NT[0..N-1] = Number of tasks
3  TET[0..N-1] = Total execution time
4  SR[0..N-1] = Selection rate
5
6  // Allocation phase
7  For each OpenMP task in the DAG do {
8    Update the information of queues
9    I = 0
10   While (I < N) {
11     NT_Fuzzy = Fuzzification (NT[I])
12     TET_Fuzzy = Fuzzification (TET[I])
13     Input_Fuzzy = Minimum (NT_Fuzzy, TET_Fuzzy)
14     SR_Fuzzy = Inference (Input_Fuzzy, Model)
15     SR[I] = Defuzzification (SR_Fuzzy)
16     I = I + 1
17   }
18   Select queue with highest selection rate
19   Allocate task to the queue
20 }
21
22 // Dispatching phase
23 For each thread do {
24   If thread is idle and its queue is not empty {
25     Select a ready task based on the FIFO
26     Dispatch task-to-thread and execute it
27   }
28 }

```

Figure 5: Intelligent task-to-thread mapping.

Figure 5 illustrates the algorithm for performing an intelligent task-to-thread mapping in OpenMP using the suggested fuzzy controller based on the model built above. First, lines 1-to-4 define the variables. Then, during the allocation phase in lines 7-to-20, each OpenMP task with no data dependencies or whose data dependencies have been already met is allocated to one of the queues. During this process, (1) the scheduler updates the information of queues (i.e., number of tasks and total execution time), (2) a selection rate is predicted for each queue based on the new inputs and the model, and (3) the queue with the highest selection rate (to improve the timing requirements of the system) is selected, and the task is allocated to this queue. Finally, during the dispatching phase in lines 23-to-28, tasks are selected for execution following the typical implementations based on multi-queue schedulers (e.g., the LLVM compilation framework), where each idle thread selects a task from its own queue using the FIFO algorithm. Note that the approach does not use work-stealing (e.g., as implemented by LLVM) where threads can steal tasks from other threads' queues.

5 Conclusions and future work

This paper introduces the preliminary ideas to implement an OpenMP scheduler for task-to-thread mapping in multi-queue systems using a controller based on fuzzy logic. The controller includes *number of tasks* and *total execution time* as the inputs, and *selection rate* as the output to select the most suitable queue (belonging to threads) for OpenMP tasks in the allocation phase. Additionally, idle threads select ready

tasks from their queues using the FIFO algorithm to execute them in the dispatching phase. In the future work, this AI-based mapping will be simulated first and implemented later under different configurations (i.e., number of tasks, types of TDGs, and number of threads, among others) to show its performance in achieving efficient task-to-thread mapping compared to the literature.

6 Acknowledgment

This work has been partly funded by the RESPECT project (Departament de Recerca i Universitats de la Generalitat de Catalunya record No 2021 PROD 00179).

References

- [1] M. A. Serrano, A. Melani, R. Vargas, A. Marongiu, M. Bertogna, and E. Quinones, "Timing characterization of openmp4 tasking model," pp. 157–166, 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2015.
- [2] A. Melani, M. A. Serrano, M. Bertogna, I. Cerutti, E. Quinones, and G. Buttazzo, "A static scheduling approach to enable safety-critical openmp applications," pp. 659–665, 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC), 2017.
- [3] M. S. Gharajeh, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quinones, "Heuristic-based task-to-thread mapping in multi-core processors," pp. 1–4, 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA), 2022.
- [4] M. Samadi, S. Royuela, L. M. Pinho, T. Carvalho, and E. Quinones, "Time-predictable task-to-thread mapping in multi-core processors," *Journal of Systems Architecture*, vol. 148, p. 103068, 2024.
- [5] E. Töugu, *Algorithms and Architectures of Artificial Intelligence*, vol. 159. IOS Press, 2007.
- [6] G. Tartarisco, G. Cicceri, R. Bruschetta, A. Tonacci, S. Campisi, S. Vitabile, A. Cerasa, S. Distefano, A. Pellegriano, P. A. Modesti, *et al.*, "An intelligent medical cyber-physical system to support heart valve disease screening and diagnosis," *Expert Systems with Applications*, vol. 238, p. 121772, 2024.
- [7] V. Novák, "Reasoning about mathematical fuzzy logic and its future," *Fuzzy Sets and Systems*, vol. 192, pp. 25–44, 2012.
- [8] M. S. Gharajeh, "Fsb-system: a detection system for fire, suffocation, and burn based on fuzzy decision making, mcdm, and rgb model in wireless sensor networks," *Wireless Personal Communications*, vol. 105, no. 4, pp. 1171–1213, 2019.

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



2 Rue Docteur Lombard
92441 Issy-les-Moulineaux Cedex
France
Contact: Jean-Pierre Rosen
rosen@adalog.fr
www.adalog.fr/en/

AdaCore

46 Rue d'Amsterdam
F-75009 Paris
France
sales@adacore.com
www.adacore.com



Jacob Bontiusplaats 9
1018 LL Amsterdam
The Netherlands
Contact: Wido te Brake
wido.tebrake@deepbluecap.com
www.deepbluecap.com



Ada Labs
innovate.all

506 Royal Road
La Caverne, Vacoas 73310
Republic of Mauritius
Contact: David Sauvage
david.sauvage@adalabs.com
www.adalabs.com



24 Quai de la Douane
29200 Brest, Brittany
France
Contact: Pierre Dissaux
pierre.dissaux@ellidiss.com
www.ellidiss.com

EUROCITY

Rue Marie de Bourgogne 52
1000 Brussels
Belgium
Contact: Emma Claus
Emma.Claus@eurocity.be
www.eurocity.com



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-tool



Enterprise House
Baloo Avenue, Bangor
North Down BT19 7QT
Northern Ireland, UK
enquiries@sysada.co.uk
sysada.co.uk



1115 Rue René Descartes
13100 Aix en Provence
France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/



Tiirasaarentie 32
FI 00200 Helsinki
Finland
Contact: Niklas Holsti
niklas.holsti@tidorum.fi
www.tidorum.fi



Beckengässchen 1
8200 Schaffhausen
Switzerland
Contact: Ahlan Marriott
admin@white-elephant.ch
www.white-elephant.ch

