ADA USER JOURNAL

Volume 45
Number 1
March 2024

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

My initial words in this editorial, the first in 2024, are about the significant changes that are happening in our Ada ecosystem. I'm referring to the integration of SIGAda (the ACM Special Interest Group on Ada) in SIGPLAN (the SIG on Programming Languages). Despite this change, SIGAda's sister publication, Ada Letters, will continue to exist in online form and share content with the Ada User Journal. Therefore, we still hope that at some point, possibly by the end of 2024, it may be possible to entail conversations with the ACM on how and in which terms a single publication may come to life. As a side note, but also very importantly, a consequence of the extinction of SIGAda is that a new global association is being created, aiming to become a solid home base for Ada friends from all over the world. The plans, as explained in a letter from the Ada-Europe Board to all Ada-Europe members, are grounded on several principles. The one directly relevant to the AUJ is that, and I quote, "The merger of our Ada User Journal with Ada Letters into a single quarterly magazine, attributed to the new association and shared with Ada-Europe's membership base, should be swiftly finalized". I will probably come back to this topic in the next editorial.

Concerning the technical contents of this issue, it provides the Proceedings of the "ADEPT: AADL by its Practitioners Workshop" of AEiC 2023, which took place in Lisbon, Portugal, last June. It was the second edition of the workshop, and again a successful event. The proceedings include eight technical contributions and a summary paper prepared by the workshop organizers H. N. Tran and F. Singhoff from the University of Brest, in France, and J. Hugues from Carnegie Mellon University in the USA. The workshop features contributions that somehow explore or are related to the Architecture Analysis and Design Language (AADL), which is an SAE International Standard dedicated to the precise modelling of complex embedded systems, which allows, among other things, to perform reliability analysis of the modelled systems. The reader is invited to read the summary paper to get a complete perspective of the contents of these proceedings.

As usual, the issue includes the News Digest section prepared by Alejandro R. Mosteo and the Calendar and Events section prepared by Dirk Craeynest.

Antonio Casimiro
Lisboa
March 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

[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,

Once more, the flagship Ada conference is upon us [1], this year taking place in Barcelona, Spain. Furthermore, among its satellite activities is an "Ada Developers Workshop" [2] that aims to fill in for the sorely missed "Ada Developer Room" of FOSDEM past.

For lovers of Ada nitty-gritty details, this period includes a discussion of Container and Cursor semantics [3] with head-butting positions, so the reader can take sides (or hold their unopposed personal truth at home ;-)).

Sincerely,

Alejandro R. Mosteo.

[1] "AEiC 2024 - Ada-Europe Conference - Deadlines Approaching", in Ada-related Events.

[2] "Ada Developer Workshop @ AEiC 2024, a New "FOSDEM DevRoom" for the Community", in Ada-related Events.

[3] "Re: Map Iteration and Modification", in Ada Practice.

Ada-related Events

Ada-Europe Conference - 31 Jan Journal Track Extended Deadline

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: Ada-Europe conference - 31 Jan Journal Track Extended Deadline
Date: Mon, 8 Jan 2024 10:43:48 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

UPDATED Call for Contributions

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

11-14 June 2024, Barcelona, Spain

www.ada-europe.org/conference2024

*** Journal track deadline EXTENDED to 31 January 2024 ***

*** Other submissions by 26 February 2024 ***

Organized by Ada-Europe and Barcelona Supercomputing Center (BSC), in cooperation with ACM SIGAda, ACM SIGBED, ACM SIGPLAN, and Ada Resource Association (ARA)

#AEiC2024 #AdaEurope #AdaProgramming

General Information

The 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) will take place in Barcelona, Spain.

AEiC is a leading international forum for providers, practitioners, and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development, and maintenance of long-lived, high-quality software systems for a challenging variety of application domains. The program will also include keynotes, Q&A and discussion sessions, and social events. Participants include practitioners and researchers from industry, academia, and government organizations active in the development of reliable software technologies.

The topics of interest for the conference include but are not limited to (more specific topics are described on the conference web page):

* Formal and Model-Based Engineering of Critical Systems;
* High-Integrity Systems and Reliability;
* AI for High-Integrity Systems Engineering;
* Real-Time Systems;
* Ada Language;
* Applications in Relevant Domains.

The conference comprises different tracks and co-located events:

* Journal track papers present research advances supported by solid theoretical foundation and thorough evaluation.
* Industrial track contributions highlight industrial open challenges and/or the practitioners' side of a relevant case study or industrial project.
* Work-in-progress track papers illustrate novel research ideas that are still at an initial stage, between conception and first prototype.
* Tutorials guide attenders through a hands-on familiarization with innovative developments or with useful features related to reliable software.
* Workshops provide discussion forums on themes related to the conference topics.
* Vendor presentations and exhibitions allow for companies to showcase their latest products and services.

Important Dates

31 January 2024 EXTENDED submission deadline for journal track papers

26 February 2024 Deadline for submission of industrial track papers, work-in-progress papers, tutorial and workshop proposals

22 March 2024 First round notification for journal track papers, and notification of acceptance for all other types of submissions

11-14 June 2024 Conference

Call for Journal Track Submissions

Following a journal-first model, this edition of the conference includes a journal track, which seeks original and

high-quality papers that describe mature research work on the conference topics. Accepted journal track papers will be published in a Special Issue of Elsevier JSA - the Journal of Systems Architecture (Q1 ranked, CiteScore 8.5, impact factor 4.5). Accordingly, the conference is listed as "Journal Published" in the latest update of the CORE Conference Ranking released in August 2023. Contributions must be submitted by 31 January 2024. Submissions should be made online at https://www.editorialmanager.com/jsa/, selecting the "Ada-Europe AEiC 2024" option (submission page open from 15 November 2023) as article type of the paper. General information for submitting to the JSA can be found at the Journal of Systems Architecture website.

JSA has adopted the Virtual Special Issue model to speed up the publication process, where Special Issue papers are published in regular issues, but marked as SI papers. Acceptance decisions are made on a rolling basis. Therefore, authors are encouraged to submit papers early, and need not wait until the submission deadline. Authors who have successfully passed the first round of review will be invited to present their work at the conference. The abstract of the accepted contributions will be included in the conference booklet.

The Ada-Europe organization will waive the Open Access fees for the first four accepted papers (whose authors do not already enjoy Open Access agreements). Subsequent papers will follow JSA regular publishing track. Prospective authors may direct all enquiries regarding this track to the corresponding chairs, Bjorn Andersson (baandersson@sei.cmu.edu) and Luis Miguel Pinho (lmp@isep.ipp.pt).

Call for Industrial Track Submissions

The conference seeks industrial practitioner presentations that deliver insight on the challenges of developing reliable software. Especially welcome kinds of submissions are listed on the conference website. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer review process. Interested authors shall submit a 1-to-2 pages abstract, by 26 February 2024, via EasyChair at https://easychair.org/my/conference?conf=aeic2024, selecting the "Industrial Track". The format for submission is strictly in PDF, following the Ada User Journal style. Templates are available at http://www.ada-europe.org/auj/guide.

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference and will also be invited to expand their contributions into full-fledged articles for publication in the Ada User Journal, which will form the proceedings of the industrial track of the Conference. Prospective authors may direct all enquiries regarding this track to its chairs Luciana Provenzano (luciana.provenzano@mdu.se) and Michael Pressler (Michael.Pressler@de.bosch.com).

Call for Work-in-Progress Track Submissions

The work-in-progress track seeks two kinds of submissions: (a) ongoing research and (b) early-stage ideas. Ongoing research submissions are 4-page papers describing research results that are not mature enough to be submitted to the journal track. Early-stage ideas are 1-page papers that pitch new research directions that fall within the scope of the conference. Both kinds of submissions must be original and shall undergo anonymous peer review. Submissions by recent MSc graduates and PhD students are especially sought. Authors shall submit their work by 26 February 2024, via EasyChair at https://easychair.org/my/conference?conf=aeic2024, selecting the "Work-in-Progress Track". The format for submission is strictly in PDF, following the Ada User Journal style. Templates are available at http://www.ada-europe.org/auj/guide.

The abstract of the accepted contributions will be included in the conference booklet. The corresponding authors will get a presentation slot in the prime-time technical program of the conference and will also be offered the opportunity to expand their contributions into 4-page articles for publication in the Ada User Journal, which will form the proceedings of the WiP track of the Conference. Prospective authors may direct all enquiries regarding this track to the corresponding chairs Alejandro R. Mosteo (amosteo@unizar.es) and Ruben Martins (rubenm@andrew.cmu.edu).

Awards

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

Call for Tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic or practitioner slant, including hands-on or practical elements. Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half-day or full-day), the intended level of the contents (introductory, intermediate, or advanced), and a statement motivating attendance. Tutorial proposals shall be submitted at any time but no later than the 26 February 2024 to the respective chair Maria A. Serrano (maria.serrano@nearbycomputing.com), with subject line: "[AEiC 2024: tutorial proposal]". Once submitted, each tutorial proposal will be evaluated by the conference organizers as soon as possible, with decisions from January 1st. The authors of accepted full-day tutorials will receive a complimentary conference registration, halved for half-day tutorials. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

The conference welcomes satellite workshops centred on themes that fall within the conference scope. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the AEiC conference. Workshop organizers shall also commit to producing the proceedings of the event, for publication in the Ada User Journal. Workshop proposals shall be submitted at any time but no later than the 26 February 2024 to the respective chair Sergio Saez (ssaez@disca.upv.es), with subject line: "[AEiC 2024: workshop proposal]". Once submitted, each workshop proposal will be evaluated by the conference organizers as soon as possible, with decisions from January 1st.

Academic Listing

The Journal of Systems Architecture, publication venue of the journal track proceedings of the conference, is Q1 ranked, with CiteScore 8.5 and Impact Factor 4.5. The Ada User Journal, venue of all other technical proceedings of the conference, is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Call for Exhibitors and Sponsors

The conference will include a vendor and technology exhibition with the option of a 20 minutes presentation as part of the conference program. Interested providers should direct inquiries to the Exhibition & Sponsorship Chair Ahlan Marriot (ahlan@ada-switzerland.ch).

Venue

The conference will take place in Barcelona, Spain. Barcelona is a major cultural, economic, and financial centre, known for its architecture, culture, and Mediterranean atmosphere, a hub for technology and innovation. There's plenty to see and visit in Barcelona, so plan in advance!

Organizing Committee

- Conference Chair

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

- Journal Track Chairs

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

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

- Industrial Track Chairs

Luciana Provenzano, Mälardalen University, Sweden
luciana.provenzano@mdu.se

Michael Pressler, Robert Bosch GmbH, Germany
Michael.Pressler@de.bosch.com

- Work-In-Progress Track Chairs

Alejandro R. Mosteo, CUD Zaragoza, Spain
amosteo@unizar.es

Ruben Martins, Carnegie Mellon University, USA
rubenm@andrew.cmu.edu

- Tutorial Chair

Maria A. Serrano, NearbyComputing, Spain
maria.serrano@nearbycomputing.com

- Workshop Chair

Sergio Saez, Universitat Politècnica de València, Spain
ssaez@disca.upv.es

- Exhibition & Sponsorship Chair

Ahlan Marriott, White Elephant GmbH, Switzerland
ahlan@Ada-Switzerland.ch

- Publicity Chair

Dirk Craeynest, Ada-Belgium & KU Leuven, Belgium
Dirk.Craeynest@cs.kuleuven.be

- Webmaster

Hai Nam Tran, University of Brest, France
hai-nam.tran@univ-brest.fr

- Local Chair

Nuria Sirvent, Barcelona Supercomputing Center, Spain
nuria.sirvent@bsc.es

Journal Track Committee

Al Mok, University of Texas at Austin, USA

Alejandro Mosteo, CUD Zaragoza, Spain

Alwyn Godloe, NASA, USA

António Casimiro, University of Lisbon, Portugal

Barbara Gallina, Mälardalen University, Sweden

Bernd Burgstaller, Yonsei University, South Korea

C. Michael Holloway, NASA, USA

Cristina Seceleanu, Mälardalen University, Sweden

Doug Schmidt, Vanderbilt University, USA

Frank Singhoff, University of Brest, FR

George Lima, Universidade Federal da Bahia, Brazil

Isaac Amundson, Rockwell Collins, USA

Jérôme Hugues, CMU/SEI, USA

José Cruz, Lockeed Martin, USA

Kristoffer Nyborg Gregertsen, SINTEF Digital, Norway

Laurent Pautet, Telecom ParisTech, France

Leonidas Kosmidis, Barcelona Supercomputing Center, Spain

Mario Aldea Rivas, University of Cantabria, Spain

Matthias Becker, KTH - Royal Institute of Technology, Sweden

Patricia López Martínez, University of Cantabria, Spain

Sara Royuela, Barcelona Supercomputing Center, Spain

Sergio Sáez, Universitat Politècnica de València, Spain

Tucker Taft, AdaCore, USA

Tullio Vardanega, University of Padua, Italy

Xiaotian Dai, University of York, England

Industrial Track Committee

Aida Causevic, Alstom, Sweden

Alexander Viehl, Research Center for Information Technology, Germany

Ana Rodríguez, Silver Atena, Spain

Aurora Agar, NATO, Netherlands

Behnaz Pourmohseni, Robert Bosch GmbH, Germany

Claire Dross, AdaCore, France

Elena Lisova, Volvo CE, Sweden

Enricco Mezzeti, Barcelona Supercomputing Center, Spain

Federico Aromolo, Scuola Superiore Sant'Anna, Italy

Helder Silva, Edisoft, Portugal

Hugo Torres Vieira, Evidence Srl, Italy

Irune Agirre, Ikerlan, Spain

Jordi Cardona, Rapita Systems, Spain

José Ruiz, AdaCore, France

Joyce Tokar, Raytheon, USA

Luciana Alvite, Alstom, Germany

Marco Panunzio, Thales Alenia Space, France

Patricia Balbastre Betoret, Valencia Polytechnic University, Spain

Philippe Waroquiers, Eurocontrol NMD, Belgium

Raúl de la Cruz, Collins Aerospace, Ireland

Santiago Urueña, GMV, Spain

Stef Van Vlierberghe, Eurocontrol NMD, Belgium

Work-in-Progress Track Committee

Alan Oliveira, University of Lisbon, Portugal

J. Javier Gutiérrez, University of Cantabria, Spain

Jérémie Guiochet, LAAS-CNRS, France

Kalinka Branco, University of São Paulo, Brazil

Katherine Kosaian, University of Iowa, USA

Kevin Cheang, AWS, USA

Kristin Yvonne Rozier, Iowa State University, USA

Leandro Buss Becker, University of Manchester, UK

Li-Pin Chang, National Yang Ming Chiao Tung University, Taiwan

Mathias Preiner, Stanford University, USA

Raffaele Romagnoli, Carnegie Mellon University, USA

Robert Kaiser, RheinMain University of Applied Sciences, Germany

Sara Abbaspour, Mälardalen University, Sweden

Sergi Alcaide, Barcelona Supercomputing Center, Spain

Simona Bernardi, Unizar, Spain

Stefan Mitsch, School of Computing at DePaul University, USA

Teresa Lázaro, Aragon's Institute of Technology, Spain

Tiago Carvalho, ISEP, Portugal

Yannick Moy, AdaCore, France

Previous Editions

Ada-Europe organizes annual international conferences since the early 80's. This is the 28th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain ('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04), York, UK ('05), Porto, Portugal ('06), Geneva, Switzerland ('07), Venice, Italy ('08), Brest, France ('09), Valencia, Spain ('10), Edinburgh, UK ('11), Stockholm, Sweden ('12), Berlin, Germany ('13), Paris, France ('14), Madrid, Spain ('15), Pisa, Italy ('16), Vienna, Austria ('17), Lisbon, Portugal ('18), Warsaw, Poland ('19), online from Santander, Spain ('21),

Ghent, Belgium ('22), and Lisbon, Portugal ('23).

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

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

(V4.1)

AEiC 2024 - Ada-Europe Conference - Deadlines Approaching

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Subject: AEiC 2024 - Ada-Europe conference - Deadlines Approaching
Date: Fri, 16 Feb 2024 19:07:10 -0000
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

UPDATED Call for Contributions - Additional Tracks

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

11-14 June 2024, Barcelona, Spain

*** DEADLINES approaching: 26 February and 4 March 2024 ***

www.ada-europe.org/conference2024

*** Submission DEADLINE 26 February 2024 ***

Workshops: submit to Workshop Chair, Sergio Saez ssaez@disca.upv.es subject "[AEiC 2024: workshop proposal]"

Tutorials: submit to Tutorial and Education Chair,
Maria A. Serrano maria.serrano@nearbycomputing.com subject "[AEiC 2024: tutorial proposal]"

*** EXTENDED submission DEADLINE 4 March 2024 ***

Industrial- and Work-in-Progress-track: submit via https://easychair.org/my/conference?conf=aeic2024 select "Industrial Track" or "Work in Progress Track"

For more information please see the full Call for Papers at www.ada-europe.org/conference2024/cfp.html

---

Organized by Ada-Europe and Barcelona Supercomputing Center (BSC), in cooperation with ACM SIGAda, ACM SIGBED, ACM SIGPLAN, and Ada Resource Association (ARA)

#AEiC2024 #AdaEurope #AdaProgramming

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

(V6.1)

Ada Developer Workshop @ AEiC 2024, a New "FOSDEM DevRoom" for the Community

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Subject: Ada Developer Workshop @ AEiC 2024, a new "FOSDEM DevRoom" for the community
Date: Sat, 24 Feb 2024 22:30:03 +0100
Newsgroups: comp.lang.ada

Dear Ada community,

I come with great news! For the past two years, there was no Ada DevRoom over @ FOSDEM, a place where the Ada community used to meet and share their work and projects. Some of us wanted to keep having such experience as we believed it to be a greatly beneficial aspect to the wider Ada community.

For this reason, Fabien Chouteau, Dirk Craeynest and Fernando Oleo Blanco, made a proposal to the Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024 aka Ada-Europe 2024) in order to have a "devroom" for the wider Ada community, just like in FOSDEM.

We were accepted and you can already find all the information over at the Ada Developer Workshop webpage [1]!

I would encourage everybody to take a look at it! Nonetheless, here is a quick summary highlighting some of the points:

- It will take place on Friday, 14th of June in Barcelona. Friday was chosen in order to minimise the amount of free days/holidays that we would need to take off from our jobs and allow us to then use the weekend to visit and enjoy Barcelona.
- The cost will be lower than for the main conference. Our goal is to make it completely free, just like FOSDEM, but this is still a Work-In-Progress (WIP).
- The nature of the event is similar to any past DevRoom that took place @ FOSDEM. The main difference is that now, being an open-source project will not be a requirement.
- March 31st, 2024 is the (current) deadline for submissions. If you would like to present your work or discuss topics, please, please please, keep this date in mind!

We are eager to hear from all of you. And if you have any questions, please, let us know!

[1] https://www.ada-europe.org/conference2024/adadev.html

From: Streaksu <streaksu@mailbox.org>
Date: Tue, 27 Feb 2024 06:51:03 +0100

That sounds amazing! Thank you so much for your work and to the people at AEiC for making it happen.

> The cost will be lower than for the main conference.

That would be a huge deal. I have not checked this edition's registrations, but if 2023's are anything to go by, as a hobbyist Ada developer, I don't think I can justify it for myself. But a cheaper event would be a great alternative. Please do keep us updated!

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Date: Fri, 22 Mar 2024 19:18:04 +0100

Hi Ada community!

This is a kind reminder that you can still submit any talks to the Ada Developer Workshop that will take place during the AEiC 2024, on the 14th of June in Barcelona!

Entry prices should be published shortly in the AEiC website. Nonetheless, we are still looking for some sponsorships :)

For more information see http://www.ada-europe.org/conference2024/adadev.html or email any of the organisers (Fabien, Dirk and Fernando).

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Date: Mon, 25 Mar 2024 23:18:42 +0100

Great news everybody! This was posted by Dirk on the Ada-Lang forum.

Hot news! Thanks to AdaCore sponsoring the Ada Developer Workshop in Barcelona, the early registration fee for in-person participation will be only 10 EUR, including lunch and coffee breaks.

That's as low-cost as attending an Ada Developer Room at FOSDEM in

Brussels, as you easily spend 10 EUR on food and drinks there… ;)

Registration info, for the conference, tutorials, workshops, social events, will shortly be added to the conference website at Ada-Europe 2024 [1].

Hope to see many of you there!

And remember, submissions are still welcome!

[1] http://www.ada-europe.org/conference2024/

Ada Monthly Meetup 2024

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Subject: Ada Monthly Meetup 2024
Date: Sun, 3 Mar 2024 20:31:05 +0100
Newsgroups: comp.lang.ada

Dear all, this is just a quick reminder that the next Ada Monthly Meetup will take place on Saturday 9th of March!

No topics were proposed for this meetup. Nonetheless, I will take the opportunity to talk a bit about FOSDEM (and WolfSSL), the newly proposed Ada Developer Workshop during AEiC, remind people about the newly released Alire v2.0-RC1 and a few other topics if we have time.

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Date: Sun, 17 Mar 2024 10:10:26 +0100

Hello everybody!

I would like to announce the April (2024) Ada Monthly Meetup which will be taking place on the 6th 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 no topics :wink:

Though I will try to focus more on Ada and I would like to bring people's attention to [Tsoding's Ada livestreams] (https://forum.ada-lang.io/t/making-a-game-in-ada-with-raylib/704).

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] (https://meet.jit.si/AdaMonthlyMeetup) 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.

Best regards and see you soon! Fer

P.S: it is that time of year when clocks have their time changed. So please, take a look at whether this affects you. (Central) Europe will now go from CET to CEST, so +2h. USA and related countries already had their time changed last week.

Ada-related Resources

[Delta counts are from February 19th to May 28th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 28 May 2024 13:23 CET
To: Ada User Journal readership

Ada groups on various social media:

- Reddit: _705 (+144) members [1]
- LinkedIn: 3_509 (+30) members [2]
- Stack Overflow: 2_405 (+12) questions [3]
- Gitter: 253 (+10) people [4]
- Ada-lang.io: 219 (+37) users [5]
- Telegram: 201 (+28) users [6]
- Libera.Chat: 75 (-1) concurrent users [7]

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

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

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

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

[5] https://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: 28 May 2024 13:33 CET
To: Ada User Journal readership

GitHub: >1_000* (=) developers [1]

Rosetta Code: 950 (+10) examples [2]

42 (+4) developers [3]

Alire: 405 (+12) crates [4]

1_048 (new) releases [5]

Sourceforge: 251 (+3) projects [6]

Open Hub: 214 (=) projects [7]

Codelabs: 57 (=) repositories [8]

Bitbucket: 38 (+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: 28 Feb 2024 13:43 CET
To: Ada User Journal readership

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

- TIOBE Index: 22 (+3) 0.83% (+0.06%) [1]
- PYPL Index: 19 (-4) 0.82% 1.08% (-0.26%) [2]
- Languish Trends: 180 (new) 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/

Certificate Error Accessing Adapower.com

From: Juanmiuk <juanmiuk@gmail.com>
Subject: Certificate Security Error when. access adapower.com
Date: Wed, 24 Jan 2024 05:30:39 -0800
Newsgroups: comp.lang.ada

When I tried to access adapower.com from the last version of Chrome and

NordVPN VPN the browser shows me this error:

Your connection isn't private. The web page you are trying to enter is not certified by a known certifying authority. Attackers might be trying to steal your information (for example, passwords, messages, or credit cards).

This error did not happen with Safari or Microsoft Edge (last version)

What's going on?

From: Stéphane Rivière <stef@genesix.org>
Date: Wed, 24 Jan 2024 16:11:42 +0100

Simply no TLS certificates (see the padlock status before the URL)

This site is in ruins, out of date and should no longer exist.

What's more, a Google search turns up some dubious links.

Ada-related Tools

NeoVim Plugin to Publish Alire Packages

From: Tama Mcglinn <t.mcglinn@gmail.com>
Subject: NeoVim plugin to publish Alire packages
Date: Sat, 17 Feb 2024 00:01:47 -0800
Newsgroups: comp.lang.ada

In case there's any NeoVim users who also publish Alire packages, I wrote a plugin for that;

https://github.com/TamaMcGlinn/nvim-alire-tools

allows you to bind or call `:AlirePublish` which handles everything for your Alire toml file, and intelligently sees where you are in the version publishing process.

Aunit.Checks

From: Simon Wright <simon@pushface.org>
Subject: AUnit.Checks
Date: Sun, 24 Mar 2024 09:19:38 +0000
Newsgroups: comp.lang.ada

Has anyone come across this package? AFAICT it doesn't appear in the AUnit repo on Github.

Even the spec would be invaluable!

From: Simon Wright <simon@pushface.org>
Date: Sun, 24 Mar 2024 11:17:06 +0000

Cancel that! It's in Stephe Leake's AUnit extensions, encountered in ada-mode.

Ada Practice

Re: Map Iteration and Modification

[Continues from AUJ 44-4, December 2023. The discussion initially addressed how to modify a container during iteration, to later move onto iteration semantics. —arm]

From: G.B. <bauhaus@notmyhomepage.invalid>
Subject: Re: Map iteration and modification
Date: Mon, 1 Jan 2024 20:27:51 +0100
Newsgroups: comp.lang.ada

>> Suppose that there is a way of orderly proceeding from one item to the next. It is probably known to the implementation of map. Do single steps guarantee transitivity, though, so that an algorithm can assume the order to be invariable?

> An insane implementation can expose random orders each time.

An implementation order should then not be exposed, right? What portable benefits would there be when another interface is added to that of map, i.e., to Ada containers for general use? Would it not be possible to get these benefits using a different approach? I think the use case is clearly stated:

First, find Cursors in map =: C*.

Right after that, Delete from map all nodes referred to by C*.

> Unless removing element invalidates all cursors. Look, insanity has no bounds. Cursors AKA pointers are as volatile as positions in certain implementations. Consider a garbage collector running after removing a pair and shuffling remaining pairs in memory.

> [...]

> you assume that cursors are ordered and the order is preserved from call to call. [...]

Yes, given the descriptions of Ordered_Maps, so long as there is no tampering, a Cursor will respect an order. Likely the one that the programmer has in mind.

[...]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 1 Jan 2024 21:55:12 +0100

> An implementation order should then not be exposed, right?

IMO, an order should be exposed. Not necessarily the "implementation order" whatever that might mean.

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 2 Jan 2024 21:15:01 -0600

>> There is no "natural" order to the key/element pairs; they are effectively unordered.

> Iteration = order. It is the same thing. If you provide iteration of pairs in the mapping by doing so you provide an order of.

Certainly not. An iteration presents all of the elements in a container, but there is no requirement that there is an order. Indeed, logically, all of the elements are presented at the same time (and parallel iteration provides an approximation of that).

If you try to enforce an order on things that don't require it, you end up preventing useful parallelism (practically, at least, no one has succeeded at providing useful parallelism to sequential code and people have been trying for about 50 years -- they were trying when I was a university student in the late 1970s).

>> [...] Certainly, no concept of "forward" or "reverse" applies to such an ordering (nor any stability requirement).

> It does. You have a strict total order of pairs which guarantees existence of previous and next pairs according to.

Again, this is unrelated. Iteration can usefully occur in unordered containers (that is, "foreach"). Ordering is a separate concept, not always needed (certainly not in basic structures like maps, sets, and bags).

[...]

Ada requires that cursors continue to designate the same element through all operations other than deletion of the element or movement to a different container. Specific containers have additional invariants, but this is the most general one. No other requirement is needed in many cases.

> Yes, position is a property of enumeration.

Surely not. This is a basis for my disagreement with you here. The only requirement for enumeration is that all elements are produced. The order is an artifact of doing an inherently parallel operation sequentially. We don't care about or depend on artifacts.

[...]

>> You have some problem with an iterator interface as opposed to an array interface??

> Yes, I am against pointers (referential semantics) in general.

This is nonsense - virtually everything is referential semantics (other than components). Array indexes are just a

poor man's pointer (indeed, I learned how to program in Fortran 66 initially, and the way one built useful data structures was to use array indexes as stand-ins for pointers). In A(1), 1 is a reference to the first component of A.

So long as you are using arrays, you are using referential semantics. The only way to avoid it is to embed an object directly in an enclosing object (as in a record), and that doesn't work for many problems.

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 2 Jan 2024 21:22:00 -0600

> Cursor is merely a fat pointer.

A cursor is an abstract reference. It *might* be implemented with a pointer or with an array index. Indeed, the bounded containers pretty much have to be implemented with an underlying array.

It would be nice if there was some terminology for abstract references that hadn't been stolen by some programming language. Terms like "pointer" and "access" and "reference" all imply an implementation strategy. That's not relevant most of the time, and many programming language design mistakes follow from that. (Anonymous access types come to mind).

From: Moi <findlaybill@blueyonder.co.uk>
Date: Wed, 3 Jan 2024 04:05:59 +0000

> It would be nice if there was some terminology for abstract references that hadn't been stolen by some programming language. [...]

What about "currency", as used in DB systems?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 3 Jan 2024 11:04:58 +0100

> Certainly not. An iteration presents all of the elements in a container, but there is no requirement that there is an order.

The meaning of the word "iterate" is doing something (e.g. visiting an element) again. That *is* an order.

> Indeed, logically, all of the elements are presented at the same time (and parallel iteration provides an approximation of that).

Parallel iteration changes nothing because involved tasks are enumerated and thus ordered as well.

> If you try to enforce an order on things that don't require it, you end up preventing useful parallelism [...]

Ordering things does not prevent parallelism. But storing cursors for later is a mother of all Sequentialisms! (:-))

Whether container elements can be effectively deleted in parallel is an interesting but rather impractical one. Nobody, literally nobody, cares because any implementation would be many times slower than the worst sequential one! (:-))

> [...] Iteration can usefully occur in unordered containers (that is, "foreach").

"An enumeration is a complete, ordered listing of all the items in a collection."

-- Wikipedia

If "foreach" exposes an arbitrary ordering rather than some meaningful (natural) one, that speaks for "insanity" but changes nothing.

> Ordering is a separate concept, not always needed

Right. But no ordering means no iteration, no foreach etc. If I can iterate, that I can create an ordered set of (counter, element) pairs. Done.

> Surely not. This is a basis for my disagreement with you here.

Then you are disagreeing with core mathematics... (:-))

> The only requirement for enumeration is that all elements are produced.

Produced in an order. Elements only produced" is merely an opaque set. Enumeration of that set is ordering its elements.

> The order is an artifact of doing an inherently parallel operation sequentially.

Yes, ordering is an ability to enumerate elements of a set. It is not an artifact it is the sole semantics of.

[...]

> So long as you are using arrays, you are using referential semantics. [...]

The key difference is that index does not refer to any element. It is container + index that do.

From the programming POV it is about avoiding hidden states when you try to sweep the container part under the rug.

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 3 Jan 2024 22:07:30 -0600

> Parallel iteration changes nothing because involved tasks are enumerated and thus ordered as well.

Nonsense. There is no interface in Ada to access logical threads (the ones created by the parallel keyword).

> Ordering things does not prevent parallelism.

Yes it does, because it adds unnecessary constraints. It's those constraints that make parallelizing normal sequential code hard. A parallelizer has to guess which ones are fundamental to the code meaning and which ones are not.

[...]

You are adding an unnecessary property to the concept of iteration. Iteration does not necessarily imply enumeration (it can, of course). Iteration /= enumeration.

[...]

Iteration is not necessarily enumeration. It is applying an operation to all elements, and doing that does not require an order. Some specific operations might require an order, and clearly for those one needs to use a data structure that inherently has an order.

> The key difference is that index does not refer to any element. It is container + index that do.

That's not a "key difference". That's exactly how one should use cursors, especially in Ada 2022. The Ada containers do have cursor-only operations, but those should be avoided since it is impossible to provide useful contracts for those operations (the container is unknown, so the world can be modified, which is bad for parallelism and understanding). Best to consider those operations obsolete. (Note that I was *always* against the cursor-only operations in the containers.)

So, using a cursor implies calling an operation that includes the container of its parameter.

> From the programming POV it is about avoiding hidden states when you try to sweep the container part under the rug.

That's easily avoided -- don't use the obsolete operations. (And a style tool like Jean-Pierre's can enforce that for you.)

> [...] Usability always trumps performance.

That's the philosophy of languages like Python, not Ada. If you truly believe this, then you shouldn't be using Ada at all, since it makes lots of compromises to usability in order to get performance.

> And again, looking at the standard containers and all these *tagged* *intermediate* objects one needs in order to do elementary things, I kind of have doubts... (:-))

The standard containers were designed to make *safe* containers with decent performance. As I noted, they're not a built-in part of the programming language, and as such have no impact on the performance of the language proper. One could easily replace them with an unsafe design to get maximum performance -- but that would have to return pointers to elements, and you've said you don't like referential semantics. So you would never use those.

You also can avoid all of the "tagged objects" (really controlled objects) by using function Element to get a copy of the element rather than some sort of reference to it. That's preferred if it doesn't cost too much for your application.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 4 Jan 2024 12:28:04 +0100

> Iteration is not necessarily enumeration. It is applying an operation to all elements, and doing that does not require an order.

That is not iteration, it is unordered listing, a totally useless thing because the result is the same unordered set.

You could not implement it without prior ordering of the elements you fed to the threads. If the threads picked up elements concurrently there would be no way to do that without ordering elements into a taken / not yet taken order. You cannot even get an element from a truly unordered set, no way! If the programmer tried to make any use of the listing he would again have to impose ordering when collecting results per some shared object.

The unordered listing is a null operation without ordering.

> [...] So, using a cursor implies calling an operation that includes the container of its parameter.

OK. It is some immensely over-designed index operation, then! (:-)) So, my initial question is back, why all that overhead? When you cannot do elementary things like preserving your indices from a well-defined set of upon deleting elements with indices outside that set?

[...]

> Specifically, the containers are separate from Ada.

Not really. Like STL with C++ it massively influenced the language design motivating adding certain language features and shifting general language paradigm in certain direction.

>> Usability always trumps performance.

> That's the philosophy of languages like Python, not Ada.

Ah, this is why Python is totally unusable? (:-))

Ada is usable and performant because of the right abstractions it deploys. If you notice performance problems then, maybe, just my guess, you are using the wrong abstraction?

> The standard containers were designed to make *safe* containers with decent performance.

Well, we always wish for the best... (:-))

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 4 Jan 2024 20:00:37 -0600

> [...]

> Ah, this is why Python is totally unusable? (:-))

I would tend to argue that it is indeed the case that you get dubious results when you put usability first. Ada puts readability/understandability, maintainability, and consistency first (along with performance). Those attributes tend to provide usability, but not at the cost of making things less consistent or understandable.

I wrote an article on this topic a year and a half ago that I wanted to publish on Ada-Auth.org. But I got enough pushback about not being "neutral" that I never did so. (I don't think discussing why we don't do things some other languages do is negative, but whatever.) I've put this on RR's blog at
http://www.rrsoftware.com/html/blog/consequences.html
so it isn't lost.

From: Simon Wright <simon@pushface.org>
Date: Fri, 05 Jan 2024 09:26:03 +0000

> http://www.rrsoftware.com/html/blog/consequences.html

Thanks for this!

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 5 Jan 2024 12:51:50 +0100

> http://www.rrsoftware.com/html/blog/consequences.html

Thanks for posting this.

I disagree with what you wrote on several points:

1. Your premise was that use = writing. To me using includes all aspects of software developing and maintenance process. Writing is only a small part of it.
2. You argue for language regularity as if it were opposite to usability. Again, it is pretty much obvious that a regular language is easier to use in any possible sense.
3. Removing meaningless repetitions contributes to usability. But X := X + Y is only one instance where Ada required such repetition. There are others. E.g.

```
if X in T'Class then
      declare
   XT : T'Class renames T'Class (X);
```

   T'Class is repeated 3 times. A discussion point is whether a new name XT could be avoided etc.

   Introducing @ for a *single* purpose contradicts the principle of regularity. I would rather have a regular syntax for most if not all such instances.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 6 Jan 2024 01:25:46 -0600

> 1. Your premise was that use = writing.

Perhaps I didn't make it clear enough, but my premise was that many people making suggestions for Ada confuse "ease-of-use" with "ease-of-writing". I said "mischaracterized" for a reason (and I see that "mis" was missing from the first use, so I just added that). "Ease-of-writing" is not a thing for Ada, and it isn't considered while the other aspects are weighed. And as I said in my last message, there is a difference in that writing more can help understandability, but it never helps writing.

[...]

> T'Class is repeated 3 times. A discussion point is whether a new name XT could be avoided etc.

Of course, this example violates OOP dogma, and some people would argue that it should be harder than following it. That's the same reason that Ada doesn't have that many implicit conversions. In this particular example, I tend to think the dogma is silly, but I don't off-hand see a way to avoid the conversion being somewhere (few implicit conversions after all).

> Introducing @ for a *single* purpose contradicts the principle of regularity.

@ is regular in the sense that it is allowed anywhere in an expression. If you tried to expand the use to other contexts, you would have to differentiate them, which would almost certainly require some sort of declaration. But doing that risks making the mechanism as wordy as what it replaces (which obviously defeats the purpose).

We looked at a number of ideas like that, but they didn't seem to help comprehension. In something like:

```
LHS:(X(Y)) := LHS + 1;
```

(where LHS is an arbitrary identifier), if the target name is fairly long, it could be hard to find where the name for the target is given, and in any case, it adds to the name space that the programmer has to remember when reading the source expression. That didn't seem to add to readability as much as the simple @ does.

In any case, these things are trade-offs, and certainly nothing is absolute. But @ is certainly much more general than ":=+" would be, given that it works with function calls and array indexing and attributes and user-defined operations rather than just a single operator.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 7 Jan 2024 16:06:10 +0100

> [...] But @ is certainly much more general than ":=+" would be [...]

For the 9X and 0X revisions I suggested adding "when <condition>" to return and raise statements, similar to its use on exit statements. This was rejected because the language already has a way to accomplish this: if statements.

Given that one can do

```
declare
   V : T renames Very_Long_Identifier;
begin
  V := V - 23;
end;
```

it seems that @ should also have been rejected. Probably more so, since @ is completely new syntax rather than reusing existing syntax on some additional statements. What is the justification of accepting @ while still rejecting the other?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 8 Jan 2024 22:46:59 -0600

> For the 9X and 0X revisions I suggested adding "when <condition>" to return and raise statements, similar to its use on exit statements.

I don't recall ever seriously considering this (might just my memory getting old). I suspect that didn't get rejected so much as not making the cut as important enough. We do try to limit the size of what gets added, not just adding everyone's favorite feature.

I'd guess that "raise Foo when Something" would get rejected now as it would be confusing with "raise Foo with Something" which means something very different. (At least the types of "Something" are different in these two.) OTOH, we added "when condition" to loops (which I thought was unnecessary, but I lost that), so arguably it would be consistent to add it to other statements and expressions as well. Perhaps you should raise it again on Github.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Tue, 9 Jan 2024 10:43:38 +0100

>I suspect that didn't get rejected so much as not making the cut as important enough.

I don't consider special syntax to shorten names in assignment statements important at all. We have renames for that, and it is a more general mechanism, applying to more than just assignments.

"Usability" (was Re: Map Iteration and Modification)

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Subject: Re: "Usability" (was Re: Map iteration and modification)
Date: Sat, 6 Jan 2024 02:54:09 -0000
Newsgroups: comp.lang.ada

> http://www.rrsoftware.com/html/blog/consequences.html

Without reading that, I would never have understood "usability" to mean "ease of writing". I learned from early on in my programming career that readability was more important than writability. So "using" a language doesn't end with writing the code: you then have to test and debug it-- basically lick it into shape--then maintain it afterwards.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 6 Jan 2024 01:03:05 -0600

> Without reading that, I would never have understood "usability" to mean "ease of writing". [...]

Usability is of course not just ease-of-writing, but a lot of people tend to co-mingle the two. For readability, too little information can be just as bad as too much. For writability, the less you have to write, the better.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 6 Jan 2024 10:14:07 +0200

> Usability is of course not just ease-of-writing, but a lot of people tend to co-mingle the two. For readability, too little information can be just as bad as too much. For writability, the less you have to write, the better.

I feel that is too narrow a definition of writability (and perhaps you did not intend it as a definition). Before one can start typing code, one has to decide what to write -- which language constructs to use. A systematically constructed, regular language like Ada makes that mental effort easier, even if it results in more keystrokes; a plethora of special-case syntaxes and abbreviation possibilities makes it harder.

Perhaps "writability" should even be taken to cover the whole process of creating /correct/ code, and include all the necessary testing, debugging and corrections until correct code is achieved. Here of course Ada shines again, with so many coding errors caught at compile time.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 6 Jan 2024 21:21:30 -0400

> Usability is of course not just ease-of-writing, but a lot of people tend to co-mingle the two.

Yes, I'm always surprised to see many languages (including Rust) praising themselves for being "concise". Apart from saving some keystrokes, I fail to see the benefit of being concise...

From: Bill Findlay <findlaybill@blueyonder.co.uk>
Date: Tue, 09 Jan 2024 15:19:52 +0000

> [...] Apart from saving some keystrokes, I fail to see the benefit of being concise...

Agreed. However, it is a bit of a totem in the FP cult.

Limited with Too Restrictive?

From: Blady <p.p11@orange.fr>
Subject: Limited with too restrictive?
Date: Sat, 13 Jan 2024 17:11:35 +0100
Newsgroups: comp.lang.ada

I want to break some unit circularity definitions with access types as for instance with record:

```
type R1;
type AR1 is access R1;
type R1 is record
   Data : Natural;
   Next : AR1;
end record;
```

In my case, I have a unit:

```
package test_20240113_modr is
   type R2 is record
      Data : Natural;
   end record;
   type AR2 is access R2;
end test_20240113_modr;
```

"limited withed" in:

```
limited with test_20240113_modr;
package test_20240113_mods is
end;
```

Let's imagine the circularity, thus PS1 and PS2 definitions are legal.

Of course the following isn't legal:

```
type AS1 is array (1..2) of
test_20240113_modr.R2; -- illegal
```

However why not with access type:

```
type AS2 is array (1..2) of
test_20240113_modr.AR2; -- illegal
```

Likewise, why not:

```
type AS3 is record
   Data : Natural;
   Next : test_20240113_modr.AR2; -- illegal
end record;
```

Isn't "limited with" too restrictive, is it?

Well, I could make some code transfers from unit to another or access conversions, that's what I actually do but at heavy cost.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 13 Jan 2024 22:31:12 -0600

> However why not with access type:

> type AS2 is array (1..2) of test_20240113_modr.AR2; -- illegal

For a limited with, one only knows the syntactic declarations (we cannot assume any analysis). Therefore, we cannot know the representation of any type, including access types.

Specifically, compilers may support multiple representations for access types, for a variety of reasons (the underlying machine has different representations, as on the 8086 and U2200 that we did compilers for; because additional data needs to be carried along to implement Ada semantics - GNAT did that for access to unconstrained arrays, and so on). The representation can depend upon aspect specifications, the designated subtype, and more, none of which is known at the point of a limited with.

We couldn't restrict implementations to a single representation for access types, and thus limited with has to treat them the same as other types.

It's necessary to declare local access types for entities that are accessed from a limited view. The reason that anonymous access types were expanded was to make that less clunky -- but I don't think it succeeded.

> Well, I could make some code transferts from unit to another or access conversions, that's what I actually do but at heavy cost.

Yup, but the alternative is worse - requiring all access types to be the most general representation (which can have a heavy performance cost).

String_Access in Unbounded String Handling?

From: Blady <p.p11@orange.fr>
Subject: String_Access in unbounded string handling?
Date: Sun, 14 Jan 2024 12:05:40 +0100
Newsgroups: comp.lang.ada

String_Access is defined in A.4.5 Unbounded-Length String Handling:

7 type String_Access is access all String;

and note:

75 The type String_Access provides a (nonprivate) access type for explicit processing of unbounded-length strings.

I wonder what String_Access is for and what could be "explicit processing"?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 14 Jan 2024 12:17:25 +0100

String_Access is a mistake that should not exist.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 14 Jan 2024 16:12:31 +0100

> String_Access is a mistake that should not exist.

Well, from one point of view, surely.

However I frequently need such a type because I in general refrain from using Unbounded_String. Now, it would be no problem to declare it as needed, except for generics! If you have generic packages like:

```
generic
   type Object_Type (<>) is private;
   type Object_Access_Type
      is access all Object_Type;
```

You want all instances to share the same String_Access. So it is conflicting. One is true, it has no place there. It should have been the package Standard or none.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 16 Jan 2024 19:24:40 -0600

> String_Access is a mistake that should not exist.

I agree with Jeffrey. Whatever reason it was initially put into the package has long since ceased to be relevant. And, as Dmitry notes, when you want such a type, it's usually because you didn't want to use Ada.Strings.Unbounded (or Bounded). So the placement is odd at best.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 16 Jan 2024 19:30:57 -0600

> ... It should have been the package Standard or none.

None for me. ;-)

One really doesn't want to put anything in Standard that isn't widely needed, as those names become hard to use in other circumstances. In particular, declarations in Standard hide anything that is use-visible with the same name, so adding something to Standard can be rather incompatible.

One could mitigate use-visibility problems by allowing more extensive overloading (for instance, of objects), but that causes rare and subtle cases where a program could change meaning without any indication. (Where a different object would be used, for instance.) That makes that too risky a change for Ada.

From: Blady <p.p11@orange.fr>
Date: Wed, 17 Jan 2024 10:54:24 +0100

Thanks for all your answers,

This is probably a very minor subject, however I submitted it:
https://github.com/Ada-Rapporteur-Group/User-Community-Input/issues/79

From: Tucker Taft <tucker.taft@gmail.com>
Date: Wed, 17 Jan 2024 05:34:12 -0800

> I wonder what String_Access is for and what could be "explicit processing"?

The idea was to support the explicit use of new String'(...), X.all, and Unchecked_Deallocation rather than the implicit use of the heap inherent in Unbounded strings. It was recognized that you need a single global access type to avoid having to do conversions all over the place. This predated the availability of stand-alone objects of an anonymous access type (aka "SAOOAAATs" ;-), but those are not universally loved either. It certainly cannot be removed now without potentially very painful disruption of existing users. It could be moved to a different package without too much disruption, but I haven't seen any groundswell of interest in doing that either.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 18 Jan 2024 19:36:59 -0600

>[...] It certainly cannot be removed now without potentially very painful disruption of existing users.

I'm dubious that there are any such users. Certainly, in the handful of cases where I needed such a type, I just declared it (strong typing, you know?) and never thought of Ada.Strings.Unbounded as being a place to find such a type already defined. It is such an odd place I doubt anyone outside of perhaps the people who defined the type ever used it.

OTOH, I agree that the compatibility impact is non-zero (anyone who did use it would have to change their code), and the benefit of removing the type at this point is close to zero (junk declarations abound in long-term Ada packages, what's one more; and certainly there is a lot of unused stuff in any particular reusable package and any particular use), so the cost-benefit ratio doesn't seem to make a change here worth it. An Ada successor language would design Ada.Strings.Unbounded rather differently (so as to be able to use string literals directly with the type) and probably would include universal character support as well, so it's hard to find an important reason to change this.

Also, I'm pretty sure we're discussed this within the ARG several times in the past, so this is well-trodden ground.

From: Blady <p.p11@orange.fr>
Date: Tue, 30 Jan 2024 16:53:22 +0100

At least, the type String_Access could be tagged as obsolescent.

Choice Must Be Static?

From: Blady <p.p11@orange.fr>
Subject: error: choice must be static?
Date: Sun, 11 Feb 2024 13:29:59 +0100
Newsgroups: comp.lang.ada

I've got the following GNAT error:

```
$ GCC -c -gnat2022 -gnatl
2024/test_20240211_static_choice.adb
GNAT 13.2.0
1. procedure test_20240211_static_choice is
2.
3. package Maps is
4. type Map_Type is private
5. with Aggregate =>  (Empty =>
   Empty_Map,
6. Add_Named => Add_To_Map);
7. procedure Add_To_Map (M : in out
   Map_Type; Key : in Integer; Value : in
    String);
8. Empty_Map : constant Map_Type;
9. private
10. type Map_Type is array (1..10) of String
     (1..10);
11. procedure Add_To_Map (M : in out
     Map_Type; Key : in
     Integer; Value : in String) is null;
12. Empty_Map : constant Map_Type :=
     [1..10 => "          "];
-- error: choice must be static
>>> error: choice must be static
```

I wonder what more static it should be. Any clue?

[Full source code removed. —arm]

From: Jeffrey R.Carter
<spam.jrcarter.not@spam.acm.org.not>
Date: Sun, 11 Feb 2024 21:56:17 +0100

I don't know what this means, but it's definitely related to the Aggregate aspect. This compiles:

```
Empty_Base : constant Map_Base :=
                  (1 .. 10 => (1 .. 10 => ' ') );
```

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 12 Feb 2024 09:12:37 +0100

Square brackets are the root of all evil! (:-))

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Mon, 12 Feb 2024 20:12:01 -0600

Looks like a compiler bug to me. The nonsense message gives that away... :-)

From: Simon Wright
<simon@pushface.org>
Date: Tue, 13 Feb 2024 11:45:17 +0000

> Looks like a compiler bug to me. The nonsense message gives that away... :-)

GCC 14.0.1 says

[...]

```
 4. type Map_Type is private
 5. with Aggregate => (Empty  =>
Empty_Map,
>>> error: aspect "Aggregate" can only be
applied to non-array type
[...]
14. Empty_Map : constant Map_Type :=
[1..10 => "                "];
>>> error: choice must be static
```

I think the first is because of ARM 4.3.5(2), "For a type other than an array type, the following type-related operational aspect may be specified"[1] and the second is a "nonsense" consequence.

[1] http://www.ada-auth.org/standards/22rm/html/RM-4-3-5.html#p2

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Tue, 13 Feb 2024 22:28:22 -0600

Ah, yes, I didn't notice that part. One cannot give the Aggregate aspect on an array type, directly or indirectly. That's because container aggregates are designed to work like array aggregates, and we didn't want visibility to determine the interpretation of an aggregate (especially where the same syntax could have a different meaning in different visibility).. Thus, there can be no point where a single type can have both array aggregates and container aggregates.

Note that record aggregates and container aggregates are always syntactically different, and thus it is OK to have both in a single location (that's one of the reasons that we adopted square brackets for container aggregates). That seemed important as the majority of private types are completed by record types, and not allowing record types in this context would be difficult to work around.

From: Blady <p.p11@orange.fr>
Date: Sat, 17 Feb 2024 09:51:39 +0100

Thanks Randy for the explanation, it helps.

In-Memory Stream

From: Drpi <314@drpi.fr>
Subject: In memory Stream
Date: Fri, 16 Feb 2024 10:41:12 +0100
Newsgroups: comp.lang.ada

I want to transfer some data between applications through a memory buffer. The buffer transfer between applications is under control. My problem is with the buffer content. I thought I'll use a Stream writing/reading in/from the memory buffer. How can I achieve this? I've found no example doing this.

Note: I use Ada 2012.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 16 Feb 2024 11:40:54 +0100

I don't know if this is what you want, but at least it is an example of using streams…

Package Storage_Streams, from Adalog's components page:
https://adalog.fr/en/components.html#Storage_Stream

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Fri, 16 Feb 2024 13:40:27 +0100

> How can I achieve this? I've found no example doing this.

It of course depends on the target operating system. You need to create a shared region or memory mapped file etc. You also need system-wide events to signal the stream ends empty or full.

Simple Components has an implementation interprocess streams for usual suspects:
http://www.dmitry-kazakov.de/ada/components.htm#12.7

> Note : I use Ada 2012.

No problem, it is kept Ada 95 compatible.

From: Pascal Obry <pascal@obry.net>
Date: Fri, 16 Feb 2024 13:49:54 +0100

AWS comes with a memory stream implementation.

https://github.com/AdaCore/aws/blob/master/include/memory_streams.ads

You may want to have a look here.

From: Simon Wright
<simon@pushface.org>
Date: Fri, 16 Feb 2024 20:19:42 +0000

A spec and body for an implementation I've had since 2008:
https://github.com/simonjwright/coldframe/blob/alire/src/common/coldframe-memory_streams.ads

https://github.com/simonjwright/coldframe/blob/alire/src/common/coldframe-memory_streams.adb

From: Drpi <314@drpi.fr>
Date: Sat, 17 Feb 2024 14:36:46 +0100

Concerning the OS and the buffer transfer mechanism, as I said, this is under control. I use Windows and the WM_COPYDATA message.

My usage is a bit special. The writing process writes a bunch of data in a memory buffer then requests this buffer to be transferred to another process by way of WM_COPYDATA. The receiving process reads the data from the "new" memory buffer. I say "new" since the address is different from the one used in the writing process (of course it cannot be the same).

The library Jean-Pierre pointed me to perfectly matches this usage. Light and easy to use. Thanks.

One enhancement I see is to manage the buffer size to avoid buffer overflow (or did I miss something?).

Thanks again to everybody.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 17 Feb 2024 15:26:45 +0100

> The library Jean-Pierre pointed me to perfectly matches this usage. Light and easy to use. Thanks.

:-)

> One enhancement I see is to manage the buffer size to avoid buffer overflow (or did I miss something?).

I don't see what you mean here... On the memory side, we are reading/writing bytes from memory, there is no notion of overflow. And the number of bytes processed by Read/Write is given by the size of Item, so no overflow either...

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 17 Feb 2024 15:28:54 +0100

You ask Windows to copy a chunk of memory from one process space into another, so yes, it is physically different memory. Different or same address tells nothing because under Windows System.Address is virtual and can point anywhere.

As you may guess it is a quite heavy overhead, not only because of copying data between process spaces, but also because of sending and dispatching Windows messages.

Note, that if you implement stream Read/Write as individual Windows messages it will become extremely slow. GNAT optimizes streaming of some built-in objects, e.g. String. But as a general case you should expect that streaming of any non-scalar object would cause multiple calls to Read/Write and thus multiple individual Windows messages.

An efficient way to exchange data under Windows is a file mapping. See CreateFileMapping and MapViewOfFile.

https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createfilemappinga

https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile

Then use CreateEvent with a name to signal states of the stream buffer system-wide. Named Windows events are shared between processes.

https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-createeventa

[This is how interprocess stream is implemented for Windows in Simple Components]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 17 Feb 2024 15:48:05 +0100

> On the memory side, we are reading/writing bytes from memory, there is no notion of overflow.

In the Simple Components there is a pipe stream.

```
type Pipe_Stream
   (Size : Stream_Element_Count)  is
    new Root_Stream_Type with private;
```

When a task writes the stream full (Size elements), it gets blocked until another task reads something out.

Another implementation

```
type Storage_Stream
    (Block_Size : Stream_Element_Count)
     is new Root_Stream_Type with private;
```

rather allocates a new block of memory. The allocated blocks get reused when their contents are read out.

From: Drpi <314@drpi.fr>
Date: Sat, 17 Feb 2024 15:56:34 +0100

> [...] As you may guess it is a quite heavy overhead [...]

In my use case, there is no performance problem. The purpose is to make an editor single instance. When you launch the editor the first time, everything is done as usual. Next time you launch the editor (for example by double clicking on a file in file explorer) the init code of the editor detects an instance of the editor is already running, transfers the command line arguments to the first instance and exits.

The buffer transfer occurs once when starting a new instance of the editor.

However, I keep your solution in mind. I might need it one day.

From: Simon Wright <simon@pushface.org>
Date: Sat, 17 Feb 2024 18:09:02 +0000

> But as a general case you should expect that streaming of any non-scalar object would cause multiple calls to Read/Write and thus multiple individual Windows messages.

Our motivation for the memory stream was the equivalent of this for UDP messages; GNAT.Sockets behaves (behaved?) exactly like this, so we buffered the result of 'Output & wrote the constructed buffer to the socket; on the other side, we read the UDP message, stuffed its contents into a memory stream, then let the client 'Input.

I can't remember at this distance in time, but I think I would have liked to construct a memory stream on the received UDP packet rather than copying the content; the compiler wouldn't let me. Perhaps worth another try.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 17 Feb 2024 19:52:00 +0100

> One enhancement I see is to manage the buffer size to avoid buffer overflow (or did I miss something?).

The purpose of this stream is to access raw memory, so there is no notion of "buffer size". It is up to you to match your (user) buffer with the memory buffer. Of course, you can add a layer with all the checks you want...

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 17 Feb 2024 22:33:17 +0100

> I can't remember at this distance in time, but I think I would have liked to construct a memory stream on the received UDP packet rather than copying the content; the compiler wouldn't let me.

UDP is a kind of thing... Basically, there is no use of UDP except for broadcasting, e.g. in LAN discovery.

In all other cases it is either TCP or multicast. Since UDP does not guarantee either delivery or ordering. It would be a huge overhead to implement reliable buffered streams on top of UDP, with sequence numbers, acknowledgements, re-sending, reordering etc.

As for taking apart a UDP packet, it is straightforward. You simply declare a stream element array of the packet size and map it on the packet using:

```
pragma Import (Ada, A);
for A'Address use UDP_Packet'Address;
```

And somewhere

```
pragma Assert (Stream_Element'Size = 8);
```

just in case...

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Sun, 18 Feb 2024 00:00:11 -0000

> The writing process writes a bunch of data in a memory buffer then requests this buffer to be transferred to another process by way of WM_COPYDATA.

I thought Windows had pipes.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Sun, 18 Feb 2024 00:02:33 -0000

> When writing in the stream, you have to care to not overflow the buffer.

With pipes, the OS takes care of this for you. Once its kernel buffer is full, further writes are automatically blocked until a reader has drained something from the buffer.

It’s called “flow control”.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 18 Feb 2024 11:06:16 +0100

> I thought Windows had pipes.

Yes it has, but very rarely used though much better designed than UNIX pipes. See https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-createnamedpipea

In general Windows has much richer and better API regarding interprocess

communication than Linux. After all Windows NT was sort of descendant of VMS, which was light years ahead of UNIX Sys V. In recent times Linux improved, e.g. they added futex stuff etc. BSD is far worse than Linux in respect of API.

From: Simon Wright <simon@pushface.org>
Date: Sun, 18 Feb 2024 10:06:46 +0000

> UDP is a kind of thing... Basically, there is no use of UDP except for broadcasting, e.g. in LAN discovery.

Worked for us, sending radar measurements p-2-p at 200 Hz

> for A'Address use UDP_Packet'Address;

OK if the participants all have the same endianness. We used XDR (and the translation cost is nil if the host is big-endian, as PowerPCs are; all the critical machines were PowerPC).

From: Björn Lundin <bnl@nowhere.com>
Date: Sun, 18 Feb 2024 12:36:54 +0100

> I thought Windows had pipes.

It does, we use it for our IPC in both Linux and Windows. Works very well. We use named pipes - where each process knows its name through via env-var At start they create a named pipe with that name

We use anonymous pipes for client communication

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 18 Feb 2024 14:02:32 +0100

> OK if the participants all have the same endianness. We used XDR [...]

I always override stream attributes and use portable formats. E.g. some chained code for integers. Sign + exponent + normalized mantissa for floats, again chained. That is all. There is no need in XDR, JSON, ASN.1 or other data representation mess. They are just worthless overhead.

Raise Expressions from AARM

From: Blady <p.p11@orange.fr>
Subject: Raise expressions from AARM.
Date: Sat, 24 Feb 2024 10:50:31 +0100
Newsgroups: comp.lang.ada

AARM Ada 2022 section 11.3 presents some uses of raise expressions including this one:
(http://www.ada-auth.org/standards/22aarm/html/AA-11-3.html)

2.a.10/4 ...

```
B : Some_Array := (1, 2, 3, others =>
   raise Not_Valid_Error);
```

What could be the use cases?

My guess: whatever the size of Some_Array (greater than 3), B is elaborated but raises Not_Valid_Error when accessing component beyond position 3:

```
type Some_Array is array
   (Positive range 1..10) of Natural;
...
B : Some_Array := (1, 2, 3, others =>
   raise Not_Valid_Error);
...
begin
X := B (2); -- OK
X := B (6); -- raises Not_Valid_Error
end;
```

Is it correct?

NB: GNAT 13.2 issues a compilation error:
>>> error: "others" choice not allowed here
see: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=113862

Thanks, Pascal.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Sat, 24 Feb 2024 11:39:08 +0100

> Is it correct?

No. This will raise the exception upon the elaboration of B.

The only use of this that I can imagine is if the length of Some_Array is 3. Then the others choice is null, so the raise expression is never evaluated. But if someone changes the definition of Some_Array to be longer, then the exception will be raised.

> NB: GNAT 13.2 issues a compilation error:

> >>> error: "others" choice not allowed here

> see: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=113862

The example in the error report has Some_Array unconstrained, in which case an others choice is not allowed. With the constrained definition given above, the aggregate is valid.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 24 Feb 2024 12:39:43 +0200

> What could be the use cases?

The point of these examples (which are only in the discussion annotation, not in the normative standard) is to discuss what is syntactically legal and why. The examples need not make practical sense.

> My guess: [...] raises Not_Valid_Error when accessing component beyond position 3:

No. A raise-expression is not a value that can be stored in an array or passed around; its evaluation raises an exception /instead/ of yielding a value.

In this example, if the evaluation of the array aggregate that initializes B evaluates the expression supplied for the "others" choice, this evaluation will raise Not_Valid_Error and disrupt the initialization of B.

It is not clear to me if the RM requires the evaluation of the "others" expression if there are no "other" indices. Experimenting with GNAT (Community 2019) shows that if the Some_Array type has 'Length = 3, the exception is not raised (so the "others" value is not evaluated), while if the 'Length is greater than 3 the exception is raised.

> type Some_Array is array (Positive range 1..10) of Natural;

> B : Some_Array := (1, 2, 3, others => raise Not_Valid_Error);

That should raise Not_Valid_Error during the initialization of B.

From: Blady <p.p11@orange.fr>
Date: Sun, 25 Feb 2024 12:09:08 +0100

If I understand well, no compiler error nor warning at compilation time but Not_Valid_Error raised at run time elaboration.

To be compared with:

```
B1 : Some_Array := (1, 2, 3);
```

No compiler error, one compiler warning "Constraint_Error will be raised at run time" and Constraint_Error range check failed raised at run time elaboration.

From: Blady <p.p11@orange.fr>
Date: Sun, 25 Feb 2024 12:23:48 +0100

> The examples need not make practical sense.

Well, despite I knew that, I wanted to draw some use cases from them.

For instance:

```
A : A_Tagged  := (Some_Tagged'
   (raise TBD_Error) with Comp => 'A');
```

It will raise TBD_Error if Some_Tagged is not a null record, good to know, isn't it?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 26 Feb 2024 22:01:23 +0200

> It will raise TBD_Error if Some_Tagged is not a null record, good to know, isn't it?

Hm, not raising the exception for a null record seems weird to me, and I cannot deduce it from the RM. Moreover, for a plain qualified expression

Some_Tagged'(raise TBD_Error)

not in an extension aggregate GNAT raises the exception even if the type is a null record. I suspect that not raising the exception for an extension aggregate where the ancestor type is a null record is a bug in GNAT.

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

April 06 **Ada Monthly Meetup 2024 April**, Internet. New edition of the monthly online meeting to gather the community, see each other, talk about some things, and let people present or showcase their work and discuss the news.

April 06-11 27th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2024), Luxembourg City, Luxembourg. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems), SPIN (Symposium on Model Checking of Software).

April 08-11 19th **European Dependable Computing Conference** (EDCC'2024), Leuven, Belgium. Topics include: hardware and software architecture of dependable systems; mixed-criticality systems design and evaluation; dependability modelling and tools; testing and validation methods; dependability and security of artificial intelligence, critical infrastructures, cyber-physical systems, (industrial) Internet of Things, ...; safety-critical system design and analysis; etc.

April 08-11 30th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2024), Winterthur, Switzerland. Theme: "Out of the Lab, into the Wild!"

April 08-12 36th ACM **Symposium on Applied Computing** (SAC'2024), Avila, Spain.

- ☺ April 08-12 **Track on Programming Languages** (PL'2024). Topics include: technical ideas and experiences relating to implementation and application of programming languages, such as compiling techniques, domain-specific languages, garbage collection, language design and implementation, languages for modeling, model-driven development, new programming language ideas and concepts, practical experiences with programming languages, program analysis and verification, etc. Deadline for submissions: October 13, 2019 (regular papers, SRC research abstracts).
- April 08-12 **Software Verification and Testing Track** (SVT'2024). Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, model checking, correct by construction development, model-based testing, software testing, static and dynamic analysis, abstract interpretation, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging, verification and validation of large scale software systems, real world applications and case studies applying software testing and verification, etc.
- April 08-12 19th **Track on Dependable, Adaptive, and Secure Distributed Systems** (DADS'2024). Topics include: Dependable, Adaptive, and secure Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; applications of DADS; etc.

April 14-20 46th **International Conference on Software Engineering** (ICSE'2024), Lisbon, Portugal.

April 14-15 12th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2024). Topics include: approaches, methods, and tools for verification and validation; formal approaches to safety and security related issues; scalability of formal method applications; integration of formal methods within the software development lifecycle; model-based engineering approaches; correctness-by-construction approaches for software and systems engineering; application of formal methods to specific domains (such as, autonomous, cyber-physical, intelligent, and IoT systems); formal methods for certification; guidelines to use formal methods in practice; usability of formal methods; etc.

April 22-25 19th **European Conference on Computer Systems** (EuroSys'2024), Athens, Greece. Topics include: 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.

April 24-25 16th **Software Quality Days** (SWQD'2024), Vienna, Austria. Theme: "Software Quality as a Foundation for Security". Topics include: all topics related to software and systems quality, such as methods and tools for constructive and analytical quality assurance; testing of software and software-intensive systems; process improvement for development and testing; automation in quality assurance and testing; domain specific quality issues such as embedded, medical, automotive systems; continuous integration, deployment, and delivery; project and risk management; secure coding, software engineering and system design; detection and prevention of vulnerabilities and security threats; etc.

May 06-10 27th **Ibero-American Conference on Software Engineering** (CIbSE'2024), Curitiba, Paraná, Brazil. Topics include: software architecture and variability; software quality, quality models and technical debt management; software reliability; software ecosystems and systems of systems; software Engineering (SE) education and training; software evolution and modernisation; SE for emerging application domains (cyber-physical systems, Internet of Things, ...); industrial experience reports in SE; software product lines and processes; software repository mining and software analytics; software processes; software reuse; software testing; etc.

May 07-11 15th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2024), London, UK.

May 11 **Ada Monthly Meetup 2024 May**, Internet. New edition of the monthly online meeting to gather the community, see each other, talk about some things, and let people present or showcase their work and discuss the news.

May 13-16 17th **Cyber-Physical Systems and Internet of Things Week** (CPS-IoT Week'2024), Hong Kong. Event includes: 5 top conferences, HSCC, ICCPS, IoTDI, IPSN, and RTAS, as well as poster and demo sessions, workshops, tutorials, competitions, industrial exhibitions, and PhD forums. Deadline for submissions: April 5, 2024 (student travel grant applications). Deadline for early registration: April 15, 2024.

☺ May 13-16 29th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2024). Topics include: time-sensitive applications; real-time and embedded operating systems; application profiling, WCET analysis, compilers, tools, benchmarks and case studies; modelling languages, modelling methods, model learning, model validation and calibration; scheduling and resource allocation; verification and validation methodologies; etc. Deadline for early registration: April 9, 2024.

May 13-16 15th ACM/IEEE **International Conference on Cyber-Physical Systems (ICCPS'2024).** Topics include: safety and resilience for CPS; software platforms and systems for CPS; specification languages and requirements; design, optimization, and synthesis; testing, verification, certification; security, trust, and privacy in CPS; tools, testbeds, demonstrations and deployments; etc.

May 27-31 38th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2024), San Francisco, California, USA. Topics include: applications to solve problems using parallel and distributed computing concepts; programming models, compilers, and runtime systems (ranging from the design of parallel programming models and paradigms to languages and compilers supporting these models and paradigms, to runtime and middleware solutions); system software; existing and emerging architectures; experiments and performance-oriented studies in the practice of parallel and distributed computing; etc.

May 27-31 17th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2024), Toronto, Canada. Topics include: manual testing practices and techniques, security testing, model-based testing, test automation, static analysis and symbolic execution, formal verification and model checking, software reliability, testability and design, testing and development processes, testing in specific domains (such as embedded/cyber-physical systems, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc.

June 04-06 16th **NASA Formal Methods Symposium** (NFM'2024), Moffett Field, California, USA. Topics include: identifying challenges and providing solutions towards achieving assurance for critical systems; formal techniques for software and system assurance for applications in space, aviation, robotics, and other NASA-relevant safety-critical systems.

June 10-12 21st **International Conference on Software and Systems Reuse** (ICSR'2024), Limassol, Cyprus. Theme: "Sustainable Software Reuse". Topics include: new and innovative research results and industrial experience reports dealing with all aspects of software reuse within the context of the modern software development landscape, such as technical aspects of reuse (model-driven development, variability management and software product lines, domain-specific languages, new language abstractions for software reuse, software composition and modularization, technical debt and software reuse, ...), software reuse in industry and in emerging domains (reuse success stories, reuse failures and lessons learned, reuse obstacles and success factors, return on investment studies, ...).

☺ June 11-12 12th **European Congress on Embedded Real Time Systems** (ERTS'2024), Toulouse, France. Topics include: all aspects of critical embedded real-time systems, such as model-based system and safety engineering, product line engineering, programming languages, verification methods, software development frameworks, dependability, safety, cyber security, quality of service, fault tolerance, maintainability, certification, etc. Deadline for submissions: April 3, 2024 (regular papers), May 5, 2024 (final short and regular papers).

June 11-14 28th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC'2024), Barcelona, Spain. Organized by Ada-Europe and Barcelona Supercomputing Center (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. Deadline for early registration: May 20, 2024. #AEiC2024 #AdaEurope #AdaProgramming

June 14 **Ada Developers Workshop**. Topics include: everything related to Ada software development, i.e. similar to the Ada DevRooms at FOSDEM, technical presentations, tutorials, demos, live performances, project status reports, discussions, etc, offering a place where the Ada community can meet and share their work and projects.

☺ June 14 9th **Workshop on Challenges and New Approaches for Dependable and Cyber-Physical System Engineering** (De-CPS'2024). Topics include: artificial intelligence for CPS; model-based system engineering for CPS; transport and mobility, vehicle of the future; Industry 4.0 / 5.0; IoT, edge and cloud continuum; digital twins; safety and (cyber)security; human/machine interaction; real-time computing; time-sensitive networking (TSN), 5G/6G networks. Deadline for submissions: April 30, 2024 (papers).

June 14 3rd **ADEPT workshop, AADL by its practitioners** (ADEPT'2024). Topics include: current projects in the field of design, implementation and verification of critical systems where AADL is a first-citizen technology.

June 17-21 19th **International Federated Conference on Distributed Computing Techniques** (DisCoTec'2024), Deadline for submissions: May 6, 2024 (workshop papers). Groningen, the Netherlands. Topics include: a broad spectrum of distributed computing subjects, from theoretical foundations and formal description techniques, testing and verification methods, to language design and system implementation approaches. Events include: FORTE (Formal Techniques for Distributed Objects, Components and Systems), COORDINATION (Coordination Models and Languages), DAIS (Distributed Applications and Interoperable Systems), ICE (Interaction and Concurrency Experience Workshop).

June 19-21 28th **International Conference on Engineering of Complex Computer Systems** (ICECCS'2024), Limassol, Cyprus. Topics include: all areas related to complex computer-based systems, including the

causes of complexity and means of avoiding, controlling, or coping with complexity, such as model-driven development, security, reliability and dependability, safety-critical and fault-tolerant architectures, formal methods, verification and validation, reverse engineering and refactoring, software architecture, agile methods, cyber-physical systems and Internet of Things (IoT), industrial case studies, etc.

☺ June 24-28 — 25th ACM SIGPLAN/SIGBED **International Conference on Languages, Compilers, Tools and Theory of Embedded Systems** (LCTES'2024), Copenhagen, Denmark. Topics include: programming language challenges (domain-specific languages; features to exploit multicore architectures; features for distributed and real-time control embedded systems; capabilities for specification, composition, and construction of embedded systems; language features and techniques to enhance reliability, verifiability, and security; compiler challenges; ...), interaction between embedded architectures, operating systems, and compilers (support for enhanced programmer productivity; support for enhanced debugging, profiling, and exception/interrupt handling; optimization for low power/energy, code/data size, and real-time performance; tools for analysis, specification, design, and implementation; hardware, system software, application software, and their interfaces; distributed real-time control; system integration and testing; run-time system support for embedded systems; support for system security and system-level reliability; ...), predictability of resource behavior: energy, space, time (validation and verification, in particular of concurrent and distributed systems; formal foundations of model-based design as the basis for code generation, analysis, and verification; ...), design and implementation of novel architectures (architecture support for new language features, virtualization, compiler techniques, debugging tools; ...), etc.

Jun 29 – Jul 04 — 24th **International Conference on embedded computer Systems: Architectures, MOdeling and Simulation** (SAMOS'2024), Samos Island, Greece. Topics include: advances in systems efficiency in various domains; software tools, compilation techniques and optimizations, and code generation for reconfigurable architectures; embedded parallel systems application-level resource management of multi-core architectures; specification languages and models; system-level design, simulation, and verification; profiling, measurement and analysis techniques (design for) system adaptivity; testing and debugging; etc. Deadline for submissions: April 1, 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. Deadline for submissions: April 1, 2024 (workshop papers), April 6, 2004 (fast abstracts, industry track, posters).

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. Deadline for submissions: May 9, 20024 (workshop contributions), May 27, 2024 (Industrial Challenge solutions).

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. Deadline for submissions: April 29, 2024 (workshop papers).

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 submissions: April 4, 2024 (journal-first papers), April 7, 2024 (2nd round: Improved and new short papers, posters, tools).

☺ 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. Deadline for submissions: May 6, 2024 (workshop papers), May 17, 2024 (posters, demos, PhD symposium).

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. Deadline for submissions: May 5, 2024 (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. Deadline for submissions: May 8, 2024 (abstracts), May 15, 2024 (papers).

☺ 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. Deadline for submissions: April 25, 2024 (abstracts), May 1, 2024 (papers).

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 submissions: April 15, 2024 (abstracts), April 19, 2024 (full papers, tutorial papers), June 17, 2024 (artifact abstracts), June 24, 2024 (artifacts).

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: May 8, 2024 (abstracts), May 15, 2024 (papers), July 3, 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. Deadline for submissions: April 26, 2024 (papers).

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: April 10, 2024 (abstracts), April 15, 2024 (papers).

☺ 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: April 17, 2024 (submissions round 2), April 23, 2024 (artifacts round 2).

☺ 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.

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. Deadline for submissions: June 17, 2024.

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: June 2, 2024 (Work-in-Progress track papers, Late Breaking track papers).

☺ 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. Deadline for submissions: April 1, 2024 (papers).

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. Deadline for submissions: May 14, 2024 (papers).

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: May 13, 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. Deadline for submissions: May 5, 2024 (papers), May 12, 2024 (artifacts).

☺ 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).

☺ Oct 20-25 **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2024). Topics include: all practical and theoretical investigations of programming languages, systems and environments, targeting any stage of software development, including requirements, modelling, prototyping, design, implementation, generation, analysis, verification, testing, evaluation, maintenance, and reuse of software systems; development of new tools, techniques, principles, and evaluations. Deadline for submissions: April 5, 2024 (round 2).

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. Deadline for submissions: April 19, 2024 (papers).

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. Deadline for submissions: May 24, 2024 (regular research papers).

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: Mar 6, 2024 (workshops), May 31, 2024 (research abstracts), Jun 7, 2024 (research papers), Jun 26, 2024 (tool demos), Jul 2, 2024 (journal-first papers), Jul 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: May 3, 2024 (abstracts), May 10, 2024 (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. Deadline for submissions: June 7, 2024 (abstracts), June 14, 2024 (papers).

☺ Nov 07-08 32nd **International Conference on Real-Time Networks and Systems** (RTNS'2024), Porto, Portugal. Deadline for submissions: June 5, 2024 (abstracts 2nd round), June 7, 2024 (papers 2nd round), 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. Deadline for submissions: June 3, 2024 (abstracts), June 10, 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: May 23, 2024 (papers).

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

Advance Information

BARCELONA
AEiC 2024

The 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) returns to Spain, for the first time in Barcelona, from the 11th to the 14th of June. The conference 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 Ada and Reliable Software in general, into academic education and research, and industrial practice.

The conference is an established international forum for providers, practitioners and researchers in reliable software technologies. The conference presentations will illustrate current work in the theory and practice of developing, running and maintaining challenging long-lived, high-quality software systems for a variety of application domains including manufacturing, robotics, avionics, space, transportation. The conference schedule comprises a keynote and an invited talk, a panel with invited speakers, a journal track, an industrial track, a work-in-progress track, parallel tutorials, a hackathon, and satellite workshops. Participants include practitioners and researchers from industry, academia and government organizations active in the promotion and development of reliable software. The conference program includes two core days with special sessions featuring presentations of invited experts, peer-reviewed academic papers, industrial presentations, and work-in-progress talks.

Invited Speakers

- Keynote Talk on "*Strategies to build safety relevant high-performance HW/SW platforms for critical embedded systems",* by **Francisco J. Cazorla** and **Jaume Abella,** Barcelona Supercomputing Center, Spain
- Panel on "*AI for Safety-Critical Systems: How 'I' Should the AI be?*", with **Kerstin Bach**, Norwegian University of Science and Technology, Norway, **Irune Yarza,** Ikerlan, Spain, **Marta Barroso,** Barcelona Supercomputing Center, Spain, moderated by **Cristina Seceleanu**, Mälardalen University, Sweden
- Invited Talk on "*Simplifying the life-cycle management of complex application workflows",* by **Rosa Badia,** Barcelona Supercomputing Center, Spain
- Invited Talk on "*The Compute Continuum: An Efficient Use of Edge-to-Cloud Computing Resources",* by **Eduardo Quiñones,** Barcelona Supercomputing Center, Spain

Tutorials and Hackathon

The following tutorials will take place on Tuesday, June 11th:

- "Lock-Free Programming in Ada-2022: Implementing a work-stealing scheduler for Ada-2022's light-weight parallelism", **S. Tucker Taft**, AdaCore
- "Ada for Business Applications", **Gautier de Montmollin**, Ada Switzerland
- "Rust Fundamentals" and "Concurrency and Parallelism in Rust", **Luis Miguel Pinho** and **Tiago Carvalho**, ISEP, Portugal
- "Modeling Concurrent State Machines in TLA+", **J. Germán Rivera**, Tesla
- "Introduction to the Development of Safety Critical Software", **Jean-Pierre Rosen**, Adalog, France
- "METASAT: Programming High Performance RISC-V Technologies for Space", **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", **Leonidas Kosmidis**, Barcelona Supercomputing Center, **Rod Burns**, Codeplay/Intel, **Verena Beckham**, Codeplay/Intel

Also on Tuesday, June 11th, the conference will host an hackathon on "Optimizing AI-driven workflows within a mission-critical cyber-physical system", organized by Damien Gratadour, CNRS – Observatoire de Paris, France.

Co-Located Workshops

On Friday, June 14th there will be 4 workshops:

- 9th DeCPS workshop on “Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering”
- 3rd ADEPT: AADL by its practitioners
- Safe AI: Enabling the use of AI in Safety-Critical Systems
- Ada Developers Workshop

Technical Presentations

The conference technical days will provide six sessions of presentations related to the conference tracks, including presentations of the 13 research articles which have passed the first round of reviews in the journal track, 5 presentations of industrially relevant research and development, and 8 work-in-progress presentations. Please see the conference website for information.

Social Events

The program includes long coffee breaks, providing the opportunity for participants to discuss their work, and network. Lunches will be served at the conference location, from Tuesday to Friday, providing further interaction opportunities. Furthermore, there will be a Welcome Reception event, a Conference Banquet, and a post-conference chill-out event.

The welcome reception is scheduled for Tuesday, June 11, starting at 17:45 at the Barcelona Supercomputing Centre (BSC). The attendees will have the opportunity to visit (in groups) the new Marenostrum 5, inaugurated in December 2023, a supercomputer on track to reach its maximum capacity of 311.95PFLOPS. In parallel to the visit, drinks and finger food will be provided. Afterwards, the reception will continue in the nearby “Jardins de l'abadessa" restaurant, for a cocktail dinner.

The conference banquet is scheduled for Wednesday, at the emblematic restaurant "7 portes". Attendees will have the opportunity to savor the finest flavors of the Catalan and Mediterranean cuisines. Among the culinary delights awaiting you is the renowned "Paella Perallada", a masterpiece that harmoniously combines semi-dry rice with succulent peeliled 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.

The organization also prepared a chill out event as a culmination party for this year's main conference. The event is scheduled for Thursday, at the Moritz Barcelona Brewery, the brewery of the first beer of Barcelona.

Conference Venue

The conference will take place in the UPC Campus Nord. Workshops and tutorials will be hosted in the BSC-Repsol Building. Functioning since 2021, this is a research infrastructure housing more than 500 workers and the recently inaugurated MareNostrum 5 supercomputer. The main conference will be hosted in the Vèrtex UPC Building, in the "Sala d'Actes" (or Conference Hall). This building offers a series of classrooms where both training and congresses are held.

Barcelona is a city renowned for its vibrant culture and rich history. Nestled along the picturesque Mediterranean coast, Barcelona offers not only a breathtaking backdrop but also a dynamic hub for academic exchange. With its world-class research institutions and cutting-edge facilities, like the Barcelona Supercomputing Center (BSC), the ALBA Synchrotron light facility, the Barcelona Biomedical Research Park (PRBB), or the Barcelona Science Park (PCB), Barcelona embodies innovation and excellence in the scientific realm.

From the iconic architecture of Antoni Gaudí to the bustling streets of the Gothic Quarter, delegates attending AEiC2024 can immerse themselves in a blend of tradition and modernity. Moreover, the city's renowned culinary scene and vibrant nightlife provide opportunities for networking and cultural exploration.

Join Ada-Europe!

Become a member of Ada-Europe and **support Ada-related activities** and the future **development of the Ada programming language**.

Membership benefits include **receiving the quarterly Ada User Journal** and a substantial **discount when registering for the annual Ada-Europe conference**.

To apply for membership, visit our web page at

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

ADEPT 2023 Workshop Summary

Hai Nam Tran, Frank Singhoff
University of Brest, Lab-STICC UMR CNRS 6285, Brest, France; email: firstname.lastname@univ-brest.fr

Jérôme Hugues
Software Engineering Institute, Carnegie Mellon University, USA; email: jhugues@andrew.cmu.edu

Pierre Dissaux
Ellidiss Technologies, 24 quai de la douane, 29200 Brest, France; email: pierre.dissaux@ellidiss.com

Bruce Lewis, Hazel Shackleton, Joseph Kiniry, Frank Zeyda
Galois, Inc., USA; email: firstname.lastname@galois.com

Rakshit Mittal, Dominique Blouin, Anish Bhobe, Laurent Pautet
LTCI, Telecom Paris, Institut Polytechnique de Paris, Palaiseau, France; email: firstname.lastname@telecom-paris.fr

Kyungmin Bae
Pohang University of Science and Technology, South Korea; email: kmbae@postech.ac.kr

Peter Csaba Ölveczky
University of Oslo, Norway; email: peterol@ifi.uio.no

Brian R Larson
Multitude Corporation; email: brl@multitude.net

Ehsan Ahmad
Saudi Electronic University; email: e.ahmad@seu.edu.sa

Leonidas Kosmidis
Barcelona Supercomputing Center (BSC) and Universitat Politecnica de Catalunya (UPC); email: leonidas.kosmidis@bsc.es

Hugo Valente, Miguel A de Miguel, Ángel G Pérez, Alejandro Alonso, Juan Zamorano, Juan A de la Puente
Universidad Politécnica de Madrid, Madrid, Spain; email: firstname.lastname@upm.es

Abstract

The Architecture Analysis and Design Language (AADL) is a SAE standard for modeling both hardware and software architecture of embedded systems. Widely embraced by stakeholders in critical real-time embedded systems, the AADL standard is used to address a large set of concerns including performances (latency, schedulability), safety, and security. The ADEPT workshop aims to present and report on current projects in the field of design, implementation, and verification of critical real-time embedded systems where AADL is a first-citizen technology. This article is a summary of the second edition of the workshop in 2023.

Keywords: AADL, critical embedded real-time systems, design, implementation and verification

1 Introduction

The Architecture Analysis and Design Language (AADL) is a SAE standard for modeling both hardware and software architecture of embedded systems. [1]. The AADL standard is now a mature standard for modeling critical real-time embedded systems. It is employed by numerous stakeholders in the domain to address a large set of concerns: safety [2], security [3], or performance (latency, schedulability) [4] but also code generation [5, 6]. One key strength of AADL as a language is the set of tools that provide those analysis capabilities.

The ADEPT workshop aims to present and report on current projects in the field of design, implementation and verification of critical real-time embedded systems where AADL is a first citizen technology. It is also an opportunity for AADL beginners to meet experienced AADL practitioners.

In 2023, the workshop was a full day workshop. It was co-located with the 27th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2023) at Lisbon, Portugal. The workshop gathered more than 20 participants, with 10 presentations.

2 Workshop Program

The workshop was organized in 4 sessions: (1) an introduction from the workshop organizers about the AADL standard, its ecosystem and the ongoing standardization activities, (2) a session about formal methods, (3) a session about TASTE, and (4) a session about model driven engineering (MDE).

2.1 Formal methods session

In this session, workshop participants present two approaches to integrate formal methods in the Open Source AADL Tool Environment (OSATE) [7], which is the reference tool for AADL. It is released under the Eclipse Integrated Development Environment.

In [8], the authors tackle the problem of specifying synchronous designs of cyber-physical systems (CPS) in AADL. Three synchronous subsets of AADL, namely Synchronous AADL, Multirate Synchronous AADL, and HybridSynchAADL, have been defined and integrated into OSATE. Formal model checking analysis of synchronous AADL models is also integrated in the tool. It allows system designers to develop synchronous systems/design in OSATE and formally verify them without leaving the tool or having to know formal methods.

In [9], the authors present the BLESS methodology which creates programs together with deductive proofs that every possible program execution will conform to its specification. This methodology applies to an architectural model of CPS using AADL. It has been demonstrated to prove the moment authority scenario of the Chinese Train Control System Level 3 [10]. The BLESS IDE is designed as a plugin that can be integrated in OSATE.

The session continues with two presentation focusing on the mechanization of the semantic of AADL in interactive theorem provers such as Coq [11] and Isabelle/HOL [12]. The objective is to provide unambiguous formal semantics for AADL. As part of Galois's Rigorous Digital Engineering process, formal specifications in Unified Theories of Programming (UTP) are also developed for AADL and other languages to allow formal cross verification across multiple system specifications.

2.2 TASTE session

In this session, workshop participants have presented the usage of TASTE in their projects. TASTE [13] is a model-based toolset dedicated to embedded, real-time systems and was created under the initiative of the European Space Agency back in 2008, after the completion of the FP6 project ASSERT.

In [14], the authors propose a solution focused on integrating cFS - a Publisher Subscriber runtime made by NASA - inside TASTE. As TASTE relies on AADL for the automatic code generation, the current property set has been extended with new properties to support the new functionalities.

In [15], the authors present the usage of TASTE in the METASAT project. In this project, the code generation capabilities from AADL models in TASTE will be extended to (1) cover support of the RISC-V compilation and emulation infrastructure and (2) support multicore using RTEMS SMP and Xtratum partitions. Integration with parallel programming models such as SPARROW SIMD intrinsics, OpenMP and at least one GPU programming API and Machine Learning framework will be implemented. All these modifications will be submitted for inclusion in the official TASTE repository.

2.3 MDE session

In this session, workshop participants discuss several approaches to improve the usage of AADL in the MDE process.

In [16], the authors tackle the view-update problem between Declarative and Instance models in OSATE. In the current OSATE, Instance models can be updated manually or by tools. However, there are no means to reflect the changes back to the Declarative model. Providing an automated Instance model Deinstantiation capability will be very beneficial since it will allow tools to process the simpler Instance model directly. The authors demonstrates a novel OSATE-based Declarative-Instance Mapping tool (OSATE-DIM) for incremental deinstantiation of AADL models.

In [17], the authors address the information preservation when multiple Domain Specific Modeling Languages (DSML) are used in the development process. Models are described in DSMLs that can capture the different but complementary aspects of a system. Thus we have to use multiple DSMLs to describe all the aspects of the entire system. However, there is a significant overlap in information captured by the many aspects of the system. When a model is edited, any changes in the information in the overlapping part must be propagated to the other models. A model synchronization approach based on Triple Grammar Graph to handle the synchronization between models in AADL and FACE - Future Airborne Capabilities Environment [18] is presented.

In [19], the authors present LAMP an introspective analysis and processing framework for AADL. With LAMP, exploration, verification, transformation or any other processing rules are directly embedded inside the AADL model as annex subclauses. The framework leverages the Prolog language to propose a powerful and flexible solution to implement online model exploration and processing features. Thus, it does not require the specification of a complete language syntax as in dedicated language for online processing AADL models like REAL [20] or RESOLUTE [21].

In [22], the authors present a SysML V1 to AADL Bridge which allows users to annotate portions of SysML models using an AADL Profile and automatically translate such portions into AADL. It provides a bidirectional workflow, which allow generating AADL models from SysML and then update the SysML model from modified AADL ones. The bridge will also create a SysMLv1 model from an AADL specification for bottom up/re-engineering/component based development.

3 Conclusion

AADL is a set of SAE international standards that aims to improve the quality of the critical real-time embedded systems design. The objective of the ADEPT workshop was to encourage discussion between members of the AADL community and to provide location to share experiences on AADL and its ecosystem.

Acknowledgments

We would like to thank Ellidiss Technologies who supported the registration fees for the speakers of the workshop.

References

[1] S. I. (Society), *Architecture Analysis & Design Language (AADL) AS5506C*. SAE International, 2017.

[2] J. Delange and P. Feiler, "Architecture fault modeling with the AADL error-model annex," in *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 361–368, IEEE, 2014.

[3] R. Ellison, A. Householder, J. Hudak, R. Kazman, and C. Woody, "Extending AADL for security design assurance of cyber-physical systems," *CMU/SEI Report*, 2015.

[4] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Scheduling and memory requirements analysis with aadl," *ACM SIGAda Ada Letters*, vol. 25, no. 4, pp. 1–10, 2005.

[5] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina AADL tool suite," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 4, pp. 1–25, 2008.

[6] F. Cadoret, E. Borde, S. Gardoll, and L. Pautet, "Design patterns for rule-based refinement of safety critical embedded systems models," in *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pp. 67–76, IEEE, 2012.

[7] P. Feiler, "Open source aadl tool environment (OSATE)," in *AADL Workshop, paris*, pp. 1–40, 2004.

[8] K. Bae and P. Ölveczky, "Formal model engineering of synchronous CPS designs in AADL," *Ada User journal*, vol. 45, no. 1, pp. 31–34, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[9] B. R. Larson and E. Ahmad, "BLESS behavior correctness proof as convincing verification artifact," *Ada User journal*, vol. 45, no. 1, pp. 35–46, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[10] M. of Railways, "CTCS level 3 train control system requirements specification (srs) vi. o [m]," 2009.

[11] J. Hugues, "Mechanizing AADL in coq – extended abstract," *Ada User journal*, vol. 45, no. 1, pp. 47–50, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[12] J. Kiniry and F. Zeyda, "Formalizing AADL in the unifying theories of programming." Presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[13] M. Perrotin, E. Conquet, J. Delange, and T. Tsiodras, "TASTE: An open-source tool-chain for embedded system and software development," in *Embedded Real Time Software and Systems (ERTS2012)*, 2012.

[14] H. Valente, M. A de Miguel, Á. G. Pérez, A. Alonso, J. Zamorano, and J. A de la Puente, "Extension of the TASTE toolset to support publisher-subscriber communication," *Ada User journal*, vol. 45, no. 1, pp. 51–53, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[15] L. Kosmidis, "METASAT's model based design solutions," *Ada User journal*, vol. 45, no. 1, p. 54, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[16] R. Mittal and D. Blouin, "Facilitating AADL model processing and analysis with OSATE-DIM," *Ada User journal*, vol. 45, no. 1, pp. 55–58, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[17] D. Blouin, A. Bhobe, and L. Pautet, "Challenges in model synchronization for information preservation illustrated with the FACE and AADL standards," *Ada User journal*, vol. 45, no. 1, pp. 63–66, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[18] "Future airborne capabilities environment." `https://www.opengroup.org/face`. Accessed: 2024-04-15.

[19] P. Dissaux, "LAMP: to shed light on AADL models," *Ada User journal*, vol. 45, no. 1, pp. 59–62, 2024. Also presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

[20] O. Gilles and J. Hugues, "Expressing and enforcing user-defined constraints of aadl models," in *2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pp. 337–342, IEEE, 2010.

[21] A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen, "Resolute: an assurance case language for architecture models," *ACM SIGAda Ada Letters*, vol. 34, no. 3, pp. 19–28, 2014.

[22] H. Shackleton, "Bidirectional translation of SysML v1 to AADL." Presented in the ADEPT 2023 workshop, Lisbon, Portugal, 2023.

Formal Model Engineering of Synchronous CPS Designs in AADL

Kyungmin Bae
Pohang University of Science and Technology, South Korea; email: kmbae@postech.ac.kr
Peter Csaba Ölveczky
University of Oslo, Norway; email: peterol@ifi.uio.no

Abstract

Many cyber-physical systems (CPSs)—such as aircrafts, cars, robots, and manufacturing plants—have synchronous designs and are realized on platforms with bounded network delays and clock skews. This paper summarizes how we have: (i) defined modeling languages for synchronous CPS designs in the embedded systems modeling standard AADL, and (ii) integrated Maude-based formal model checking ("push-button") analysis of such AADL synchronous designs into the OSATE tool environment for AADL. This enables a "formal model engineering" approach which combines the convenience of domain-specific modeling with automatic "under-the-hood" formal analysis. Furthermore, by the PALS synchronizers, the correctness of such synchronous designs implies the correctness of the much more complex and harder-to-analyze asynchronous implementations, greatly simplifying the task of designing and analyzing "virtually synchronous" CPSs.

1 Introduction

A cyber-physical system (CPS) is a collection of "controller" components ("cyber") that interact with each other and with physical environments ("physical"). Many CPSs, such as avionics and automotive systems, networked medical devices, industrial manufacturing plants, and other distributed control systems are *virtually synchronous*: The underlying design of the system is *synchronous*; i.e., in each iteration of the system, all components should *in lockstep* read inputs from the previous iteration, change their local states, and produce output (for the next iteration). A sound design principle for such virtually synchronous CPSs should be to develop a correct *synchronous* design, *before* this task is made *much harder* by considering deployments on actual networks.

Such a design principle raises the following questions:

1. What modeling languages and modeling environments are suitable for domain experts to develop these underlying synchronous CPS designs?
2. How can we ensure that these designs are correct?
3. How can we ensure that the *deployed system* is correct?

In the body of work that we summarize in this paper, we address Question (1) as follows: The *Architecture Analysis and Design Language* (AADL) [1] is an industrial modeling standard for the classes of CPSs that we target (avionics, automotive, and other cyber-physical systems), developed and used by entities such as Carnegie Mellon University, US Army, Honeywell, Rockwell Collins, Lockheed Martin, General Dynamics, Airbus, the European Space Agency, Dassault, EADS, Ford, and Toyota. AADL also has a good open-source modeling environment, OSATE. However, (i) in AADL a system is modeled as software components mapped onto hardware platforms, and thereby models the deployed (asynchronous) system instead of its "underlying synchronous design," and (ii) AADL does not support specifying continuous physical environments, which also hinders our ability to simulate/analyze the models.

We address these last two issues by identifying a "synchronous subset" of AADL (and its Behavior Annex), where AADL constructs keep their standard intuitive meaning, and define an AADL property set—including properties for specifying continuous environment behaviors—which should be suitable to define the synchronous designs of CPSs in AADL.

To address Question (2), we integrate automatic *formal model checking* analysis of such *HybridSynchAADL* models into OSATE. This enables a *formal model engineering* methodology in which the AADL modeler develops her designs in AADL, specifies requirements of her design in an intuitive "AADL-friendly" way, and gets powerful formal analysis *for free*, without leaving OSATE or having to know formal methods.

This is made possible for HybridSynchAADL—with its expressive control programming language combined with continuous behaviors—using the expressive rewriting-logic-based Maude language and tool combined with SMT solving. In addition, benchmarking shows that such Maude-with-SMT-based analysis in many cases outperforms state-of-the-art hybrid systems reachability tools (such as HyComp) [2].

In this way, the modeler can develop correct "synchronous" designs of her CPSs. However, these designs have to be realized in a messy "real" world, with asynchronous communication, message delays, and imprecise local clocks, so Question (3) still remains: Is the deployed system correct as well? CPSs such as cars, airplanes, factories, and so on,

typically run on networks where the communication delay is bounded; clock synchronization is also well understood. With colleagues at the University at Illinois and Rockwell Collins we have therefore developed the PALS ("physically asynchronous, logically synchronous") family of *synchronizers for CPSs* where the network delays and clock skews are bounded. PALS takes a synchronous design SD—without message delays, asynchronous, imperfect clocks, etc.—and infrastructure performance bounds Γ, and provides the corresponding asynchronous "realization" $PALS(SD, \Gamma)$, so that SD and $PALS(SD, \Gamma)$ satisfy the same properties [3]. PALS therefore reduces the very complicated task of designing and verifying a distributed CPS to the *much* simpler task of designing and verifying its underlying synchronous design; furthermore, HybridSynchAADL allows these latter tasks to be performed using AADL inside OSATE.

The benefit of the PALS methodology is significant: PALS was motivated by a seemingly simple, yet hard to design and verify, avionics system. Even with perfect local clocks and no message delays, model checking the asynchronous model took 35 minutes; when message delays could be either 0 or 1, automatic model checking was unfeasible. However, the "PALS-equivalent" underlying synchronous design could be model checked in much less than a second.

2 Overview

2.1 The PALS family of synchronizers for CPSs

PALS. Although many CPSs are *virtually synchronous*, i.e., have an underlying synchronous design, they have to be realized in a distributed setting, with asynchronous message communication, message delays, execution times, imprecise local clocks, etc. This makes their *design* very challenging, and model checking verification quickly becomes unfeasible due to the state space explosion caused by asynchrony. This is the case even when the underlying infrastructure guarantees bounds on network delays, as in aircrafts, cars, factories, etc.

Motivated by a simple avionics *active standby* system (which of two *cabinets* is the active one?) developed by Rockwell Collins, which was surprisingly hard to get right and model check, together with colleagues at the University of Illinois and Rockwell Collins we developed the PALS formal *design and verification pattern* [3,4]. The active standby system is a simple virtually synchronous CPS: in each round, the three components (the two cabinets and their environment, which includes the pilot) together send 10 boolean-valued messages. Before even accounting for the difficulty of ensuring that messages are read in the correct round (which requires buffering and backoff timers), the 10 messages sent in one round can be received in 10! (> 3.6 million) different orders.

The key thing with PALS is that if the infrastructure has bounds Γ on the network delays, clock skews, and execution times, it is sufficient to *specify* and *verify* the idealized underlying *synchronous design* SD. We prove in [3] that if the simple synchronous design SD satisfies a temporal logic property Φ, then so does the distributed "realization" $PALS(SD, \Gamma)$, as long as Γ satisfies the *PALS constraint*. PALS therefore allows us to abstract from asynchronous communication, message delays, clock skews, and execution times. In the active standby system, this reduced the reachable state space from 3,047,832 in the simplest possible distributed setting—with no message delays, executions times, or clocks skews—to a mere 185 reachable states in the synchronous model [3].

Multirate PALS. A CPS may be composed of different "off-the-shelf" components whose controllers operate with different frequencies, yet need to synchronize. A prototypical example is turning an airplane, whose aileron and rudder controllers operate at different frequencies (e.g., 30-100 Hz and 30-50 Hz for, respectively, commercial rudder and aileron controllers), yet need to synchronize to achieve a safe turn. We therefore developed the *Multirate PALS* pattern for such systems [5]. One challenge is that a "fast" controller executes $k > 1$ rounds—reading inputs and producing outputs in each of them—while a slow component with which it communicates only executes one round in the same time. User-defined *input adaptors*, which generate k inputs from one output, and vice versa, allow us to compose different-rate controllers into a "synchronous" (single-rate-equivalent) design [5]. We applied the Multirate PALS design and verification methodology on a textbook algorithm for turning an aircraft, albeit with Euler approximations of the continuous dynamics [6].

Hybrid PALS. The airplane turning system underscores that many CPSs are *hybrid* systems having physical environments with *continuous dynamics*. Controllers sample and actuate such continuous environments at times defined by their imprecise local clocks. We can therefore no longer abstract from clock skews, since the precise sampling and actuation times matter. In the synchronous abstractions we must analyze the continuous behaviors for all possible samplings and actuation times based on the imprecise local clocks. *Hybrid PALS* [7] extends PALS to this setting, and has been used to specify and formally analyze the airplane turning algorithm and networked controllers for water tanks and thermostats.

2.2 The Synchronous AADL modeling languages

To support formal model engineering of (virtually) synchronous CPSs (with the PALS methodology) we: (i) Developed the *Synchronous AADL* modeling language by identifying a *software subset* of AADL (with its Behavior Annex) which, together with a small AADL *property set*, is convenient for defining synchronous designs in AADL, *without* changing the "meaning" of AADL constructs, so that the AADL expert can easily specify synchronous designs in AADL; (ii) Defined the formal semantics of Synchronous AADL in Maude; and (iii) Integrated into OSATE: (a) checking whether an AADL model is also a Synchronous AADL model, (b) synthesizing a Maude model from a Synchronous AADL model, and (c) reachability analysis and temporal logic model checking of the (Maude model generated from the) Synchronous AADL model [8,9].

We successfully applied Synchronous AADL modeling language and analysis tool to model and verify the active standby system inside OSATE. We emphasize that Synchronous AADL—while developed in the context of PALS—should be useful *in general* to model and formally analyze synchronous systems/designs in AADL.

Multirate Synchronous AADL [10] extends Synchronous AADL to the multirate setting, and has been applied to, e.g., the airplane turning system.

HybridSynchAADL [2, 11, 12] defines another AADL subset and property set, e.g., to specify clock skews and continuous behaviors, for specifying synchronous designs of *hybrid* CPSs. In this setting, explicit-state Maude analysis cannot cover all possible behaviors, since we need to cover all possible sensing and actuation times and capture continuous behaviors.

We therefore exploit that Maude has been combined with SMT solving. We represent sensing and actuating times symbolically, and use SMT solving to deal with continuous behaviors and clock skews. The HybridSynchAADL OSATE plugin provides: (i) randomized simulations; (ii) bounded reachability analysis; and (iii) portfolio analysis running these two analysis methods in parallel. HybridSynchAADL has been applied to collections of collaborating autonomous drones.

3 The HybridSynchAADL language

HybridSynchAADL is a subset of AADL extended with the property set `Hybrid_SynchAADL`. HybridSynchAADL can specify synchronous designs of distributed controllers, environments with continuous dynamics, and controller-environment interactions based on imprecise local clocks and sampling and actuation times.

For discrete controllers, we consider a *behavioral subset* of AADL suitable for defining synchronous designs. It includes system, process, and thread components; subprograms; ports and connections; data components and composite data types; and thread behaviors modeled using AADL's Behavior Annex [13]. We do not consider the hardware aspects of AADL, since they are not needed for synchronous designs.

For environments, we consider real-valued variables that change continuously over time. The continuous dynamics is specified using a `Hybrid_SynchAADL` property, as real functions over time or ordinary differential equations. An environment component can have multiple modes to specify different trajectories. An input from a controller may change the mode of the environment or the value of a state variable.

Example. We consider distributed drones that collaborate to achieve common goals. Figure 1 shows the AADL architecture of a rendezvous model for four drones [2, 12]. Each drone is connected to two other drones to exchange their positions. A drone component consists of an environment (with the drone's position and velocity) and its controller.

The corresponding components in HybridSynchAADL are shown below. The implementation of `Drone` declares three `Hybrid_SynchAADL` properties to specify the maximal clock skew, and sampling and actuating time intervals.

```
system FourDronesSystem
end FourDronesSystem;

system implementation FourDronesSystem.impl
 subcomponents
   drone: system Drone[4];
 connections
   C: port drone.oP –> drone.iP {Timing => Delayed;};
```

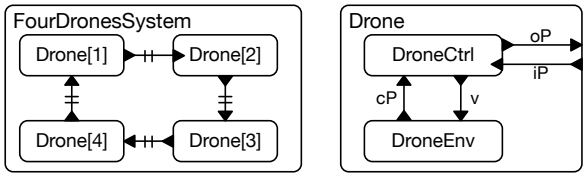

Figure 1: Four drones (left) and a drone component (right).

```
 properties
   Hybrid_SynchAADL::Synchronous => true;
   Period => 100ms;
   Connection_Pattern => ((Cyclic_Next)) applies to C;
end FourDronesSystem.impl;

system Drone
 features
   iP: in data port Vector;      oP: out data port Vector;
end Drone;

system implementation Drone.impl
 subcomponents
   ct: system DroneCtrl.impl;  ev: system DroneEnv.impl;
 connections
   C1: port ct.oP –> oP;       C2: port iP –> ct.iP;
   C3: port ct.v –> ev.v;      C4: port ev.cP –> ct.cP;
 properties
   Hybrid_SynchAADL::Max_Clock_Deviation => 10ms;
   Hybrid_SynchAADL::Sampling_Time => 30ms .. 33ms;
   Hybrid_SynchAADL::Response_Time => 60ms .. 63ms;
end Drone.impl;
```

4 The HybridSynchAADL tool

Property specification language. The HybridSynchAADL tool provides an intuitive language for specifying system requirements of the form

$$type\ \texttt{[}name\texttt{]:}\ \psi\ \texttt{==>}\ \phi\ \texttt{in time}\ \tau$$

A reachability property, with *type* `reachability`, holds if a state satisfying ϕ is reachable from a state satisfying ψ within time τ. An invariant property, with *type* `invariant`, holds if, for every initial state satisfying ψ, all states reachable within time τ satisfy ϕ.

Two requirements for the example in Section 3 are that drones do not collide within $500\,\mathrm{ms}$ (`safety`), and that all drones can gather together within $500\,\mathrm{ms}$ (`rendezvous`):

```
invariant [safety]: ?init ==> not ?collide in time 500ms;
reachability [rendezvous]: ?init ==> ?gather in time 500ms;
```

The user-defined propositions `init`, `collide`, and `gather` are declared as follows, where the keyword `const` is used to introduce a `VectorArray` constant `p`:

```
proposition [initial]:
  forall i in {1..4}. abs(drone[i].ev.p.x – c[i].x) < 0.1 and
                      abs(drone[i].ev.p.y – c[i].y) < 0.1;

const c:VectorArray = [{x: 1.2, y: 1.7}, {x: –1.7, y: –1.2},
                       {x: 1.7, y: 1.2}, {x: –1.2, y: –1.7}];

proposition [gather]:
  forall i in {1..3}. forall k in {1..4}. k <= i or
    (abs(drone[i].ev.p.x – drone[k].ev.p.x) < 2.0 and
     abs(drone[i].ev.p.y – drone[k].ev.p.y) < 2.0);
```

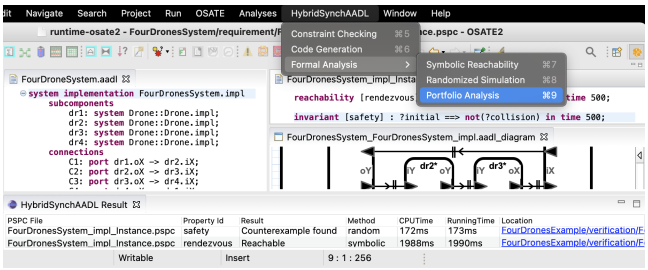

Figure 2: HybridSynchAADL window in OSATE.

```
proposition [collision]:
  forall i in {1..3}. forall k in {1..4}. k <= i or
    (abs(drone[i].ev.p.x – drone[k].ev.p.x) < 0.1 and
     abs(drone[i].ev.p.y – drone[k].ev.p.y) < 0.1);
```

Tool interface and performance. Figure 2 shows the tool interface that is fully integrated into OSATE. The HybridSynchAADL menu contains three items for constraint checking (to check if a given model is a valid HybridSynchAADL model), code generation (to generate the corresponding Maude-with-SMT model), and formal analysis. The Portfolio Analysis item has already been clicked, and the Result view at the bottom displays the analysis results.

The tool provides three analysis methods: symbolic reachability analysis, which can verify that all possible behaviors satisfy a given requirement using Maude combined with SMT solving; randomized simulation, which repeatedly executes the model by randomly choosing concrete data values, sampling and actuating times, etc.; and portfolio analysis, which invokes both methods in parallel using multithreading.

We have compared the performance of HybridSynchAADL's symbolic reachability analysis with four hybrid systems reachability analysis tools, HyComp, SpaceEx, Flow*, and dReach. The experiments showed that (in most cases) HybridSynchAADL outperforms these state-of-the-art tools [2, 12].

5 Concluding remarks

We have defined the Synchronous AADL, Multirate Synchronous AADL, and HybridSynchAADL modeling languages for specifying synchronous (designs of) CPSs in AADL. We have integrated the modeling and Maude-based intuitive push-button formal analysis of such models into the OSATE tool environment for AADL, so that the AADL expert can easily develop and formally verify her synchronous systems/designs within OSATE. In addition, because of the PALS equivalences, verifying such synchronous designs also verifies the much more complex "distributed implementations" of the designs, as long as the infrastructure satisfies the PALS bounds on network delays and clocks skews.

Future work—apart from more applications and including larger subsets of AADL and richer continuous dynamics—includes: (i) combining Synchronous AADL-defined designs with deployment platforms (defined using, e.g., AADL) by developing methods for checking whether the deployment platform satisfies the PALS constraints, and then generating the corresponding deployment model; and (ii) develop symbolic temporal logic analyses also for the hybrid setting.

Acknowledgments. We thank the organizers of ADEPT 2023 for inviting us to present this work. Bae was supported by the NRF grants funded by the Korea government (No. 2021R1A5A1021944 and No. RS-2023-00251577). This work was supported by the NATO Science for Peace and Security Programme project SymSafe (grant number G6133).

References

[1] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL*. USA: Addison-Wesley, 2012.

[2] J. Lee, S. Kim, K. Bae, and P. C. Ölveczky, "HybridSynchAADL: Modeling and formal analysis of virtually synchronous CPSs in AADL," in *CAV*, vol. 12759 of *LNCS*, pp. 491–504, Springer, 2021.

[3] J. Meseguer and P. C. Ölveczky, "Formalization and correctness of the PALS architectural pattern for distributed real-time systems," *Theoretical Computer Science*, vol. 451, pp. 1–37, 2012.

[4] A. Al-Nayeem, M. Sun, X. Qiu, L. Sha, S. P. Miller, and D. D. Cofer, "A formal architecture pattern for real-time distributed systems," in *RTSS*, pp. 161–170, IEEE, 2009.

[5] K. Bae, J. Meseguer, and P. C. Ölveczky, "Formal patterns for multirate distributed real-time systems," *Sci. Comput. Program.*, vol. 91, pp. 3–44, 2014.

[6] K. Bae, J. Krisiloff, J. Meseguer, and P. C. Ölveczky, "Designing and verifying distributed cyber-physical systems using Multirate PALS: An airplane turning control system case study," *Science of Computer Programming*, vol. 103, pp. 13–50, 2015.

[7] K. Bae, P. C. Ölveczky, S. Kong, S. Gao, and E. M. Clarke, "SMT-based analysis of virtually synchronous distributed hybrid systems," in *HSCC*, ACM, 2016.

[8] K. Bae, P. C. Ölveczky, J. Meseguer, and A. Al-Nayeem, "The SynchAADL2Maude tool," in *FASE*, vol. 7212 of *LNCS*, pp. 59–62, Springer, 2012.

[9] K. Bae, P. C. Ölveczky, A. Al-Nayeem, and J. Meseguer, "Synchronous AADL and its formal analysis in Real-Time Maude," in *ICFEM*, vol. 6991 of *LNCS*, pp. 651–667, Springer, 2011.

[10] K. Bae, P. C. Ölveczky, and J. Meseguer, "Definition, semantics, and analysis of Multirate Synchronous AADL," in *FM*, vol. 8442 of *LNCS*, pp. 94–109, Springer, 2014.

[11] J. Lee, K. Bae, and P. C. Ölveczky, "An extension of HybridSynchAADL and its application to collaborating autonomous UAVs," in *ISoLA*, vol. 13703 of *LNCS*, pp. 47–64, Springer, 2022.

[12] J. Lee, K. Bae, P. C. Ölveczky, S. Kim, and M. Kang, "Modeling and formal analysis of virtually synchronous cyber-physical systems in AADL," *Int. J. Softw. Tools Technol. Transf.*, vol. 24, no. 6, pp. 911–948, 2022.

[13] R. França, J.-P. Bodeveix, M. Filali, J.-F. Rolland, D. Chemouil, and D. Thomas, "The AADL Behaviour Annex - experiments and roadmap," in *ICECCS*, pp. 377–382, IEEE, 2007.

BLESS Behavior Correctness Proof as Convincing Verification Artifact

Brian R. Larson

Multitude Corporation, St. Paul, MN 55126, USA; email: brl@multitude.net

Ehsan Ahmad

College of Computing and Informatics, Saudi Electronic University, Riyadh, Saudi Arabia; email: e.ahmad@seu.edu.sa

Abstract

Safety-critical cyber-physical systems require evidence they are indeed safe. In practice, such evidence is results of system tests. Unfortunately, tests can only demonstrate the presence of software errors, not their absence, and can practically cover a tiny fraction of system state space. Valiant efforts to formally verify program correctness have either been excruciatingly difficult (theorem proving) or incomplete (static analysis, model checking, SAT or SMT solving).

The BLESS Methodology was created specifically for engineers in industry to formally verify software controlling machines. The BLESS Methodology transforms programs that control machines, annotated with assertions to form proof outlines, into deductive proofs that every possible program execution will conform to its specification. To the extent that cyber-physical system specifications have been validated to express system safety and performance, a deductive proof can be a convincing argument to a person of program correctness.

This paper uses a simple safety-critical system to argue that behavior correctness proof under the BLESS Methodology is a convincing verification artifact in addition to customary testing.

Keywords: BLESS, AADL, correctness proof, formal method, declarative specification

1 Introduction

BLESS is an acronym for Behavior Language for Embedded Systems with Software, but the BLESS Methodology is a way to specify, design, and verify software controlling machines.

The BLESS Methodology provides a formal method for verification of software that controls machines. Such software-controlled machines are often called embedded or cyber-physical systems. The BLESS Methodology creates programs together with deductive proofs that every possible program execution will conform to its specification. A deductive proof is a sequence of theorems, each of which are axioms or derived from prior theorems by sound inference rules. A deductive proof will be a convincing argument to a person when he or she can examine and understand each theorem to determine whether it is individually convincing.

The BLESS Methodology applies to an architectural *model* of a cyber-physical system using the Architecture Analysis and Design Language (AADL) [1]. A deployed system can be correct-by-construction to the extent that the implementation can be auto-generated from the architecture.

1.1 What is Proof?

Colloquially, "proof" usually means "evidence", with perhaps some reasoning about it. Legally, proof is confirmation of fact by evidence. For civil trials, "preponderance of the evidence" suffices for proof. For criminal trials, proof requires "beyond a reasonable doubt." Proof is (or should be) an *argument* to *convince* people of its conclusion.

A *deductive proof* is a sequence of theorems, each of which is given, or an axiom, or derived from theorems in the sequence by some reason. The last theorem is the conclusion: what is being proved. When the axioms have been proved (by some other means) to be tautologies, the reasons are inference rules proved to be sound (derive true facts from true facts), and the givens appropriately describe the subject of the argument, people can decide if they believe the conclusion and with what confidence.

Metamath [2] strives for proofs of (meta-)theorems with maximal confidence with a simple, robust proof checker. Metamath's proof checker has been implemented in more than a dozen languages, by as many people, to provide maximal confidence in its theorems. Although Metamath allows anyone to define their own axioms, and will check the theorems derived from them, the most popular and largest theorem data base, `set.mm` uses the axioms of logic together with axioms of ZFC set theory. At the time of this writing, `set.mm` has 43509 axioms, definitions, and theorems. Every theorem can be traced back to the axioms and definitions. Best of all, the native text representation of Metamath theorems can be exported to LaTeX and typeset into mathematical notation, so people can examine the proofs and decide if they're convinced.

Too many formal methods for verification of software provide evidence without argument. Others produce proofs that

uses a much different language that the programs being verified, and often are unreadable. Unreadable proofs cannot be convincing.

1.2 BLESS Proofs

Correctness proofs under BLESS Methodology are written in the same language as BLESS programs. A BLESS program expresses detailed behavior of an architectural component using state-transition machines. Every effort has been made to make generated proofs understandable and persuasive. All of the axioms, and many of the inference rules used in BLESS proofs have Metamath soundness proofs. The BLESS proof assistant is written in Java, thus have no proof that the proved-sound axioms and inference rules were implemented correctly. However, the proof themselves can be examined to determine if they are convincing arguments for program correctness–regardless of how they were produced.

1.3 Outline

The rest of the paper is organized as follows. Section 2 presents an overview of the AADL and BLESS Methodology. In Section 3, the movement authority scenario of Chinese Train Control System Level 3 is presented as an exemplar to demonstrate precise behavior specification and its correctness proof under the BLESS Methodology. Section 4 presents structural modeling of the components involved in the movement authority scenario. Behavior specification using BLESS properties is presented in Section 5. Behavior implementation as a state-transition machine in a BLESS annex subclause appears in Section 6. Section 7 discuses verification condition generation while Section 8 presents the correctness proof for one of the thread component. Potential doubts are discussed in Section 9, and Section 10 concludes the study.

2 Background

2.1 AADL

An embedded system architecture in AADL consists of connected components specified by their *type* and *implementation* classifiers for both the application software and the execution platform. Components are connected through externally visible interfaces called *ports* specified in the *features* section of the type classifier. Port communication is typed and directional and is used to transmit and receive data, control and data and control through *data* ports, *event* ports and *event data* ports, respectively. Internal structure of a particular component is realized by specifying its subcomponents and the connections between them in the implementation classifier.

Logical architectures (software) are modeled with *process*, *data*, *subprogram*, *thread*, and *thread group* components. Process components model the protected memory space shared among thread subcomponents which model active software which can issue and respond dynamically to events. Data components model types, persistent values directly accessible by multiple threads. Subprogram components model procedures and functions which must be invoked by threads or other subcomponents.

Physical architectures are modeled with *processor*, *memory*, *bus*, and *device* components. Hardware that executes software is modeled with processor components. Memory components model data storage. Device components model other physical entities like actuators and sensors or custom logic. Bus components model the physical connections (i.e. wires) between execution platform components.

AADL allows binding of logical components and connections to physical components. This allows separation of functional architecture (software) from physical architecture (what must be manufactured) to allow implementation by teams with specialized skills, yet assure that the software will execute on the hardware.

AADL also provides *system* components to model composition of logical and physical components, and *abstract* components to model interfaces without further elaboration. In a previous study, we used abstract components with Hybrid annex [3] subclauses to model environments controlled by the system being designed. In this way, AADL models electronics being designed, and the system/environment being controlled.

The core AADL standard only provides support for structural modeling of embedded computing systems and nothing related to detailed behavior of the software and physical processes which are controlled by the software can be modeled. BLESS was introduced as an annex sublanguage of AADL to model precise behavior for formal verification [4].

Open Source AADL Tool Environment (OSATE) [5] is an open source platform and tool set built on Eclipse to implement AADL for the modeling and analysis of real-time embedded systems.

2.2 BLESS Methodology

BLESS Methodology creates programs together with their specifications and correctness proofs. BLESS Methodology cannot be applied to legacy programs–even though they may be correct. BLESS programs are specially crafted to include “proof outlines" so that correctness proofs may be constructed. Proof outlines are predicates (assertions) attached to states, and interspersed throughout actions. The predicates are true when the program is at a particular state, or a point during execution. Correctness proofs are derived from BLESS programs having proof outlines using an interactive proof assistant.

BLESS methodology uses five domain specific languages (DSLs) edited with a plugin extension to OSATE. Each DSL is used in AADL subclauses, libraries, or properties. An *Assertion* language to make assertions (statements of fact), a *Typedef* language to define types and units, a *Unit* language to specify measure units, a *State Machine* language to express thread behavior, and an *Action* language to express imperative control-flow computation.

The BLESS proof assistant transforms program proof outlines into deductive proofs, given human selection of proof tactics. Complete description of the BLESS Methodology is beyond the scope of this paper. For more details, refer to the BLESS reference and proof assistant guide [6]. Below, we briefly introduce the components of the BLESS Methodology related to this study.

2.2.1 Assertion Language

A BLESS *assertion* is a statement of fact believed to be true, enclosed in angle brackets `<<>>`. AADL component behavior can be declaratively-specified using BLESS assertions. BLESS assertions are interspersed throughout actions in programs to be proof outlines. BLESS assertions are first-order predicates extended with a simple temporal operator to formally express the behavior constraints.

For concision and clarity, a *named* BLESS assertion defines a label for a predicate which may be used as a placeholder in other assertions as if the label was replaced by the predicate. Named assertions may have parameters. Formal parameters are replaced by actual parameters before replacement of the named assertion invocation. A comprehensive use of the BLESS assertion language for behavior specification is detailed in [7].

An `Assertion` annex library collects globally-visible named assertions and defines "ghost" variables denoting values occurring in assertions, but not actions in programs.

2.2.2 Behavior Language

AADL component behavior may be defined in the architectural model by an annex subclause attached to the component. BLESS behavior language is used to express detailed behavior of threads and devices as state-transition machines. A set of states is declared, followed by transitions from a source state to a destination state when a transition condition is true, with possibly an action performed after leaving the source state and prior to entering the destination state. There are four kinds of states. There must be exactly one "initial" state which cannot be the destination of any transition. There may be one or more "final" states which cannot be the source of any transition. Entering a "complete" state suspends execution until next dispatch. An "execution" state is transitory between dispatch and suspension, therefore must always have at least one, enabled outgoing transition.

`BLESS` annex subclauses attached to thread components express software behavior while `BLESS` annex subclauses attached to device components express hardware behavior and its device driver.

`Action` annex subclauses define behavior of passive subprogram components which calculate the values of `out` parameters solely from values of `in` parameters. Subprograms may not have side-effects such as persistent state, waiting on locks, or other component interaction.

2.2.3 BLESS IDE Plugin to Eclipse/OSATE

After installing OSATE, the BLESS IDE (integrated development environment) plugin can be easily installed from the "Help" menu, selecting "Install Additional OSATE Components". The BLESS plugin checks syntax of all BLESS annex sublanguages, performs unified unit-type checking, validates many other rules and constraints, and generates code in other languages.

2.2.4 BLESS Proof Assistant

The BLESS Proof Assistant generates verification conditions (VCs), reducing them to axioms according to human selected tactics. When all VCs have been reduced to axioms, theorem numbers are assigned, depth-first, and the deductive proof of correctness emitted.

Tactics applied manually are recorded in a *proof script*, which may be subsequently applied. As VCs are incrementally solved, proof scripts can be extended, so effort can be focused on the next unsolved VC. Should program changes made to solve the current VC affect previously solved VCs, re-running the proof script stops when three consecutive tactics have no effect. This invariably indicates where program text, tactics, or both must be updated for VCs affected by the change.

3 Chinese Train Control System Level 3

Chinese Train Control System Level 3 (CTCS-3) has been cited in various studies as an example of how formal approaches might be employed [8], [9], [10]. The CTCS-3 defines two subsystems: an *on-board* subsystem and a *trackside* subsystem. The on-board subsystem monitors the movement of the train based on communications with the trackside subsystem and the train to which it belongs. The behavior of CTCS-3 is defined in terms of 14 basic scenarios, all of which cooperate with each other to constitute safe functionality of train control system. The Movement Authority scenario is considered in this paper to demonstrate that BLESS behavior correctness proofs are convincing verification artifacts.

In a previous paper [10], we used the Movement Authority scenario to explain the use of the BLESS and Hybrid annex sublanguages to model the discrete behavior of controller and continuous behavior of the train, respectively. The system-level behavior was verified using the Hybrid Hoare Logic theorem prover. On other hand, this study considers a modified set of requirements with different modeling components. As a result, the architectural model (shown in Figure 3) is more complex, and the detailed behavior is specified using latest BLESS behavior and assertion languages. Additionally, this paper aims to investigate BLESS behavior correctness proof as convincing verification artifact, so behavior specification, verification condition generation, and transformation of verification conditions into theorems are all extensively covered.

3.1 The Movement Authority Scenario

According to [11], in CTCS-3, the train applies for a *movement authorization* (MA) from the Radio Block Center (RBC). If MA is granted by RBC, the train gets permission to move within the portion of track in the domain of the MA under the control of a train operator. To assure safety, under certain conditions, control of the train may be overridden and the brakes are automatically applied to slow or stop the train. First, the normal, service brake is applied. If the service brake fails, or does not slow the train sufficiently, emergency brakes are automatically applied. Thus hazards of both operator error and brake malfunction are mitigated.

Each MA is composed of a sequence of segments where each segment has two speed limits v_s and v_e, the segment end point e and the *mode* to represent the operating mode of the train.

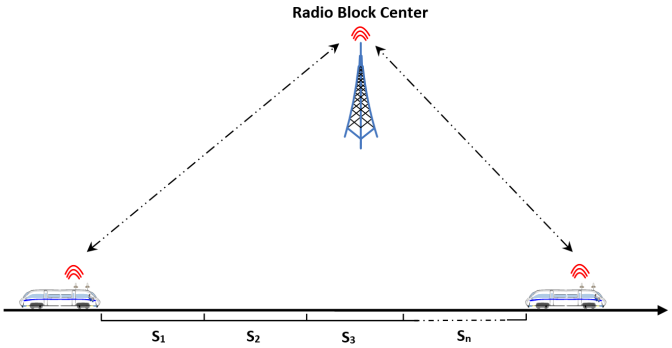

Figure 1: Trains coordination with dynamic movement authorities in CTCS-3

Speed limits v_s and v_e represent the limits for train to apply normal service brake and emergency brake, respectively.

Figure 1 depicts an MA with n segments; s_1 to s_n. The train applies for a MA for the next section of track when it enters the final segment of its current MA. If MA for the next section of track is not received, the train fully stop at the end of the final segment of its current MA.

Figure 2 depicts the static and dynamic speed profiles for each segment for a given MA segment. Let the current train velocity be v and the current segment be *seg*, with service brake limit v_s and emergency brake limit v_e. The service brake is applied if the velocity is at least the service brake limit, $v \geq seg.v_s$, and the emergency brake is applied if the velocity is at least the emergency brake limit, $v \geq seg.v_e$.

If the speed limits for the next segment are lower than the current segment, service or emergency brakes are applied before the end of the current segment. Let the current position be p, the end of the current segment be $seg.e$, and the service and emergency speed limits of the next segment be $next(seg).v_s$ and $next(seg).v_e$, respectively.

If the speed limits for the next segment are less than the current segment, automatic braking may need to be applied even though the train velocity is below $seg.v_e$ and $seg.v_s$. The dynamic speed profiles (shown as dotted lines in Figure 2) are calculated using train's deceleration for either normal braking b, or emergency braking e, its position p, the position of the end of the segment $seg.e$, and the speed limits of the next segment, $next(seg).v_s$ and $next(seg).v_e$.

Apply service brake:

$$SB = v \geq seg.v_s \vee v^2 \geq next(seg).v_s^2 + 2 \times b \times (seg.e - p)$$

Apply emergency brake:

$$EB = v \geq seg.v_e \vee v^2 \geq next(seg).v_e^2 + 2 \times e \times (seg.e - p)$$

Except for the last segment of a MA, the next segment, $next(seg)$ has a segment number one higher than the current segment seg. Upon entering the last segment of a MA, an new MA is requested. Until the new MA is received, the speed limits are zero, so the train won't leave the current MA. If a new MA is received the next segment is changed to the first segment of the new MA.

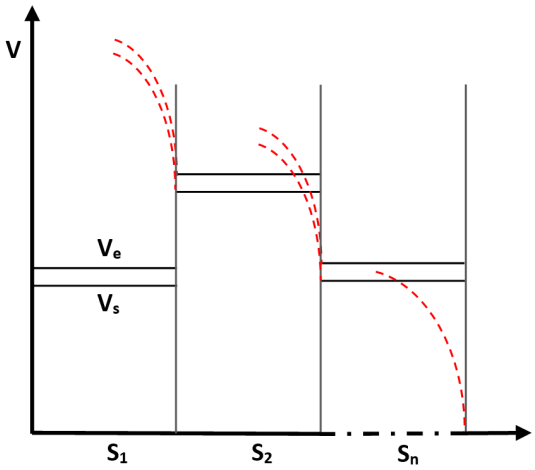

Figure 2: Static and dynamic speed profiles

3.2 Involved Components

Below we explain the components involved in the MA scenario.

3.2.1 Radio Block Center

The Radio Block Center (RBC), a computer-based system, communicates with the train using information gathered from trackside and on-board subsystems. The main objective of RBC is to provide/extend MAs (in CTCS-3) to allow the safe movement of trains. As depicted in Figure 1, dynamic MA is assigned depending on the current track situation and movement of other trains within the region of responsibility of a particular RBC.

3.2.2 Train

The Train is a typical hybrid system in CTCS-3 with continuous evolution of its position, speed and discrete changes in its acceleration computed to follow the speed constrains in each segments of a particular MA. The speed and distance of the train is sampled periodically by the controller and new acceleration value is computed on the basis of the current position and speed of the train and the information received from RBC.

3.2.3 Automatic Braking

In the MA scenario, a human train operator normally controls the train's acceleration, and thus velocity, subject to velocity limitations. The automatic braking responsible for controlling the speed of the train such that its movement is restricted to the MA it owns. Depending on the current speed and the position of the train (sampled after every 200 milliseconds) the controller adjusts the acceleration so that the train never runs beyond the static and dynamic speed profiles of a particular segment of the current MA and stops safely before the *End of Authority* (EoA) if the MA is not extended in time.

4 Structural Modeling using AADL

This section describes modeling of the movement authority scenario using AADL. As shown in Figure 3, the MA scenario is modeled with an AADL system implementation `MovementAuthority.i`. It is composed of two main subcomponents, the `RadioBlockCenter ( rbc )` and the `Train ( train )`.

```
system MovementAuthority
end MovementAuthority;

system implementation MovementAuthority.i
 subcomponents
  rbc : system RadioBlockCenter;
  train : system Train::Train.i;
 connections
  ma_request : port train.r -> rbc.r;
  ma : port rbc.ma -> train.ma;
end MovementAuthority.i;
```

4.1 RadioBlockCenter

The RBC is modeled by the `RadioBlockCenter` system. Its `r` in event port receives MA requests, and `ma` out event data port transmits a granted MA. The **`BLESS::Value`** property of port `ma` specifies that it has BLESS type `movementAuthorization` with value `RMA`, which is a "ghost" variable only appearing in assertions representing the requested movement authority.

```
system RadioBlockCenter
 features
  r : in event port;  --MA request
  ma : out event data port --MA grant
    CTCS_Types::movementAuthorization
  {BLESS::Value =>
 "<<returns movementAuthorization := RMA>>";};
end RadioBlockCenter;
```

4.2 Train

The `Train` system type features mirror the `RadioBlockCenter` features having opposite directions. The features are connected in the `MovementAuthority.i` system implementation

```
system Train
 features
  r : out event port; --train requests MA
  ma : in event data port  --received MA
    CTCS_Types::movementAuthorization
  {BLESS::Value =>
 "<<returns movementAuthorization := RMA>>";};
end Train;
```

The `Train.i` system implementation has six subcomponents:

motor accelerate and decelerate the train

sensor measure train position and velocity

controller auto-braking software

driver human train operator

ebrake emergency brake

sbrake service (normal) brake

```
system implementation Train.i
 subcomponents
  motor : device Motor;
  sensor : device Sensor;
  controller : process ControllerProcess.i;
  driver: abstract Operator;
  ebrake: device EmergencyBrake;
  sbrake: device ServiceBrake;
 connections
  ma_request : port controller.r -> r;
  auth : port ma -> controller.m_a;
  pos: port sensor.p -> controller.p;
  vel : port sensor.v -> controller.v;
  mxl : port controller.ca -> motor.ca;
  dxl : port driver.xl -> controller.xl;
  cpsb: port controller.sb -> sbrake.brake;
  cpeb: port controller.eb -> ebrake.brake;
end Train.i;
```

4.2.1 *Motor*

The `Motor` device component moves the train with acceleration chosen by operator, unless overridden by automatic braking.

```
device Motor
 features
  xl : --current acceleration
    in data port CTCS_Types::Acceleration
  {BLESS::Value =>
 "<<returns quantity mpss := OPERATOR_XL>>";};
end Motor;
```

4.2.2 *Sensor*

The `Sensor` device measures train position `p` in meters and velocity `v` in meters per second. The position value is relative to the endpoints of segments to determine whether dynamic braking must be done to transition to the next segment.

```
device Sensor
 features
  p : out event data port CTCS_Types::Position
   {BLESS::Value=>
    "<<returns quantity m := POSITION>>";};
  v : out event data port CTCS_Types::Velocity
   {BLESS::Value=>
    "<<returns quantity mps := VELOCITY>>";};
 properties
  Dispatch_Protocol => Periodic;
  Period => 200 ms;
end Sensor;
```

4.2.3 *Operator*

The human `Operator` is modeled as an abstract component. The `xl` out data port transmits the intended train acceleration value, without automatic braking.

```
abstract Operator
 features
  xl : out data port CTCS_Types::Acceleration
   {BLESS::Value =>
    "<<returns quantity mpss := OPERATOR_XL>>";};
end Operator;
```

4.2.4 *Controller Process*

In AADL, a *process* is a protected address space. Thread components must be contained within a process component. Because `ControllerProcess.i` contains a single thread which has the same features, so further explanation is unnecessary.

4.2.5 *AutoBrake Thread*

The `AutoBrake` thread automatically applies either the service brake or emergency brake. Its behavior is the subject of the next section.

5 Behavior Specification using BLESS Properties

Behavior of the `AutoBrake` thread is *specified* using **`BLESS::Assertion`** and **`BLESS::Value`** properties of its ports.

```
thread AutoBrake
 features
  sb: out event data port BLESS_Types::Boolean -- apply service brake
  {BLESS::Assertion => "<<SB() and not EB()>>";};
  eb: out event data port BLESS_Types::Boolean -- apply emergency brake
  {BLESS::Assertion => "<<EB()>>";};
  r: out event port; -- request new movement authorization (MA)
  m_a: in event data port CTCS_Types::movementAuthorization -- received MA
  {BLESS::Value => "<<returns movementAuthorization := RMA>>";};
  p: in event data port CTCS_Types::Position -- current measured position
  {BLESS::Value => "<<returns quantity m := POSITION>>";};
  v: in event data port CTCS_Types::Velocity -- current measured velocity
  {BLESS::Value => "<<returns quantity mps := VELOCITY>>";};
  xl : in data port CTCS_Types::Acceleration  --operator chosen acceleration
  {BLESS::Value => "<<returns quantity mpss := OPERATOR_XL>>";};
  ca : out  data port CTCS_Types::Acceleration  --acceleration to motor
  {BLESS::Value => "<<returns quantity mpss := TRAIN_XL()>>";};
 properties
  Dispatch_Protocol => Sporadic;
end AutoBrake;
```

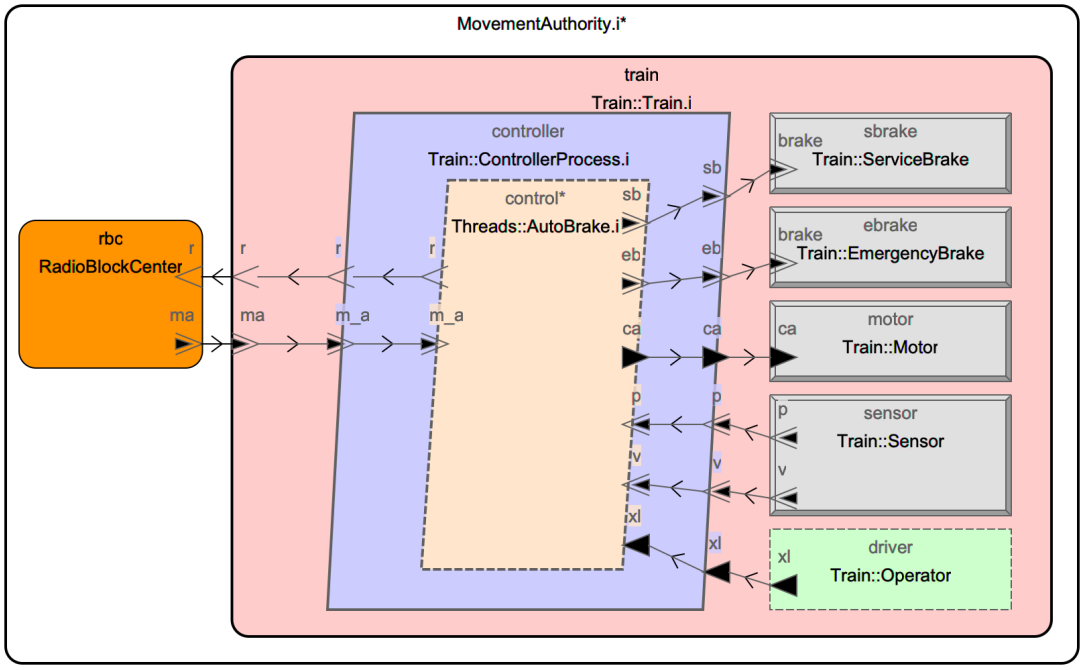

Figure 3: AADL graphical model for Movement Authority scenario

`BLESS::Assertion` properties define predicates within `<<>>` which must match the values of boolean-typed ports. Therefore port `sb` outputs true when `<<SB() and not EB()>>` and false otherwise. Because `sb` is an "out" port, thread `AutoBrake` *guarantees* the `BLESS::Assertion` property. Conversely, a `BLESS::Assertion` property of an "in" port is assumed. Connections from out ports to in ports have assume-guarantee verification conditions, which are not considered in this paper. The parentheses of `SB()` and `EB()` indicate they reference labelled assertions, without parameters, defined elsewhere.

`BLESS::Value` properties define non-boolean expressions which must match the values of (non-boolean) data ports. Port `m_a` expects a value having type `movementAuthorization` with value `RMA`, which is a "ghost" variable defined in `Assertion` annex libraries.

```
annex Assertion
{**
 ghost variables
  def POSITION ~ quantity m       -- measured position
  def VELOCITY ~ quantity mps     -- measured velocity
  def RMA ~ movementAuthorization     -- requested movement authorization
  def CMA ~ movementAuthorization     -- currently active movement authorization
  def NEXT_MA ~ movementAuthorization -- next MA if granted
  def OPERATOR_XL ~ quantity mpss --operator commanded acceleration
  . . .
```

`RMA` represents the value of the movement authorization provided by the RBC in 4.1.

Thread specifications may include a thread invariant in addition to `BLESS::Assertion` and `BLESS::Value` properties of its ports, but was not needed for this example. Thus users of a thread need only concern themselves with the thread type, provided the thread implementation has proof that it conforms to its specification by its type.

6 Behavior Implementation using BLESS Annex Subclause

Behavior of the `AutoBrake` thread is *implemented* using BLESS annex sublanguage of AADL within thread implementation `AutoBrake.i`. In AADL, names of component types are identifiers, and names of implementations are the identifier of their type separated from the identifier of the implementation by a period. A component type may have several implementations having different implementation identifiers.

`BLESS` annex subclauses occur within the text of a component beginning with `annex BLESS {**` and ending with `**};`. The optional `assert` section defines assertions with labels. Here, assertions `SB()` and `EB()` are defined. The mandatory `invariant` defines the assertion which must be true whenever the thread suspends or dispatches. For this case, no invariant is necessary so it is always `true`.

```
30 thread implementation AutoBrake.i
31  annex BLESS
32 {**
33  assert
34   <<SB: :  --apply service brake
35    v >= iSeg.v_n or v*v >= nSeg.v_n*nSeg.v_n + 2*b*(iSeg.e-p)>>
36   <<EB: :  --apply emergency brake
37    v >= iSeg.v_e or v*v >= nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p)>>
38
39  invariant << true >>  --no thread invariant is necessary
```

6.1 Variables

The `variables` section, in the `BLESS` annex, declares variables having values which persist between thread suspension and dispatch. Temporary, scope-limited variables may be declared if necessary. BLESS Methodology has a type-unit system which unifies type and unit checking. Consequently, every number is a `quantity` having a unit which may be scalar. Simple, scalar integers are `whole` quantities. Otherwise all quantities are real numbers to be approximated by floating point.

Units are defined in `Unit` annex libraries. Unit `mpss` is defined as m/s^2 for acceleration in `SI.aadl` which defines base and derived units for the SI international system of units including `m` for meter, `s` for second, and `mps` for m/s.

Variables `b` and `e` are constants defined to provide short names for AADL property constants. `AXIOM_B` and `AXIOM_E` define "givens" that the short identifiers are equivalent to their AADL property constants. Because assertions with labels beginning with `AXIOM` are accepted by the proof assistant as given, special scrutiny must be appleid to assure that they are actually true, or proofs using them may be fallacious.

```
variables
 i ~ quantity whole := 0 whole -- segment indexing
 b ~ quantity mpss constant :=  CTCS_Property::SB_Rate mpss --service braking
  <<AXIOM_B: : b = CTCS_Property::SB_Rate mpss>>
 e ~ quantity mpss constant := CTCS_Property::EB_Rate mpss --emergency braking
  <<AXIOM_E: : e = CTCS_Property::EB_Rate mpss>>
 ma ~ movementAuthorization -- current movement authorization
 next_ma ~ movementAuthorization  --next movement authorization
 iSeg ~ segment --current segement
 nSeg ~ segment --next segment
```

BLESS types are declared in `Typedef` annex libraries.

```
annex Typedef
{**
 type segment is record (
  v_n :  quantity mps -- velocity for (normal) service brake limit (SBL)
  v_e :  quantity mps -- velocity for emergency brake limit (EBL)
  e :  quantity m )  -- position of end of this segment

 type movementAuthorization is record (
  ba : quantity m  --beginning position of this MA
  num_segments : quantity whole  --number of segments in this MA
  seg : array[#CTCS_Property::MaxSegments] of segment
  ea : quantity m )  --ending position of authority
**};
```

6.2 States

The **states** section of the **BLESS** annex declares states used by the state-transition machine. Each state may have an assertion which must be true when the state-transition machine has that state. There must be exactly one **initial** from which the state-transition machine begins. There may be one or more **final** states from which no transitions are allowed. When the state-transition machine enters a **complete** state, it suspends until next dispatch. Execution states have no other label and must be transitory. When dispatched from a complete state, a finite number of execution states may be traversed before entering a complete state to suspend execution until next dispatch.

We find descriptive, camel-case, state identifiers to be helpful in writing and understanding state machine–even when they get long.

```
states
 Start: initial state --train stopped
 WaitFirstMA: complete state  --Wait for first MA
 CheckFirstMA: state      -- Check first MA
 MoveForward: complete state  --Move Forward
  << i<CMA.num_segments and iSeg=CMA.seg[i] and nSeg=CMA.seg[i + 1] and ma=CMA>>
 CheckMoveForward: state      --Check Move Forward
  << i<CMA.num_segments and iSeg=CMA.seg[i] and nSeg=CMA.seg[i + 1] and ma=CMA>>
 CheckForLastSegment: state  --check for last segment
  << iSeg = CMA.seg[i] and ma=CMA >>
 MoveForwardLastSegment: complete state  --Move Forward Last Segment, no new MA
  << i=CMA.num_segments and iSeg=CMA.seg[i] and nSeg=NULL_SEGMENT() and ma=CMA>>
 CheckMoveForwardLastSegment: state --check move forward last segment, no new MA
  << i=CMA.num_segments and iSeg=CMA.seg[i] and nSeg=NULL_SEGMENT() and ma=CMA>>
 GotNewMA: complete state  --on last segment, got new MA
  << i=CMA.num_segments and iSeg=CMA.seg[i] and nSeg=NEXT_MA.seg[1] and ma=CMA
    and next_ma=NEXT_MA >>
 CheckMATransition: state  --change to new MA?
  << i=CMA.num_segments and iSeg=CMA.seg[i] and nSeg=NEXT_MA.seg[1] and ma=CMA
    and next_ma=NEXT_MA >>
 FAIL: final state  --failure occurred
```

6.3 Transitions

Transitions leaving initial or execution states have transition conditions which are boolean expressions which may use values of variables or ports, or nothing at all which is taken to be "true". At least one transition condition leaving an execution state must be true. If more than one condition of transitions leaving an execution state are true, any of them may be taken, non-deterministically.

Transitions leaving complete states have dispatch conditions, which begin **on dispatch** and are restricted to a disjunction of conjunctions of dispatch triggers. Dispatch triggers can be arrival of events or event data, or a timeout. In this example dispatch is limited to arrival of position data at event data port `p`.

The transitions, `Go`, `FirstMA`, `GotFirstMA`, and `NotYet` request and receive the first MA necessary for the train to move. The `CheckSpeed` transition does most of the work, determining whether the service brake or the emergency brake should be applied, or the train operator's choice of acceleration should be used. Transitions `SameSegment`, `NextSegment`, `NotLastSegment`, `IsLastSegment`, and `PastLastSegment` check whether a new segment of the current MA is entered, and if so, whether it is the last segment of the current MA. `IsLastSegment` requests the next MA, and set the braking velocities temporarily to 0. Transition `LastSegment` performs the same calculation as `CheckSpeed`. Transitions `NoMAYet` and `GetNewMA` handle receipt (or not) of a new MA. Transitions `LastBitOfMa`, `NotEndOfMA`, and `StartNextMa` handle transition to the new MA.

```
transitions
 Go: --request movement authorization
 Start -[]-> WaitFirstMA { r! }

 FirstMA: --dispatch before first MA
 WaitFirstMA -[on dispatch p]-> CheckFirstMA

 NotYet: --did not get requested movement authorization
 CheckFirstMA -[not m_a'fresh]-> WaitFirstMA

 GotFirstMA:  --received movement authorization
 CheckFirstMA -[m_a'fresh]-> MoveForward
  {  << AXIOM_CMA_IS_RMA() >>
  m_a?(ma)  --save received movement authorization
  ;  << ma=CMA >>
  i := 1  --first segment of new movement authorization
  ; << i=1 and ma=CMA >>
  iSeg := ma.seg[1] --set current segment to first segment
  ; << i=1 and ma=CMA and iSeg=CMA.seg[i]
      and AXIOM_NUM_SEG(ma:ma) >>
  nSeg := ma.seg[2]  --set next segment to second segment
  << i=1 and ma=CMA and iSeg=CMA.seg[i]
     and nSeg=CMA.seg[i+1] and AXIOM_NUM_SEG(ma:CMA) >>
  }

 CheckSpeed:
 MoveForward -[on dispatch p]-> CheckMoveForward
 { << i<CMA.num_segments and iSeg=CMA.seg[i]
   and nSeg=CMA.seg[i + 1] and ma=CMA and AXIOM_B()
   and AXIOM_E() and AXIOM_V(seg:iSeg)
   and AXIOM_V(seg:nSeg) >>
 if  --exceed emergency brake velocity?
   (v >= iSeg.v_e  )~>
   { eb!(true) & sb!(false) & ca!(0 mpss) }
 []  --emergency brake for next segment?
   (v*v >= nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p) )~>
   { eb!(true) & sb!(false) & ca!(0 mpss) }
 []  --exceed service brake velocity?
   (v >= iSeg.v_n and v < iSeg.v_e and
   v*v < nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p) )~>
   { sb!(true) & eb!(false) & ca!(0 mpss) }
 []  --service brake for next segment?
   (v*v < nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p)
     and v < iSeg.v_e
     and v*v >= nSeg.v_n*nSeg.v_n + 2*b*(iSeg.e-p) )~>
   { sb!(true) & eb!(false) & ca!(0 mpss) }
 []  --no auto brake needed
   ( v < iSeg.v_n
     and v*v < nSeg.v_n*nSeg.v_n + 2*b*(iSeg.e-p)
     and v*v < nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p) )~>
   { sb!(false) & eb!(false) & ca!(xl) }
 fi
 }

 SameSegment: --not at end of this segment
```

```
129   CheckMoveForward -[p < iSeg.e]-> MoveForward
130
131   NextSegment:  --go to next segment
132   CheckMoveForward -[p >= iSeg.e]-> CheckForLastSegment
133   {
134   iSeg := nSeg
135   ; << iSeg = CMA.seg[i+1] and ma=CMA >>
136   i := i+1
137   }
138
139   NotLastSegment:  --not the last segment
140   CheckForLastSegment -[i < ma.num_segments]-> MoveForward
141   { nSeg := ma.seg[i+1] }
142
143   IsLastSegment:  --is the last segment
144   CheckForLastSegment -[i = ma.num_segments]-> MoveForwardLastSegment
145   {
146   r!  --request new movement authorization
147   ; << i=CMA.num_segments and iSeg=CMA.seg[i] and ma=CMA >>
148   --set nSeg to stop
149   nSeg := [ segment :
150       v_n => 0 mps
151       v_e => 0 mps
152       e => ma.ea ]
153   }
154
155   PastLastSegment:  --only for Serban's theorem, will never be executed
156   CheckForLastSegment -[i > ma.num_segments]-> FAIL
157
158   LastSegment:
159   MoveForwardLastSegment -[on dispatch p]-> CheckMoveForwardLastSegment
160   { << i=CMA.num_segments and iSeg=CMA.seg[i] and nSeg=NULL_SEGMENT()
161     and ma=CMA and AXIOM_B() and AXIOM_E()
162     and AXIOM_V(seg:iSeg) and AXIOM_V(seg:nSeg) >>
163   if  --exceed emergency brake velocity?
164     (v >= iSeg.v_e  )~>
165     { eb!(true) & sb!(false) & ca!(0 mpss) }
166   []  --emergency brake for next segment?
167     (v*v >= nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p) )~>
168     { eb!(true) & sb!(false) & ca!(0 mpss) }
169   []  --exceed service brake velocity?
170     (v >= iSeg.v_n and v < iSeg.v_e and
171     v*v < nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p) )~>
172     { sb!(true) & eb!(false) & ca!(0 mpss) }
173   []  --service brake for next segment?
174     (v*v < nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p)
175       and v < iSeg.v_e
176       and v*v >= nSeg.v_n*nSeg.v_n + 2*b*(iSeg.e-p) )~>
177     { sb!(true) & eb!(false) & ca!(0 mpss) }
178   []  --no auto brake needed
179     ( v < iSeg.v_n
180       and v*v < nSeg.v_n*nSeg.v_n + 2*b*(iSeg.e-p)
181       and v*v < nSeg.v_e*nSeg.v_e + 2*e*(iSeg.e-p) )~>
182     { sb!(false) & eb!(false) & ca!(x1) }
183   fi
184   }
185
186   NoMAYet:
187   CheckMoveForwardLastSegment -[not m_a'fresh]-> MoveForwardLastSegment
188
189   GetNewMA:
190   CheckMoveForwardLastSegment -[m_a'fresh]-> GotNewMA
191   { << i=CMA.num_segments and iSeg = CMA.seg[i] and ma = CMA
192      and AXIOM_NEXT_MA_IS_RMA() >>
193   m_a?(next_ma)
194   ;  << i=CMA.num_segments and next_ma = NEXT_MA
195       and iSeg = CMA.seg[CMA.num_segments] and ma = CMA >>
196   nSeg := next_ma.seg[1]
197    << i=CMA.num_segments and next_ma = NEXT_MA
198      and iSeg = CMA.seg[CMA.num_segments] and ma = CMA
199      and nSeg = NEXT_MA.seg[1] >>
200   }
201
202   LastBitOfMa:
203   GotNewMA -[on dispatch p]-> CheckMATransition
204
205   NotEndOfMA:
206   CheckMATransition -[p < ma.ea]-> GotNewMA
207
208   StartNextMa:
209   CheckMATransition -[p >= ma.ea]-> MoveForward
210   {  << AXIOM_CMA_IS_NEXT_MA() and next_ma=NEXT_MA >>
211   ma := next_ma
212   ;  <<ma=CMA>>
213   i := 1  --first segment of new movement authorization
214   ; << i=1 and ma=CMA >>
215   iSeg := ma.seg[1] --set current segment of MA to first segment
216   ; << i=1 and ma=CMA and iSeg=CMA.seg[i]>>
217   nSeg := ma.seg[2]  --set next segment of MA to second segment
218   << i=1 and ma=CMA and iSeg=CMA.seg[i] and nSeg=CMA.seg[i+1]
219     and AXIOM_NUM_SEG(ma:ma)  >>
220   }
221
222 **};
223
224 end AutoBrake.i;
225
226 end Threads;
```

7 Verification Conditions

Correctness of a thread as a whole is reduced to a set of verification conditions (VCs), which if true imply the program meets its specification for all executions. Upon command VCs are generated for every state and transition in the state-transition machine defining thread's behavior.

For a sequential program S, beginning with predicate P being true applied to program variables, will terminate with predicate Q being true applied to program variables has been traditionally represented as a Hoare triple [12]:

$$\{P\}\ S\ \{Q\}$$

The curly brackets mean "set of variable values". P is the precondition of S; Q is the postcondition of S. $\{P\}$ is the set of variable values for which P is true. If S begins in an element of $\{P\}$ it will terminate in an element of $\{Q\}$. $\{P\}\ S\ \{Q\}$ is the verification condition for S.

Because in BLESS state-transition machines, curly brackets are used for action grouping, the verification condition for S is expressed as:

$$\ll P \gg\ S\ \ll Q \gg$$

7.1 Transition VCs

Each transition in a BLESS state machine has a verification condition:

$$\ll P \wedge b \gg\ S\ \ll Q \gg$$

where P is the assertion of the source state, Q is the assertion of the destination state, b is the transition condition, and S is the action of the transition.

When source or destination state assertions are missing, P or Q default to true. Similarly for empty transition conditions. So the VC for transition `Go: Start -[]-> WaitFirstMA {r!}` is:

```
[serial 1013]: Threads::AutoBrake.i
P [53] << true >>
S [76]r!
Q [54] << true >>
What for:   <<M(Start)>> A <<M(WaitFirstMA)>> for GoStart-[ ]->WaitFirstMA{A};
```

Numbers is square brackets are line numbers. In `Threads.aadl` which has package `Threads` and thread implementation `AutoBrake.i` , line 53 holds the declaration of source state `Start` , line 54 holds the declaration of destination state `WaitFirstMA` , and line 76 holds the action `r!` which outputs an event on port `r` to request the first MA.

Transitions leaving complete states have dispatch conditions which makes the transition condition b more complex. Basically, the dispatch condition holds when dispatched (at time **now**), and dispatch has not happened since the time-of-previous-dispatch (**tops**). So the VC for transition `FirstMA: WaitFirstMA -[on dispatch p]-> CheckFirstMA` is

```
[serial 1014]: Threads::AutoBrake.i
P [79] << ( true )
  and ( p@now )
  and not ( exists u ~ time
  in tops ,, now
  that p@u  ) >>
S [78]->
Q [55] << true >>
What for:   <<M(WaitFirstMA) and x>> -> <<M(CheckFirstMA)>> for
  FirstMAWaitFirstMA-[x]->CheckFirstMA{};
```

Because `FirstMA` has no action, it defaults to implication:

$$\ll P \wedge b \gg\ \rightarrow\ \ll Q \gg$$

Transition `CheckSpeed` has an appropriately more complex VC:

```
[serial 1017]: Threads::AutoBrake.i
P [100] << ( i < CMA.num_segments
  and iSeg = CMA.seg[i]
  and nSeg = CMA.seg[i + 1]
  and ma = CMA )
  and ( p@now )
  and not ( exists u ~ time
  in tops ,, now
  that p@u  ) >>
S [101]<< i < CMA.num_segments
  and iSeg = CMA.seg[i]
  and nSeg = CMA.seg[i + 1]
  and ma = CMA
  and AXIOM_B()
  and AXIOM_E()
  and AXIOM_V(seg : iSeg)
  and AXIOM_V(seg : nSeg) >>
  if
    (v >= iSeg.v_e)~>
        {
          eb!(true)
          &
          sb!(false)
          &
          ca!(0 mpss)
        }
  []
    (v*v >= nSeg.v_e*nSeg.v_e + 2*e*( iSeg.e - p ))~>
        {
          eb!(true)
          &
          sb!(false)
          &
          ca!(0 mpss)
        }
  []
    (v >= iSeg.v_n
    and v < iSeg.v_e
    and v*v < nSeg.v_e*nSeg.v_e + 2*e*( iSeg.e - p ))~>
        {
          sb!(true)
          &
          eb!(false)
          &
          ca!(0 mpss)
        }
  []
    (v*v < nSeg.v_e*nSeg.v_e + 2*e*( iSeg.e - p )
    and v < iSeg.v_e
    and v*v >= nSeg.v_n*nSeg.v_n + 2*b*( iSeg.e - p ))~>
        {
          sb!(true)
          &
          eb!(false)
          &
          ca!(0 mpss)
        }
  []
    (v < iSeg.v_n
    and v*v < nSeg.v_n*nSeg.v_n + 2*b*( iSeg.e - p )
    and v*v < nSeg.v_e*nSeg.v_e + 2*e*( iSeg.e - p ))~>
        {
          sb!(false)
          &
          eb!(false)
          &
          ca!(x1)
        }
  fi
Q [59] << i < CMA.num_segments
  and iSeg = CMA.seg[i]
  and nSeg = CMA.seg[i + 1]
  and ma = CMA >>
What for:   <<M(MoveForward) and x>> A <<M(CheckMoveForward)>> for
  CheckSpeedMoveForward-[x]->CheckMoveForward{A};
```

7.2 Complete State VCs

Each state (except for final states) has a VC. The assertion of complete states must imply the thread invariant.

A loop invariant must be true before and after each iteration of the loop. Similarly, a thread invariant must be true before dispatch and after suspension. In `AutoBrake.i` the thread invariant is **`<<true>>`** so its VCs for complete states are trivially proved. For example, the VC for `MoveForwardLastSegment`:

```
[serial 1003]: Threads::AutoBrake.i
P [63] << i = CMA.num_segments
  and iSeg = CMA.seg[i]
  and nSeg = NULL_SEGMENT()
  and ma = CMA >>
S [39]->
Q [39] << true >>
What for:  <<M(MoveForwardLastSegment)>> -> <<I>> from invariant I
  when complete state MoveForwardLastSegment has Assertion
  <<M(MoveForwardLastSegment)>> in its definition.
```

7.3 Execution State VCs

The assertion of initial and execution states must imply the disjunction of outgoing transition conditions.

There must be a finite number of transitions between dispatch and suspension. Therefore every intervening execution state must have an enabled, outgoing transition, and loops are disallowed. For example, the VC for `CheckForLastSegment` is

```
[serial 1010]: Threads::AutoBrake.i
P [61] << iSeg = CMA.seg[i]
  and ma = CMA >>
S [61]->
Q [61] << ( i < ma.num_segments )
   or ( i = ma.num_segments )
   or ( i > ma.num_segments ) >>
What for:  Serban's Theorem:  disjunction of execute conditions leaving
  execution state CheckForLastSegment,
  <<M(CheckForLastSegment)>> -> <<e1 or e2 or . . . en>>
```

It is called "Serban's Theorem" in honor of Serban Gheorghe who inspired the theorem during discussions at an AADL Standard Committee meeting.

7.4 Verification Condition Sufficiency

Suppose proofs are constructed for every transition VC and every state VC, and there are no infinite loops among execution states. Is this sufficient to conclude correctness of the state machine as a whole?

We believe it is, but cannot *prove* sufficiency. It's definitional.

Similarly, is a proof of Hoare triple $\{P\}\ S\ \{Q\}$ sufficient to conclude S is correct? Many, including us, believe it's sufficient, but are wiling to consider counterexamples which will falsify this belief.

8 AutoBrake Proof

The proof of `AutoBrake.i` has 658 theorems. Each theorem is an axiom, given, or derived from a previous theorem by a sound inference rule. Proofs for all of the axioms, and soundness proofs for many, but not yet all, inference rules can be found in [6], for those interested.

However, here we try to show how the theorems form a convincing argument (to a person) of program correctness.

Starting from the last theorem:

```
Theorem (658)                                  [serial 1002]
P [33] <<   >>
S [39] ->
Q [33] << AutoBrake.i proof obligations >>
Why created:  Initial proof obligations for AutoBrake.i
Solved by:  Component verification conditions
and theorems 1 2 3 5 9 11 12 15 17 21 25 26 27 106 313 315 322 332 357 358 549
  550 593 595 596 657:
Theorem (1) [serial 1003] used for:
  <<M(MoveForwardLastSegment)>> -> <<I>> from invariant I when complete state
  MoveForwardLastSegment has Assertion <<M(MoveForwardLastSegment)>>
  in its definition.
Theorem (2) [serial 1004] used for:
  <<M(WaitFirstMA)>> -> <<I>> from invariant I when complete state WaitFirstMA
  has Assertion <<M(WaitFirstMA)>> in its definition.
Theorem (3) [serial 1005] used for:
  <<M(MoveForward)>> -> <<I>> from invariant I when complete state MoveForward
  has Assertion <<M(MoveForward)>> in its definition.
Theorem (5) [serial 1006] used for:
  <<M(GotNewMA)>> -> <<I>> from invariant I when complete state GotNewMA
  has Assertion <<M(GotNewMA)>> in its definition.
Theorem (9) [serial 1007] used for:
  Serban's Theorem:  disjunction of execute conditions leaving execution state
  CheckMoveForward, <<M(CheckMoveForward)>> -> <<e1 or e2 or . . . en>>
Theorem (11) [serial 1008] used for:
  Serban's Theorem:  disjunction of execute conditions leaving execution state
  CheckMoveForwardLastSegment,
  <<M(CheckMoveForwardLastSegment)>> -> <<e1 or e2 or . . . en>>
Theorem (12) [serial 1009] used for:
  Serban's Theorem:  disjunction of execute conditions leaving execution state
  Start,  <<M(Start)>> -> <<e1 or e2 or . . . en>>
Theorem (15) [serial 1010] used for:
  Serban's Theorem:  disjunction of execute conditions leaving execution state
  CheckForLastSegment, <<M(CheckForLastSegment)>> -> <<e1 or e2 or . . . en>>
Theorem (17) [serial 1011] used for:
  Serban's Theorem:  disjunction of execute conditions leaving execution state
  CheckFirstMA, <<M(CheckFirstMA)>> -> <<e1 or e2 or . . . en>>
Theorem (21) [serial 1012] used for:
  Serban's Theorem:  disjunction of execute conditions leaving execution state
  CheckMATransition, <<M(CheckMATransition)>> -> <<e1 or e2 or . . . en>>
Theorem (25) [serial 1013] used for:
   <<M(Start)>> A <<M(WaitFirstMA)>> for GoStart-[ ]->WaitFirstMA{A};
Theorem (26) [serial 1014] used for:
```

```
   <<M(WaitFirstMA) and x>> -> <<M(CheckFirstMA)>> for
   FirstMAWaitFirstMA-[x]->CheckFirstMA{};
Theorem (27) [serial 1015] used for:
   <<M(CheckFirstMA) and x>> -> <<M(WaitFirstMA)>> for
   NotYetCheckFirstMA-[x]->WaitFirstMA{};
Theorem (106) [serial 1016] used for:
   <<M(CheckFirstMA) and x>> A <<M(MoveForward)>> for
   GotFirstMACheckFirstMA-[x]->MoveForward{A};
Theorem (313) [serial 1017] used for:
   <<M(MoveForward) and x>> A <<M(CheckMoveForward)>> for
   CheckSpeedMoveForward-[x]->CheckMoveForward{A};
Theorem (315) [serial 1018] used for:
   <<M(CheckMoveForward) and x>> -> <<M(MoveForward)>> for
   SameSegmentCheckMoveForward-[x]->MoveForward{};
Theorem (322) [serial 1019] used for:
   <<M(CheckMoveForward) and x>> A <<M(CheckForLastSegment)>> for
   NextSegmentCheckMoveForward-[x]->CheckForLastSegment{A};
Theorem (332) [serial 1020] used for:
   <<M(CheckForLastSegment) and x>> A <<M(MoveForward)>> for
   NotLastSegmentCheckForLastSegment-[x]->MoveForward{A};
Theorem (357) [serial 1021] used for:
   <<M(CheckForLastSegment) and x>> A <<M(MoveForwardLastSegment)>> for
   IsLastSegmentCheckForLastSegment-[x]->MoveForwardLastSegment{A};
Theorem (358) [serial 1022] used for:
   <<M(CheckForLastSegment) and x>> -> <<M(FAIL)>> for
   PastLastSegmentCheckForLastSegment-[x]->FAIL{};
Theorem (549) [serial 1023] used for:
   <<M(MoveForwardLastSegment) and x>> A <<M(CheckMoveForwardLastSegment)>> for
   LastSegmentMoveForwardLastSegment-[x]->CheckMoveForwardLastSegment{A};
Theorem (550) [serial 1024] used for:
   <<M(CheckMoveForwardLastSegment) and x>> -> <<M(MoveForwardLastSegment)>> for
   NoMAYetCheckMoveForwardLastSegment-[x]->MoveForwardLastSegment{};
Theorem (593) [serial 1025] used for:
   <<M(CheckMoveForwardLastSegment) and x>> A <<M(GotNewMA)>> for
   GetNewMACheckMoveForwardLastSegment-[x]->GotNewMA{A};
Theorem (595) [serial 1026] used for:
   <<M(GotNewMA) and x>> -> <<M(CheckMATransition)>> for
   LastBitOfMaGotNewMA-[x]->CheckMATransition{};
Theorem (596) [serial 1027] used for:
   <<M(CheckMATransition) and x>> -> <<M(GotNewMA)>> for
   NotEndOfMACheckMATransition-[x]->GotNewMA{};
Theorem (657) [serial 1028] used for:
   <<M(CheckMATransition) and x>> A <<M(MoveForward)>> for
   StartNextMaCheckMATransition-[x]->MoveForward{A};
```

Theorem (658) says which prior theorems it depends upon, and why. It lists all complete state, execution state, and transition VCs. We believe that all listed theorems with proofs (and all the necessary theorems are in the list) form a convincing verification artifact for behavior correctness of the `AutoBrake.i` thread implementation.

Theorem (1) comes from the VC for complete state `MoveForwardLastSegment` from 7.2:

```
Theorem (1)                                [serial 1003]
P [63] << i = CMA.num_segments
  and iSeg = CMA.seg[i]
  and nSeg = NULL_SEGMENT()
  and ma = CMA >>
S [39] ->
Q [39] << true >>
Why created:  <<M(MoveForwardLastSegment)>> -> <<I>> from invariant I when
 complete state MoveForwardLastSegment has Assertion
 <<M(MoveForwardLastSegment)>> in its definition.
Solved by:  True Conclusion Schema (tc): P->true
```

For those that need convincing that anything implies true, perhaps the proof of (tc) will suffice.

Theorem (15) comes from the VC for execution state `CheckForLastSegment` using Theorem (14) which uses Theorem (13):

```
Theorem (13)                               [serial 1059]
P [61] << CMA = ma
  and CMA.seg[i] = iSeg >>
S [61] ->
Q [61] << true >>
Why created:  Less than, greater than, or equal:
|-a<b or b<a or a=b [serial 1057]
Solved by:  True Conclusion Schema (tc): P->true
```

Theorem (13) claims to be true by the True Conclusion Schema. The “Why created:” statement explains for what the theorem is used–not why it’s true.

```
Theorem (14)                               [serial 1057]
P [61] << CMA = ma
  and CMA.seg[i] = iSeg >>
S [61] ->
Q [61] << i = ma.num_segments
   or i < ma.num_segments
   or ma.num_segments < i >>
Why created:    normalization of [serial 1010]
Solved by:  Less than, greater than, or equal:  |-a<b or b<a or a=b
and theorem 13:
Theorem (13) [serial 1059] used for:
  Less than, greater than, or equal:  |-a<b or b<a or a=b [serial 1057]
```

Theorem (14) uses Theorem (13) and that

$$a < b \vee b < a \vee a = b$$

```
Theorem (15)                               [serial 1010]
P [61] << iSeg = CMA.seg[i]
  and ma = CMA >>
S [61] ->
Q [61] << ( i < ma.num_segments )
   or ( i = ma.num_segments )
   or ( i > ma.num_segments ) >>
Why created:  Serban's Theorem:  disjunction of execute conditions leaving
  execution state CheckForLastSegment,
  <<M(CheckForLastSegment)>> -> <<e1 or e2 or . . . en>>
Solved by:  Reflexivity of Equality: (a=b) = (b=a)
  Irreflexivity of Greater Than: (a>b) = (b<a)
  Add Unnecessary Parentheses For No Good Reason: a = (a)
  Reflexivity of Conjunction: (m and k) = (k and m)
  Reflexivity of Disjunction: (m or k) = (k or m)
and theorem 14:
Theorem (14) [serial 1057] used for:
    normalization of [serial 1010]
```

Theorem (15) comes from Theorem (14) and a bunch of transformations called "normalization". Proof derivation starts with a VC, applying human-selected tactics until reduced to axiom(s). Here, Theorem (15) was "normalized" to put it in normal form by removing parentheses, changing greater than to less than, and sorting reflexive operators. In the resulting proof, unnecessary parentheses are added, and operators unsorted. Conjunction is not proved to be reflexive, but *defined* to be reflexive.

Theorem (313) proves the VC for transition `CheckSpeed` using 207 theorems which is too long for inclusion in this paper. However, each theorem says why it is axiomatic, or derived from prior theorems in natural language intended to be both understood by, and persuasive to a person[1].

9 Potential Doubts

Contention that a proof is a convincing argument for its conclusion should honestly state reasons for doubt. Here, we claim that a given deductive proof means that BLESS behavior meets its specification for *every* possible execution.

Reasons for doubting our claim:

1. The executable code generated from the state machine may be incorrect.
2. The set of verification conditions generated for a state machine may be incorrect or incomplete.
3. The formal semantics of the BLESS language may be incorrect (or implemented incorrectly).
4. The built-in axioms may not be tautologies (or implemented incorrectly).
5. User-defined axioms (really givens) may be incorrect or inappropriate.
6. The inference rules may not be sound (or implemented incorrectly).
7. The specification of state machine behavior may be incorrect or incomplete.

[1] AADL model with complete set of all verification conditions and the proof script is available at https://github.com/brlarson/BLESS-models/tree/master/CTCS-3.

For (1), the usual testing of programs will detect gross compilation errors, providing at least as much confidence in correctness of compilation alone. Higher confidence could be achieved with a formally verified compiler like CompCert C, or decompilation of binary into execution graphs to compare with executions graphs of source code similar to [13].

For (2), the proof assistant could generate VCs inconsistent with their mathematical definitions in [6], the mathematical definitions themselves could be incorrect, or additional VCs not generated, should be. For example, Yannick Moy noticed a tiny gap between a thread's time-of-previous-suspension (**tops**) and the next dispatch. Because the thread is suspended, it's not changing any of its persistent variables, and new values are not loaded into its in port buffers, so there's nothing to falsify an assertion at **tops** , but there's no positive proof of this either. An option for the proof assistant allows VC generation which includes this gap. BLESS thread behaviors re-proved for the gap-less VCs required no changes in their actions, but the text in their proof outlines roughly doubled, and lengths of generated proofs similarly extended. For those who want to prove that when nothing changes assertions remain true, they can do so. If others believe that the mathematical definitions of VCs are deficient, please inform us so they may be rectified, particularly if necessary VCs are missing. Whether VCs generated by the proof assistant match their definitions can be more easily discerned.

For (3), the formal semantics of BLESS language constructs are definitional; they are correct by definition. Whether the semantics are correctly implemented by the proof assistant can be checked for theorems when either composite or atomic actions are reduced.

For (4), all of the axioms used by the proof assistant have proofs in Metamath down to the axioms of logic and set theory. Whether they have been correctly implemented in the proof assistant can be easily discerned because any theorem claiming to be an axiom says why, which can be compared with the text of the theorem.

For (5), user-defined axioms (assertions beginning `AXIOM`) are givens which help define the context of the proof. They must have special scrutiny in both their definition and use. In this example, `RMA` is the ghost variable for the movement authorization send to the Train by the RBC. Initially, `RMA` is used as the current MA, `CMA` . When the train enters the last segment, it requests MA for the next portion of track. Then `RMA` from RBC is `NEXT_MA` . Finally, when the train crosses into the next MA `CMA` becomes `NEXT_MA` . Therefore use of the axioms which state these facts must be scrutinized.

```
<<AXIOM_CMA_IS_RMA: : CMA = RMA >>
<<AXIOM_NEXT_MA_IS_RMA: : NEXT_MA = RMA >>
<<AXIOM_CMA_IS_NEXT_MA: : CMA = NEXT_MA >>
```

Similarly, the fact that for any segment the velocity at which the normal, service brake velocity is less than the emergency brake velocity is

```
<<AXIOM_V:seg~segment : seg.v_n < seg.v_e >>
```

and that ever MA must have at least two segment is

```
<<AXIOM_NUM_SEG: ma~movementAuthorization : 1 < ma.num_segments>>
```

For (6), many inference rules derived from the formal semantics of the language, hence are definitional. Soundness proofs for some of the other inference rules have proofs in Metamath, but this work is not complete, so we cannot (yet) claim soundness for the proof system as a whole. Much more likely would be incorrect implementation of inference rules by the proof assistant. That's why it's so important that theorems be human comprehensible–each theorem derived by an inference rule from prior theorems states explicitly what rule is being used, and for what the prior theorems are used.

For (7), this is the most likely cause of a proof being inapplicable. The BLESS properties of features (ports) specifying behavior might be wrong or incomplete. That's why specifications must be validated by domain experts. In [7], we show how natural language requirements can be expressed by BLESS assertions. In addition, a thread state machine may comply with its specification, but fail to do something it should. The proof shows that everything the state machine does conforms to its specification (verification)–not that it does everything its supposed to (validation). Such validation could be assisted by architecture-level simulation before any hardware is build or executable binary generated. Architecture-level simulation is but one of the capabilities of HAMR [14] with which the BLESS IDE is being integrated.

10 Conclusion

Confidence that cyber-physical systems perform as intended is enhanced by proof that executions of programs meets their specifications. Understandable proofs can be convincing arguments for program correctness–regardless of what, who, or how the proof was created. To be understandable, proofs should be written in the same language, having the same semantics, as programs themselves and clearly trace to source code they purport to prove. The BLESS Methodology was created so practising engineers and programmers could craft programs, specifications, and correctness proofs together. The BLESS Proof Assistant attempts to create proofs (given human guidance) which are convincing arguments of program correctness. Whether such arguments actually convince is entirely subjective, but that is one goal of the BLESS Methodology to which much effort is applied. Suggestions for improvement in the persuasiveness of BLESS proofs are eagerly requested.

References

[1] SAE International, "SAE AS5506D, Architecture Analysis & Design Language (AADL)," 2022.

[2] N. D. Megill and D. A. Wheeler, *Metamath: A Computer Language for Mathematical Proofs*. Morrisville, North Carolina: Lulu Press, 2019.

[3] E. Ahmad, B. R. Larson, S. C. Barrett, N. Zhan, and Y. Dong, "Hybrid annex: An aadl extension for continuous behavior and cyber-physical interaction modeling," in *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, (New York, NY, USA), p. 29–38, Association for Computing Machinery, 2014.

[4] B. R. Larson, P. Chalin, and J. Hatcliff, "Bless: Formal specification and verification of behaviors for embedded systems with software," in *NASA Formal Methods* (G. Brat, N. Rungta, and A. Venet, eds.), (Berlin, Heidelberg), pp. 276–290, Springer Berlin Heidelberg, 2013.

[5] OSATE, "Open Source AADL Tool Environment, version version 2.12.0." http://osate.org/, 2023.

[6] B. R. Larson, *Behavior Language for Embedded Systems with Software Language Reference and Proof Assistant Guide*, 2022. Version 3.

[7] B. R. Larson, "Formal semantics for the pacemaker system specification," in *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, (New York, NY, USA), p. 47–60, Association for Computing Machinery, 2014.

[8] L. Zou, J. Lv, S. Wang, N. Zhan, T. Tang, L. Yuan, and Y. Liu, "Verifying chinese train control system under a combined scenario by theorem proving," in *Verified Software: Theories, Tools, Experiments* (E. Cohen and A. Rybalchenko, eds.), (Berlin, Heidelberg), pp. 262–280, Springer Berlin Heidelberg, 2014.

[9] L. Jiang, X. Wang, and Y. Liu, "Reliability evaluation of the chinese train control system level 3 using a fuzzy approach," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 232, no. 9, pp. 2244–2259, 2018.

[10] E. Ahmad, Y. Dong, B. Larson, J. Lü, T. Tang, and N. Zhan, "Behavior modeling and verification of movement authority scenario of chinese train control system using aadl," *Science China Information Sciences*, vol. 58, pp. 1–20, Nov 2015.

[11] Ministry of Railways, *CTCS Level 3 Train Control System Requirements Specification (SRS) VI.O[M]*. Beijing: China Railway Publishing House, 2009.

[12] D. Gries, *The Science of Programming*. Monographs in Computer Science, Springer New York, 1981.

[13] T. A. L. Sewell, M. O. Myreen, and G. Klein, "Translation validation for a verified os kernel," *SIGPLAN Not.*, vol. 48, p. 471–482, jun 2013.

[14] J. Hatcliff, J. Belt, Robby, and T. Carpenter, "HAMR: An aadl multi-platform code generation toolset," in *Leveraging Applications of Formal Methods, Verification and Validation: 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, October 17–29, 2021, Proceedings*, (Berlin, Heidelberg), p. 274–295, Springer-Verlag, 2021.

Mechanizing AADL in Coq – Extended Abstract

J. Hugues
Carnegie Mellon University/Software Engineering Institute; email: jhugues@andrew.cmu.edu

Abstract

In this extended abstract, we present a mechanization of the SAE AADL language using Coq along with specific analysis capabilities. Our contribution provides an unambiguous semantics for a large set of the language and can be used as a foundation to build rich analysis capabilities. Our contribution differs from typical language mechanization, it provides support for manipulating modeling constructs, perform verification - ranging from syntactic and semantics checks to formal verification of key properties, and support simulation capabilities.

Keywords: Coq, mechanization, Ravenscar

1 Introduction

In this paper, the author is interested in the following question: "How can a modern interactive theorem prover like Coq assist in implementing a large DSML and its verification toolchain?". By large, we mean a complete DSML defined by an industry standard that would encompass static and dynamic semantics. Our contribution focuses on defining and implementing the SAE AADL language, a large DSML designed for modeling safety-critical systems. We provide the mechanization using the Coq theorem prover. The AADL language has rich static and dynamic semantics which allows one to model the architecture of real-time safety-crticial systems and perform performance, safety, or security analyses in addition to code generation. Although multiple tools exist to process AADL models, very few capture the entirety of the language dynamic semantics. The Coq theorem prover has a rich set of features to support the definition and implementation of a DSL. We selected Coq because of its large user base and set of libraries.

In the following, we show that Coq is a viable implementation strategy for defining and implementing a DMSL.

2 The Coq Theorem Prover

Coq [1] is a proof assistant, or Interactive Theorem Prover (ITP). A developer may use the Coq language, Gallina, to write mathematical definitions, executable algorithms, and theorems within an Interactive Development Environment (IDE). Coq has been used to prove non-trivial mathematical theorems and also develop formally certified software and hardware. One interesting feature of many ITPs is the capability to generate certified code (e.g. OCaml or Haskell) from Coq definitions. In this context, "certified" means that there exists a proof script – the certificate – that connects the software produced and the proofs that accompany it. Many references exist to learn more about the Coq ITP such as Pierse et al. [2]. In the following, we introduce a minimal set of notions prior to introducing our Coq development.

Coq relies on Gallina, a functional language that acts as a specification language to describe types, functions, and proofs of some statement. Coq's core follows the rules of the Calculus of Inductive Constructions. The basic types in Coq are either propositions, `Prop`, or sets `Set`. All other types are built on top of these two types. `Prop` is the type of logical propositions. It denotes the class of terms representing proofs. `Set` represents typical data types.

In Listing 1, we show some fragments of Coq. First, we define the `component` inductive type, the type definition lists the elements of a component definition: identifier, category, etc. This `component` type is recursive: one of its elements is actually a list of `component`s. Then, we define the `Unfold` function that recursively build the list of components and subcomponents by iterating over subcomponents, with `++` being the list concatenation operator.

Listing 1: A Coq inductive type

```
Inductive component :=
 | Component : identifier →
    ComponentCategory → (* category *)
    fq_name →           (* classifier
        *)
    list feature →      (* features *)
    list component →    (*
        subcomponents *)
    list property_association →
    component
 (* .. *)

Fixpoint Unfold (c : component) : list
    component :=
  c ::
  (( fix Unfold_subs (ls : list component) :=
    match ls with
    | nil ⇒ nil
    | c :: lc ⇒ Unfold (c) ++Unfold_subs (lc
        )
    end ) (c→ subcomps)).
```

Coq has a rich standard library that defines regular types such as booleans, integers, lists, and typical results on them. Coq type system support polymorphisms and Java-like interfaces (typeclasses). Because `Prop` and `Set` are both types, they can be part of expressions. One of the key aspects of Coq is that a proposition is more powerful than a boolean expression.

The type `bool` is computational: one can define functions or perform case analysis. On the other hand, the type `Prop` supports universal quantification over elements, that is typical exists ($\exists$) and for all ($\forall$) quantifiers.

Writing proofs can be a repetitive task. Coq provides a rich library of tactics that encode specific reasoning steps. A typical example is `auto` that will prove basic propositions by applying well-known facts and computations. Coq's set of tactics can be extended by the user using the LTAC language.

3 AADL Mechanization – Core Concepts

3.1 AADL generic component model

At its core, AADL relies on a restricted set of concepts to support its constructs. Formally,

Definition 1 *An AADL component Comp is a tuple* $(c_{id}, cat_C, F, Prop, S)$ *s.t.*

c_{id}*: the unique component id,*

cat_C*: the component category,*

F*: the set of features,*

$Prop$*: the set of properties associated with this component,*

S*: the set of subcomponents.*

The mechanization of this type in Coq relies on inductive type definitions, i.e. the capability to define a new type from constants and functions that create terms of that type. We illustrate this in the code snippet shown in Listing 2. First, we define `ComponentCategory`, it is an enumeration type that lists all possible component categories. We then provide a joint definition for components, features, and connections. Because these types are mutually dependent, they must be defined in conjunction. The types `DirectionType` and `FeatureCategory` are also enumerations, whereas `identifier`, `fq_name`, and `feature_ref` are variants of string-like type to denote respectively an identifier, a fully qualified name, and a patch (a list of identifiers). They are omitted for conciseness.

The definitions shown in Listing 2 are understood as follows. A connection can be built using the `Connection` constructor and three parameters: an identifier and two feature references.

Another key aspect of an AADL model is the concept of property association, linking a property value (e.g. 42) to a property type (e.g. `Priority`) that applies to component of a specific category such as threads. The definition of those types follow the same pattern.

An important concept in Coq is that programs and proofs are equivalent. This applies to all concepts, including equality. Coq relies on so-called Leibniz equality which states that, given any x and y, $x = y$ if and only if, given any predicate P, $P(x)$ if and only if $P(y)$. Building such proof could be seen as a tedious and repetitive effort. Fortunately, Coq provides appropriate tactics to derive these equality principles automatically.

Listing 2: A Coq inductive type

```
Inductive ComponentCategory : Type :=
 | system | abstract | process | thread
 (* [..] *)

 Inductive component :=
  | Component : identifier →
     ComponentCategory → (* category *)
     fq_name →           (* classifier
         *)
     list feature →      (* features *)
     list component →    (*
         subcomponents *)
     list property_association →
     component
 with feature :=
  | Feature : identifier →
      DirectionType →
      FeatureCategory →
      component →
      list property_association →
      feature
 with connection :=
  | Connection : identifier →
      feature_ref →
      feature_ref →
      connection.
```

3.2 AADL concepts as a Coq DSL

Using Coq constructors to build a full AADL model results in cumbersome syntax constructs as shown in Listing 3. A solution is to use Coq notations: advanced commands to modify the way Coq parses and prints objects. A notation is an alternate syntax for entering or displaying a specific term or term pattern and is well-used [3].

Listing 3: Building AADL components

```
Example A_Component := Component
(Id "a_component")
(abstract)
(FQN [Id "pack1" ] (Id "foo_classifier")
    None)
nil nil nil nil.
```

A notation is a construct similar to regular expressions that maps elements to arguments of some constructors. This is shown in the code snippet of Listing 4. In this example, we present the notation to define an abstract component and build the corresponding Coq term along with an example of use. Coq notations can be combined. In the example below, we also use the notation for representing lists without explicitly using the `cons` constructor.

With this notation, we provide a way for the user to define AADL models. We devised the notations to be close to the AADL original syntax.

An alternative strategy, not presented in this paper, is to import a JSON serialization of an AADL model as a Coq type and then map it to the corresponding AADL declarations.

Listing 4: Notation for AADL

```
(* Coq notation for defining an AADL
    abstract component *)
 Notation "'abstract:'
 id → | classifier
 features: lf
 subcomponents: ls
 connections: lc
 properties: lp" :=
 (Build_Component id abstract classifier
     lf ls lc lp)
 (at level 200).

Example A_Component_2 :=
  abstract: "a_component" → | "pack1::
      foo_classifier"
    features: [
      feature: in_event "a_feature" ;
      feature: out_event "a_feature2" ]
    subcomponents: [
      thread: "a_thread" → | "pack2::
          thread_t"
        features: nil
        subcomponents: nil
        connections: nil
        properties: nil ]
    connections: [
      connection: "c1" * "a_feature" −→ "
          a_feature_2" ]
    properties: nil.
```

3.3 AADL rules as decidable properties

The previous elements can be used to build AADL models elements that are typed statically by Coq rules. The AADL standard defines additional rules that judge the correctness of an AADL model through naming, legality, and consistency rules.

These rules, in their most general form, are defined in the standard as natural language predicates. For instance naming rule $4.5(N1)$ states that "The defining identifier of a subcomponent declaration placed in a component implementation must be unique within the local namespace of the component implementation that contains the subcomponent." In other words, the list of identifiers built from the list of subcomponents has no duplicates.

Such a predicate has a direct translation in Coq: we build the list of the subcomponent identifiers and check there is no duplicate. This rule is obviously decidable: one can derive a proof for either the true or false case by computing the term explicitly.

We implemented most rules defined by the AADL standards. They have a uniform pattern: a predicate that combines conjunctions or disjunctions of basic predicates and the evaluation of properties on lists (test for duplicated elements, inclusion, etc.). This is a direct consequence of the general structure of the AADL and Coq languages. AADL model elements are components, and their well-formedness depends on the way their features, connections, and subcomponents are defined, all of which are built on top of Coq basic types and lists. Hence, the implementation in Coq of these rules has no practical difficulty. Likewise the proof of their decidability that has been partially automated thanks to Coq tactics mechanism.

In addition, we defined a specific induction principle to extend the previous rules to the case of a hierarchy of components. This induction scheme is similar to a visitor pattern that will walk through a model and check that each node is defined according to language rules.

Although this step yields little implementation challenge, it proved to be an interesting step to assess AADL rigor in defining its semantics. First, it confirmed that all rules are defined with sufficient details and can be written as formal propositions. Second, it confirmed that all rules are unambiguous and decidable, which is stronger than boolean-based implementation used in other AADL tools. In a few situations, we engaged with the AADL standardization body to clarify the semantics of rules and their order of evaluation.

3.4 Coverage of AADL concepts

One of our goals is to support an extensive subset of AADL as Coq elements. In this section, we introduced the key elements of our mechanization. Our mechanization allows for the definition of AADL property sets and packages. We implemented Coq types, functions, and lemmas that cover the AADL instance model, and the definition of property sets. We provide a full library to manipulate AADL concepts as either native Coq types or through notations.

Our mechanization covers all AADL concepts except for flows and modes. Flows are annotations on an AADL architecture to mark a particular data flow for an analysis. We do not consider them given their special usage. Modes are specific configurations of a system, e.g. specific values for property values of activated/deactivated components and connections at specific time. The support of modes is a contending issue in the standard. The authors' analysis is that the current definion of modes is not precise enough to warrant a mechanization effort.

The resulting Coq library can be used in two different ways. First, code can be extracted to validate the JSON-serialization of AADL models produced by AADL third-party tools like Ocarina [4]. Code extraction consider only the elements that are in the transitive closure of a main function. This includes typical functions (e.g. model navigation or name resolution), but also decision procedures that judge that a component is valid that is defined as a decidable proposition. All rules stated in the standard fall into this category, and we could generate a proof-carrying oracle that checks the conformance of an AADL model to the standard. This capability allows us to derive a validity checker for AADL models that is deemed sound by construction: it reflects AADL semantics defined in a mathematically grounded framework.

Another option is to use Coq as an interactive theorem prover so that one can proof more complex lemmas. We discuss this in the next section.

4 Conclusion

In this paper, we presented the main elements of the mechanization of AADL in Coq. Our contribution delivers the mechanization of a significantly large subset of AADL along with verification capabilities. We used multiple features of Coq to define an AST-like structure as a Coq inductive type representing an AADL instance model, along with accessor functions to build decidable properties. We exercised this on the verification of schedulability for some classes of real-time processors on mono-processor systems.

The resulting Coq development represents approximatively $12KSLOCS$ and include examples and testsuite. It demonstrates a first step towards the full mechanization of AADL in Coq. Our development is available as an Open Source software at `https://github.com/Oqarina/`.

Acknowledgments

Copyright 2024 ACM. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. [DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution. DM24-0684

References

[1] Y. Bertot and P. Castéran, *Interactive theorem proving and program development - Coq'Art: The calculus of inductive constructions*. Texts in theoretical computer science. An EATCS series, Springer, 2004.

[2] B. C. Pierce, A. Azevedo de Amorim, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjöberg, and B. Yorgey, *Logical Foundations*. Software Foundations series, volume 1, Electronic textbook, May 2018.

[3] C. Pit-Claudel and T. Bourgeat, "An experience report on writing usable DSLs in Coq," 2021.

[4] J. Hugues, B. Zalila, L. Pautet, and F. Kordon, "From the prototype to the final embedded system using the ocarina AADL tool suite," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 4, pp. 42:1–42:25, 2008.

Extension of the TASTE Toolset to Support Publisher-Subscriber Communication

Hugo Valente, Miguel A de Miguel, Ángel G Pérez, Alejandro Alonso, Juan Zamorano, Juan A de la Puente

Universidad Politécnica de Madrid, Madrid, Spain; email: hugo.svalente@upm.es

Abstract

New Space has been revolutionizing how space software is developed. While in the past the development of systems lasted years to minimize errors, nowadays, with the reduction in manufacturing costs of micro and nanosatellites, companies are capitalizing by focusing on launching frequently, focusing on rapid iterations and innovations. As a result, software development of space systems is also adapting to meet the demands from platform specific code to code that can be reused across different platforms. As a result, there has been a shift from developing tightly coupled client-server systems to loosely coupled publisher subscriber systems.

In this article, we propose a solution focused on integrating cFS, a Publisher Subscriber runtime made by NASA, inside TASTE, a model-based toolset developed by ESA, to build space systems focusing on a publisher-subscriber methodology, allowing the user to develop platform agnostic components, allowing for faster iterations, reducing the development time and increasing portability and reusability.

Keywords: automatic code generation, publisher-subscriber architecture, model-driven development, component-based design, new space.

1 Introduction

Due to considerable reductions in the manufacturing and launch costs associated with satellite miniaturization, teams are more willing to take risks and innovate than they were in the past. This paradigm change is known as New Space [1].

While the cost of building and launching satellites has decreased, the complexity and size of the software, as well as its price, has been increasing.

It can be quite expensive to start a flight software project from scratch, which serves as a barrier for businesses with limited financial resources. This can be managed with the use of toolchains and frameworks. As a result, better-quality software can be developed with less time and effort, lowering the barrier to entry for new companies in the space industry [2].

TASTE, or The Asserted Set of Tools for Engineering, was created by the European Space Agency (ESA) to enable the use of Model-Based Systems Engineering (MBSE) for heterogeneous, embedded, real-time systems. It offers the ability to develop, assemble, test, and deploy embedded technologies [3].

Core flight system (cFS) is a framework developed by NASA. The cFS architecture consists of three essential concepts: a layered architecture, a dynamic run-time environment, and a component-based design. The combination of these features allows for a bigger reusability and portability of components across different projects and platforms, resulting in a reduction in the development time and cost of space software [4], [5].

In this article we explain how TASTE was modified to support a Publisher Subscriber communication pattern using cFS as the runtime.

2 Toolset modifications

The original TASTE toolchain follows a client-server communication paradigm with PolyORB-HI as the middleware.

In our modified TASTE toolset, in order to support a publisher-subscriber communication paradigm, PolyORB-HI has been replaced with the cFS middleware, as shown in Figure 1.

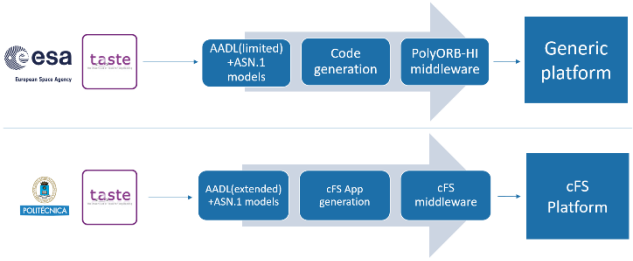

Figure 1 – Comparison original TASTE vs cFS TASTE

Taking advantage of the TASTE code generation capabilities, the metamodel and templates were extended to generate code that takes advantage of the communication functionalities provided by cFS. The extension process followed was the one described in [6] by the authors.

2.1 Graphical representation

The main tool that the user interacts with is Space Creator, the graphical user interface (GUI), to build the system description. Consequently, although it has no impact on the code generated, visual experience is an important consideration when adding new functionalities.

As a result, the GUI has been extended to show the flow of data between components, with the message icon represented based on the SAVOIR notation, as shown in Figure 2.

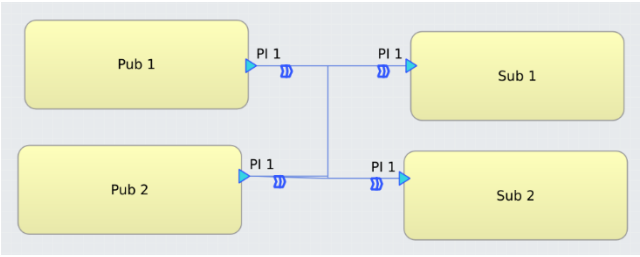

Figure 2 – Publisher subscriber cFS representation

2.2 AADL representation

TASTE relies on AADL for the automatic code generation. As a consequence, the current property set has been extended with new properties to support the new functionalities.

Listing 1 shows the new property set.

```
-- Message properties

MessageContent: aadlstring applies to
(subprogram);

MessageID: aadlinteger applies to
(subprogram);

MessageSize: aadlinteger applies to
(subprogram);

MessageType: aadlstring applies to
(subprogram);

MessageDirection: enumeration
(PUB,SUB) applies to (subprogram);
```

2.3 Kazoo

Kazoo is responsible for parsing the generated AADL model description and rendering the templates.

Since the property set has been modified, Kazoo needs to be modified to parse the new properties.

Listing 2 uses the example of the parsing of the message ID to show how properties are parsed using Kazoo.

```
--------------------
-- Get_Message_ID --
--------------------
function Get_Message_ID (D : Node_Id)
return String is

    Event_Name : constant Name_Id :=
        Get_String_Name
        "taste::messageid");
begin

    return Get_Integer_Property (D,
    Event_Name);

end Get_Message_ID;

[…]

function Parse_Interface (If_I :
Node_Id) return Taste_Interface is

 […]

    Result.Message_ID := US (
    Get_Message_ID (If_I));
```

Finally, new templates were created to make the necessary API calls to provide new functionality. An excerpt of the initialization of a message or subscription to a message pipe depending on whether the component is a publisher, or a subscriber is shown in Listing 3.

```
@@IF@@ @_Direction_@ = "PUB" and
@_Kind_@ = MESSAGE_OPERATION

// Initialize message @_Name_@
@_UPPER:Parent_Function_@_Data.MsgId_@
_Name_@ =
@_UPPER:Parent_Function_@_APP_COMMUNIC
ATION_MID_@_Message_ID_@;
@_UPPER:Parent_Function_@_Data.Size_@_
Name_@ = asn1Scc@_REPLACE_ALL((-
)/_):Param_Types_@_REQUIRED_BITS_FOR_E
NCODING;

CFE_MSG_Init(&@_UPPER:Parent_Function_
@_Data.MsgPtr_@_Name_@,
@_UPPER:Parent_Function_@_Data.MsgId_@
_Name_@,
@_UPPER:Parent_Function_@_Data.Size_@_
Name_@);

@@ELSIF@@ @_Kind_@ = MESSAGE_OPERATION

    // Initialize pipe @_Name_@
    // Subscribe to @_Name_@
0x0@_Message_ID_@ msg id

CFE_SB_Subscribe(0x0@_Message_ID_@,
@_UPPER:Parent_Function_@_Data.Command
Pipe);

@@END_IF@@
```

3 Case study UPMSat2

The UPMSat2 system was built using the new communication paradigm moving from a client-server-oriented architecture to a publisher-subscriber communication architecture.

The resulting system is shown in Figure 3.

The Manager is responsible for setting the operation mode of the satellite. The Telemetry, Tracking and Command (TTC) is responsible for receiving the telecommands from the ground station and sending telemetry data back.

The Sensors are responsible for gathering data of the environment. The Storage is responsible for storing the data gathered by the sensors. The Platform is responsible for checking the sensor data received to ensure correct functioning of the sensors. The Simulated ACS HW, Attitude Control System Hardware is responsible for

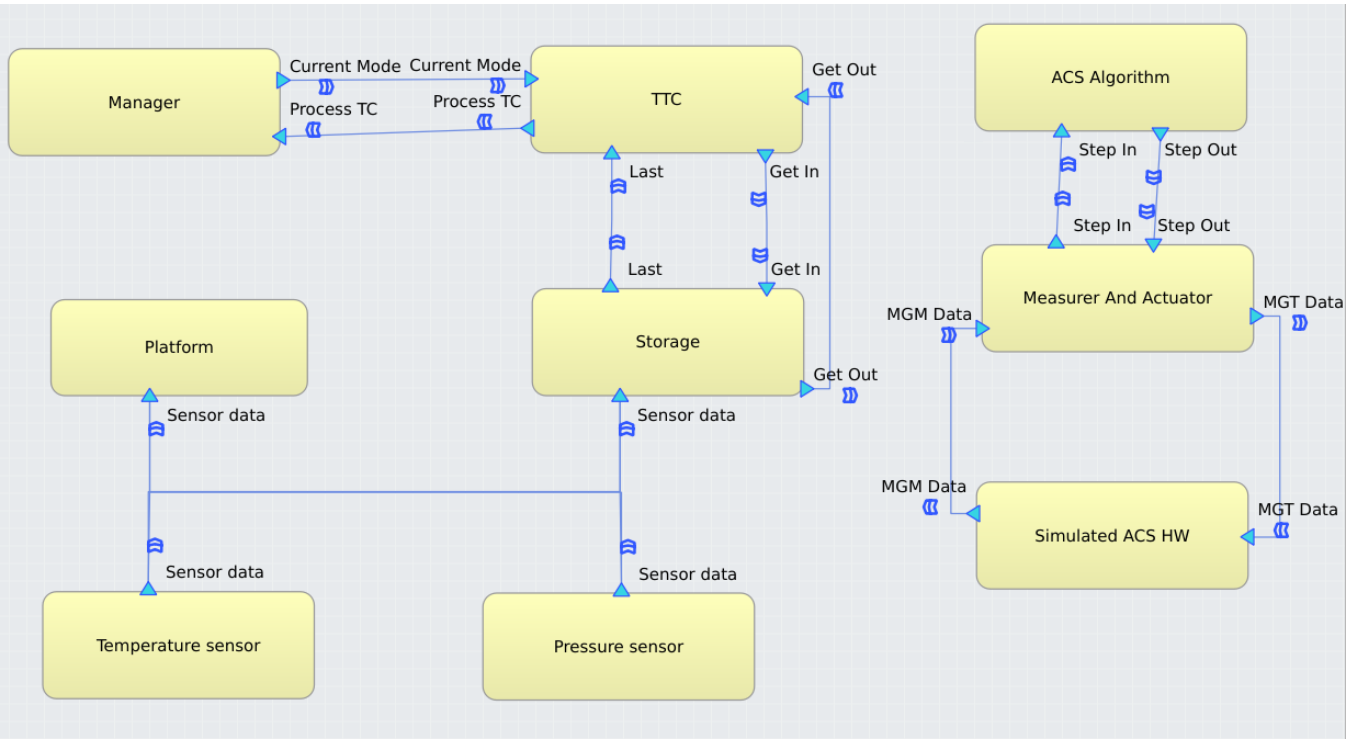

Figure 3 – Case Study UPMSat2

gathering the data from the magnetometers and activating the magnetorquers to change the current satellite attitude. The ACS Algorithm is a Simulink algorithm designed to do the calculations necessary to correct the attitude. The Measurer and Actuator is the middleman between the two components mentioned above, responsible for sending the magnetometer data, receiving the corresponding magnetorquer data and sending it back to the first component in order to change the satellite attitude.

The proposed system enables demonstrate the use of new technologies for special use.

The experimental results demonstrate that this communication paradigm can significantly improve the portability, reusability and scalability of the components of the system when compared with the traditional client-server TASTE approach by allowing a loosely coupled N to M publisher-subscriber communication

Conclusions

In this article, we proposed a modification to the TASTE toolchain to support the publisher-subscriber communication paradigm. This functionality is supported with the addition of the cFS runtime to the TASTE toolchain. A case study has been presented validating the proposed solution.

Acknowledgements

The work described in this paper has been developed within the European project AURORA (101004291), and the project OAPES-CM (Y2020/NMT-6427). We would like to acknowledge the financial support of the European Union's H2020 R+I programme, and the Comunidad de Madrid (Spain) Proyectos Sinérgicos de I+D plan, as well as the collaboration with the partners in both projects. The authors are indebted to the Horizon 2020 IOD/IOV Programme of the European Union that funded the UPMSat-2 launch.

References

[1] D. Paikowsky, "What Is New Space? the Changing Ecosystem of Global Space Activity," *67th International Astronautical Congress*, vol. 20, no. 20, pp. 84–88, 2017, doi: 10.1089/space.2016.0027.

[2] D. J. F. Miranda, M. Ferreira, F. Kucinskis, and D. McComas, "A comparative survey on flight software frameworks for 'new space' nanosatellite missions," *Journal of Aerospace Technology and Management*, vol. 11, 2019, doi: 10.5028/jatm.v11.1081.

[3] M. Perrotin, E. Conquet, P. Dissaux, T. Tsiodras, and J. Hugues, "The TASTE Toolset: turning human designed heterogeneous systems into computer built homogeneous software.," *European Congress on Embedded Real-Time Software (ERTS 2010)*, pp. 1–10, 2010.

[4] CCSDS, *Space Packet Protocol: Recommended Standard*, no. CCSDS 133.0-B-2. Washington, DC: The Consultative Committee for Space Data Systems, 2020.

[5] NASA, "NASA cFS." https://github.com/nasa/cfs (accessed Jun. 23, 2022).

[6] H. Valente *et al.*, "Extension of the Modeling Tool Suite for Development of Embedded Systems for the Space Domain," *IFAC-PapersOnLine*, vol. 55, no. 4, pp. 286–291, 2022, doi: 10.1016/j.ifacol.2022.06.047.

METASAT's Model Based Design Solutions

Leonidas Kosmidis
Barcelona Supercomputing Center (BSC) and Universitat Politecnica de Catalunya (UPC); email: leonidas.kosmidis@bsc.es

Abstract

METASAT is a recently started project (January 2023) in the Horizon Europe programme, in the SPACE call, coordinated by the Barcelona Supercomputing Center (BSC). METASAT will develop model-based design (MBD) solutions for high performance on-board processors such as multicores, Graphics Processing Units (GPUs) and Artificial Intelligence (AI) Accelerators. While the developed tools and methodologies are particularly focusing on the space domain, reusability to other safety critical domains is also a project goal. This talk will provide an overview of the solutions which will be developed during the project, which will be centered around the open source TASTE framework used at the European Space Agency (ESA), which leverages AADL.

1 Talk Details

Modern space systems require increasing levels of performance for the implementation of advanced functionalities such as high performance on-board processing and the incorporation of artificial intelligence algorithms which will enable a higher level of autonomy. In fact, there is general trend of moving functionality that once was available only on ground segments, to on-board space systems. However, these functionalities cannot be provided with existing conventional hardware architectures used in current space systems. Instead, there is a need to adopt new multi-core systems and high performance hardware elements such as Graphics Processing Units (GPU) and Artificial Intelligence (AI) accelerators which have been proven very efficient in many domains of embedded systems on ground applications. Apart from the hardware complexity, the implementations of these features require complex, usually low-level parallel programming models such as SIMD (Single Instruction, Multiple Data) intrinsics, OpenMP, OpenCL and other safety critical oriented ones like Brook Auto, Vulkan SC or the upcoming SYCL SC (Safety Critical).

The METASAT consortium which consists of the Barcelona Supercomputing Center, IKERLAN, fentISS, Collins Aerospace and OHB will address these challenges by employing a model-based design approach, which is widely used in all safety critical domains, and specifically in aerospace. METASAT will rely on the TASTE [1] framework, which has long heritage of use at ESA, as well as in prior EU funded projects within the PERASPERA cluster.

The METASAT MBD solutions will be demonstrated on a novel, prototype hardware platform for future high-performance on-board processing currently developed by BSC, which will consist of a high-performance multicore platform based on FrontGrade Gaislers's next generation space processor based on RISC-V, NOEL-V, extended with the SPARROW SIMD AI accelerator [2] and coupled with an open source, RISC-V based GPU, Vortex [3]. The METASAT hardware platform will be open source and implemented on an FPGA, and will be capable of mixed-criticality workloads, using the Xtratum hypervisor.

During the project, the TASTE code generation capabilities from AADL models, which rely on Ocarina [4] will be extended in multiple ways. First, existing support will be extended to cover support of the RISC-V compilation and emulation infrastructure, including the SPARROW accelerator. Next, code generation for necessary multicore support using RTEMS SMP and Xtratum multicore partitions will be added. Integration with parallel programming models such as SPARROW SIMD intrinsics, OpenMP and at least one GPU programming API and (Machine Learning) ML framework will be implemented. All these modifications will be submitted for inclusion in the official TASTE repository.

Finally, the solutions will be evaluated using two open source aerospace ML use cases developed by BSC for ESA – a cloud screening and a ship detection application from satellite images – as part of the OBPMark benchmarking suite [5] as well as with an industrial use case provided by OHB.

Acknowledgments: This work was supported by the European Community's Horizon Europe programme under the METASAT project (GA 101082622).

References

[1] M. Perrotin, E. Conquet, J. Delange, and T. Tsiodras, "TASTE: An Open-source Tool-chain for Embedded System and Software Development," in *ERTS*, 2012.

[2] M. S. Bonet and L. Kosmidis, "SPARROW: A Low-Cost Hardware/Software Co-designed SIMD Microarchitecture for AI Operations in Space Processors," in *DATE*, 2022.

[3] B. Tine et al, "Vortex: Extending the RISC-V ISA for GPGPU and 3D-Graphics," in *MICRO*, 2021.

[4] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues, "Ocarina : An Environment for AADL Models Analysis and Automatic Code Generation for High Integrity Applications," in *Ada-Europe*, 2009.

[5] D. Steenari, L. Kosmidis, I. Rodriguez-Ferrandez, A. Jover-Alvarez, and K. Förster in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 2021. https://doi.org/10.5281/zenodo.5638577.

Facilitating AADL Model Processing and Analysis with OSATE-DIM

Rakshit Mittal

Universiteit Antwerpen, Middelheimlaan 1, 2018 Antwerp, Belgium.; email: rakshit.mittal@uantwerpen.be

Dominique Blouin

Telecom Paris, Institut Polytechnique de Paris, Place Marguerite Perey, 91120 Palaiseau, France.; email: dominique.blouin@telecom-paris.fr

Abstract

The Architecture Analysis and Design Language (AADL) is a rich component-based language for modelling embedded systems. To ease processing AADL models, OSATE, the reference tool for AADL, provides the 'instance' model derived from base 'declarative' model/s. An instance model represents the operational view of a declarative model in a simple object tree where information is flattened (with no component extensions / refinements) so that tools can easily analyze the model. Note that information is lost in instantiation. Since the instance model is a (un-symmetric) 'view' of the declarative model, the capability to directly modify the instance model requires a solution to the view-update problem. We demonstrate the OSATE Declarative-Instance Mapping Tool (OSATE-DIM) to perform incremental deinstantiation in AADL. OSATE-DIM significantly eases the development of AADL model processing tools for analysis and code generation.

Keywords: view-update problem, AADL, deinstantiation, RAMSES.

1 Introduction

The Architecture Analysis and Design Language (AADL) [1] is used to model real-time embedded systems composed of software and execution platform components tightly coupled with actuators and sensors to interact with the physical world. It is standardized by the Society of Automotive Engineers (SAE-AS5506D) to be primarily used for scheduling/flow-control analyses and code generation for various embedded platforms. It is supported by the Open-Source AADL Tool Environment (OSATE), which is the reference tool for AADL, and released under the Eclipse Integrated Development Environment (IDE).

OSATE includes two meta-models: *Declarative* and *Instance*. Factorization of component declarations in AADL is made possible through constructs like component extensions, refinements and different levels of component abstractions allowing for a rich specification of the structural and behavioral characteristics of embedded systems in the *Declarative* part of the language.

This richness of the declarative sub-language complicates the analysis of an AADL model. To solve this, OSATE provides the simpler *Instance* metamodel extending the declarative metamodel. An *Instance* model represents the run-time configuration of a system. It is generated from the original *Declarative* model through a transformation called *Instantiation*. During *Instantiation*, all properties of the system components and their elements like *Features*, *Connections* and *Modes* are collected from all parent classifiers/specifications, and collapsed into one entity. The architecture of the system is represented in the *Instance* model through a containment tree of components and other elements. Traces in the form of references relate the generated *Instance* elements to their corresponding *Declarative* elements.

Many AADL analysis tools work with the *Instance* model, since all information is readily available in a single tree of objects. Frequently, results computed by these tools must be set in the original system model (e.g. static scheduling tables computed by our MC-DAG tool [2] for automatic code generation), or tools may even change the structure of the system by splitting or merging components following the application of some design patterns (e.g. RAMSES [3], an AADL-based tool developed by our group that refines a model taking into account a specific target operating system and generates code from the refined model).

In the current OSATE, *Instance* models can be updated manually or by tools. However, this is not recommended because there are no means to reflect the changes back to the *Declarative* model. This situation makes an *Instance* model an *updatable view* of the corresponding AADL *Declarative* model(s), and our problem becomes similar to the well known view update problem for databases [4] and for Model-Driven Engineering (MDE) [5]. Synchronization of these view updates (changes in *Instance* model) back to the base model (*Declarative* model) is an important task to maintain consistency and reliability of knowledge/data throughout the development process (Fig. 1). Currently, tools must always take care of setting their results at the right place in the *Declarative* model, which is very cumbersome. Furthermore, the *Instance* model must again be regenerated to remain consistent with the updated *Declarative* model.

It is also more natural / intuitive for systems engineers to

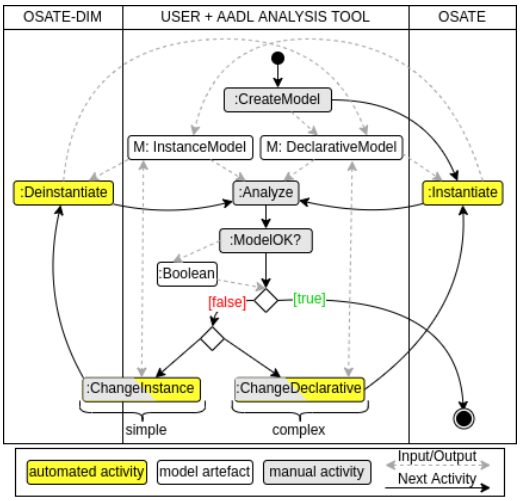

Figure 1: Process diagram of AADL analysis

work in a top-down approach. The information in *instance* model is clear and concise because it is flattened. Hence, it will be easier for an engineer to design the AADL model of the system-under-study from its *instance* model rather than the *declarative* model.

Therefore, providing an automated *Instance* model *Deinstantiation* capability will be very beneficial for users of the growing AADL community, since it will allow tools to process the simpler *Instance* model directly. This is a crucial advantage especially that AADL is being more and more used in industry. For instance, it is ranked among one of the most used architecture description languages in industry [6] and even the US Department of Defense has made it the heart of its Digital Engineering Strategy [7].

2 OSATE-DIM

OSATE-DIM has been developed with the following key objectives: maximum information preservation, minimal/simplest change in the declarative model, and maximum efficiency, while providing flexibility for users to take part in the *Deinstantiation* decision algorithms.

OSATE-DIM supports *Deinstantiation* in many different scenarios. A *View-Update*, may be an in-place or an out-of-place transformation. Consequently, the *Model-Update*, should be an in-place or out-of-place transformation respectively. The transformation rules for all the scenarios are the same. The difference is only in the interface to the transformation rules.

Firstly, OSATE-DIM supports a state-based deinstantiation scenario as shown in Fig.2.(i). This state-based case is derived from the delta-based case as a pure backward transformation when it is assumed there is no original *Declarative* model. In this scenario, OSATE-DIM takes all information from the *Instance* and creates the simplest *Declarative* model from that information. By comparing the output of this scenario with the original *Declarative* model (if it exists) used to create the *View*, the user can also understand what kind of information is lost in the *Instantiation* transformation.

Modifications of an instance model can be carried out in two ways. In an in-place transformation, the changes are made directly in the model. Hence, for an in-place *View-Update*, the corresponding *Update* in the base *Declarative* model should be in-place as well. That is, it should make changes directly on the *Declarative* model. OSATE-DIM detects the changes in this scenario, using VIATRA's built-in transformation engine (see section 3) that listens for changes to query patterns in the *View* model as shown in Fig.2.(ii).

The scenario where an entirely new *Instance* model (with modifications) is computed from the base *Instance* model, is the out-place scenario as shown in Fig.2.(iii). In this case, a new corresponding *Declarative* model should be constructed reflecting the modifications, instead of directly modifying the *Declarative* model.

In this scenario, the *View-Updates* can be computed from differences between the new and original *Instance* models. OSATE-DIM provides the delta-trace model to define trace relations between an instance model and its out-of-place update. The delta-trace model borrows concepts from EMF Change [8] to store information regarding the specific change operations that were performed on the *Instance* model to lead to the new *Instance* model.

3 Implementation

The proposed tool, OSATE-DIM, has been implemented as a set of Eclipse IDE-based plugins. The source code for the tool is available in a GitLab repository[1]. Users can install this tool into their Eclipse installations through an update-site[2].

OSATE-DIM uses VIATRA[3] for executing graph-transformations. The *Instance* and *Declarative* models are graphs where objects are nodes, and their relationships are edges. VIATRA is a scalable reactive framework, which allows for incremental execution of transformations. Incrementality is offered by separating the pattern matching and transformation steps. The transformation uses information of each pattern (and its state of creation, updation, or deletion) as input. When the state of a pattern changes, the corresponding transformation rules are 'fired'. A new match for a pattern is a creation, the disappearance of a match is a deletion, and a change in the properties of objects in the match is an update. Patterns are specified through a Domain-Specific Language (DSL) called Viatra Query Language. The transformations are written in an Xtend-based DSL providing a model manipulation Application Programming Interface (API).

OSATE-DIM is packaged as two plugins, one for the user-interface, and the other of for providing the core transformation rules and framework. The transformations can be run in a standalone mode, i.e. without the need for the Eclipse IDE, as is done in the test classes (provided in a separate 'test' package).

[1] https://gitlab.telecom-paris.fr/mbe-tools/osate-dim/

[2] https://mem4csd.telecom-paristech.fr/download/update-site/osate-dim/

[3] https://projects.eclipse.org/projects/modeling.viatra

Figure 2: Possible AADL deinstantiation scenarios supported by OSATE-DIM

4 Validation and Scope

OSATE-DIM is validated through two case studies.

The first case-study is taken from the MC-DAG framework where the change on *Instance* models consists of addition of various mixed-criticality static scheduling tables and time-related properties to various components of an unmanned aerial vehicle control system. These additions need to be reflected on the declarative side in the appropriate AADL elements.

The second case-study is an example from the RAMSES tool, which consists of various refinements of an instance model corresponding to design patterns. This is a more complex example which includes additions, deletions, and element updates for the outplace scenario (figure 2). The RAMSES refinement rules are described in detail in [3].

Scenarios were prepared for both the case studies, and the OSATE-DIM tool was invoked for deinstantiation. Successful deinstantiation was verified by instantiating the generated declarative model, which should be equivalent to the initial changed instance model. The entire test suite is also made available in the tool repository as a specific test project.

It is to be noted that the current scope of OSATE-DIM is limited to component instances, features, connections, properties, and modes, which are the major and most frequent AADL constructs. We intend to integrate support for deinstantiation of flow-specifications in later versions of the tool. It is also envisioned to give users greater flexibility in terms of level of modification of the declarative model in future iterations. While the current implementation of OSATE-DIM is designed with the principle of minimal changes, adding more information to the deinstantiated *declarative* model could be more desirable. For example, the engineer may not desire any changes to the original component declarations of the *declarative* model (for example, if the component is imported from a component library which should not be modified). In this case, new component declarations need to be added for every change recorded, which is not a minimal change.

More information about the tool is available on the OSATE-DIM web-site[4]. The tool is also listed on OpenHub[5]. Further clarification of the technical concepts is given in [9]. A demonstration of the tool was also presented in [10]. The overall approach used for *Deinstantiation* within OSATE-DIM can also potentially be applied for de-compilation of various High-Level Languages for example from Rust, Swift, or C/C++ to LLVM-IR [9].

[4] https://mem4csd.telecom-paristech.fr/blog/index.php/osate-dim/

[5] https://openhub.net/p/osate-dim

5 Discussion

While developing OSATE-DIM we experienced some shortcomings and unnecessary complications of AADL that need to be discussed within the community:

1. The current *Declarative* metamodel is highly complicated, since it includes specific classes for each category of *Component*, *Feature*, and *Connection*. This was done to introduce constraints on component composition. The Object Constraint Language (OCL) could be used to expressed these constraints instead of component category subclasses.
2. In the current version of OSATE, there is no class in the *Instance* model to represent *Subprograms* and *Subprogram Calls*. Information for these elements is only available from the *Declarative* model.
3. Another issue relates to *Annexes*. The core AADL language can be extended by embedding sub-languages such as the Behavioral Annex (BA) and the Error Model Annex (EMV2). Currently those are not represented in the *Instance* model although it is currently under development for EMV2.

6 Envisioned Utility of OSATE-DIM

OSATE-DIM has been developed keeping in mind the needs of both AADL-based tool developers and AADL-users. Support for various scenarios allows for integration of OSATE-DIM into a wide array of workflows for AADL-based research and development. For users, OSATE-DIM is envisioned to provide incremental model-synchronization capabilities, which ensures no loss and consistency of information. It also simplifies the modification of AADL models for users, who previously had to make changes in the *Declarative* models directly.

For the AADL-based tool developer, OSATE-DIM is useful for them by simplifying the development of their tool. Instead of having to design complex algorithms to modify the *Declarative* model at the correct location, the developer can simply implement algorithms to modify the *Instance* model. They can integrate OSATE-DIM within this modification (whether in-place our out-of-place) to automatically perform the synchronization with the quality of being the simplest changes having least loss of knowledge.

This especially will simplify the migration of tools to later versions/implementations of AADL.AADL targets significant changes of the *Declarative* metamodel on the lines of suggestion (1) in Section 5. If developers have integrated OSATE-DIM into their pipeline being therefore isolated from the *Declarative* model, they need not worry about such updates of the *Declarative* language, since OSATE-DIM after having been updated will take care of interfacing tools with *Declarative* models.

7 Related Work

Many tools apply the concept of views to MDE. EMF Views [11]uses an SQL-like DSL to define a virtualization engine. It looks at views as non-concrete entities, and implements them as virtualization of real base models so that there is no data duplication. Thus, changing data in the view implies change of the data in the base model.

OpenFlexo [12]is a tool for homogeneously handling and relating data from various sources. As soon as a view is computed, it is connected with different base models for synchronization. The synchronization is conceptually similar to EMF Views.

ModelJoin [13] is a tool for the creation of heterogeneous models. Its DSL is used to define not just the elements of the view, but also the meta-model of the view. Support for editability inside the views is provided using OCL constraints.

Orthographic Software Modelling (OSM) [14] is a hub-and-spoke architecture-based approach and tool that allows for the definition of multiple views from a Single Underlying Model (SUM). The definition is through a unique bidirectional transformation between the SUM and each view. Vitruvius [15] is based on the OSM approach but instead of a SUM, it uses a Virtual-SUM that is a non-invasive combination of many legacy metamodels. Flexible view definition allows restriction of possible view updates. Updating a view results in execution of corresponding synchronizing transformations.

These methods provide light-weight backward transformations, through virtualization using invertible transformations to define virtual views (EMF Views, OpenFlexo), or through constraints and restrictions on possible model edits (ModelJoin, OSM, Vitruvius). They are not as flexible and as specifically made for AADL as OSATE-DIM and often require to severely limit the class of possible updates to the view to guarantee well-behavedness.

8 Conclusion

In this article, we demonstrated a novel OSATE-based Declarative-Instance Mapping tool, OSATE-DIM, for incremental deinstantiation of AADL models. Deinstantiation is an important task for synchronization of information in the view-update paradigm. OSATE-DIM is a light-weight tool that accomplishes the task in various complex scenarios. An additional delta-trace meta-model is developed with OSATE-DIM, to track out-of-place model refinements. The tool, verified through two case studies, is available online.

References

[1] P. Feiler, D. Gluch, and J. Hudak, "The architecture analysis & design language (aadl): An introduction," Tech. Rep. CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2006.

[2] R. Medina, E. Borde, and L. Pautet, "Scheduling multi-periodic mixed-criticality dags on multi-core architectures," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, pp. 254–264, 2018.

[3] S. Rahmoun, A. Mehiaoui-Hamitou, E. Borde, L. Pautet, and E. Soubiran, "Multi-objective exploration of architectural designs by composition of model transformations," *Software & Systems Modeling*, vol. 18, 02 2019.

[4] Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, and F. Orejas, "From state- to delta-based bidirectional model transformations: The symmetric case," vol. 6981, pp. 304–318, 10 2011.

[5] H. Klare, M. E. Kramer, M. Langhammer, D. Werle, E. Burger, and R. Reussner, "Enabling consistency in view-based system development — the vitruvius approach," *Journal of Systems and Software*, vol. 171, p. 110815, 2021.

[6] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 869–891, 2012.

[7] A. Boydston, P. Feiler, S. Vestal, and B. Lewis, "Architecture Centric Virtual Integration Process (ACVIP): A Key Component of the DoD Digital Engineering Strategy," tech. rep., Software Engineering Institute, 2019.

[8] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd ed., 2009.

[9] R. Mittal, D. Blouin, A. Bhobe, and S. Bandyopadhyay, "Solving the instance model-view update problem in aadl," in *International Conference on Model Driven Engineering Languages and Systems 2022*, 2022.

[10] R. Mittal and D. Blouin, "Osate-dim solves the instance model-view update problem in aadl," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '22, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2022.

[11] H. Bruneliere, J. Garcia, M. Wimmer, and J. Cabot, "Emf views: A view mechanism for integrating heterogeneous models," vol. 9381, 10 2015.

[12] S. Guérin, J. Champeau, J.-C. Bach, A. Beugnard, F. Dagnat, and S. Martínez, "Multi-Level Modeling with Openflexo/FML," *Enterprise Modelling and Information Systems Architectures*, vol. 17, 2022.

[13] P. Langer, K. Wieland, M. Wimmer, and J. Cabot, "Emf profiles: A lightweight extension approach for emf models," *Journal of Object Technology*, vol. 11, p. 8, 04 2011.

[14] C. Atkinson, "Orthographic software modelling: A novel approach to view-based software engineering," in *Modelling Foundations and Applications* (T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, eds.), (Berlin, Heidelberg), pp. 1–1, Springer Berlin Heidelberg, 2010.

[15] H. Klare, M. E. Kramer, M. Langhammer, D. Werle, E. Burger, and R. Reussner, "Enabling consistency in view-based system development — the vitruvius approach," *Journal of Systems and Software*, vol. 171, p. 110815, 2021.

LAMP: to Shed Light on AADL Models

Pierre Dissaux

Ellidiss Technologies, 24 quai de la douane, 29200 Brest, Brittany, France ; Tel: +33 298 45 18 70; email: pierre.dissaux@ellidiss.com

Abstract

LAMP is an introspective analysis and processing framework for AADL. With LAMP, exploration, verification, transformation or any other processing rules are directly embedded inside the AADL model as annex subclauses. LAMP is based on the underlying LMP (Logic Model Processing) technology that itself leverages the Prolog language and a Prolog inference engine. This paper illustrates the use of LAMP by five practical use cases addressing some of the most important design issues in critical real-time software development.

Keywords: AADL, Prolog. LMP, LAMP

1 Introduction

LAMP [6] (Logic AADL Model Processing) is one of the model processing features that are provided by the AADL Inspector tool [7]. It addresses any kind of AADL [2] model processing, such as:

- AADL model exploration
- AADL model static analysis
- X to AADL and AADL to X model transformations

LAMP is an "online" model processing tool (the model and the processing functions are collocated), as opposed to LMP [4,5] (Logic Model Processing) that is a more conventional "offline" model processing solution where the processing functions are stored in a tool configuration area. However, many LMP features are reused by LAMP, and they are both based on a particular way to use of the Prolog language [1].

This paper provides a description of this LAMP framework and a selected list of practical use cases that are all supported by the current AADL Inspector distribution.

2 The LAMP framework

The LAMP framework is composed of a control panel (LAMP Lab) that is part of the AADL Inspector Graphical User Interface and a library of predefined rules (LAMP Lib) expressed in Prolog source code embedded inside AADL annexes in packages.

LAMP Lab allows for building Prolog fact bases on several sources (AADL, SysML, FACE, Capella, XML, CSV, simulation traces...) and specifying the processing goals to reach.

LAMP Lib contains accessors to the AADL model (declarative model, instance model, behavior annex, error annex), several predefined model transformations (SysML to AADL, FACE to AADL, Capella to AADL, AADL to HOOD), as well as an AADL printer (unparser), an end-to-end flow latency computation algorithm and a framework to implement cyber-security policies.

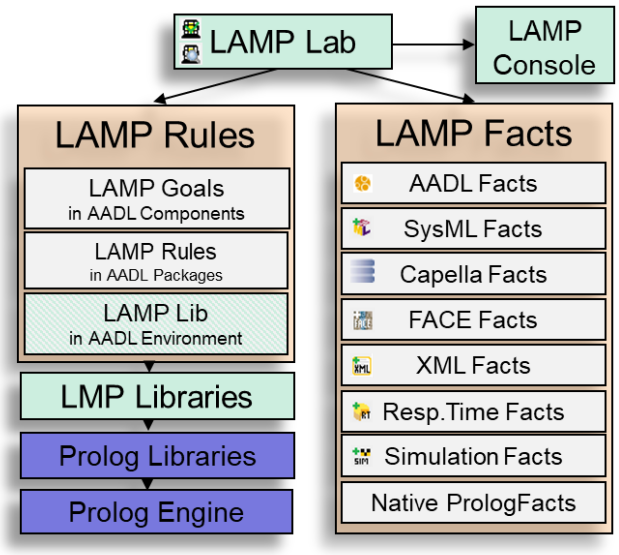

Figure 1 the LAMP framework

An end-user can interactively write his own processing rules within AADL Packages and combine them with the existing library.

As shown in Figure 1, the LAMP framework can also be represented by the association of a set of Prolog fact bases and rule bases.

The fact bases correspond to the various kinds of input data that can be processed, including of course the AADL model itself, that is composed of a list of packages. Other fact bases can be added and may come from any other source of information thanks to dedicated syntactic transformations into Prolog. This is in particular the case for Prolog representations of other models or tools results.

The rule bases include the predefined LAMP Lib as well as all the rules that are embedded inside the AADL applicative model packages. A special case concerns the high-level goal that is submitted as the query to the Prolog engine. By convention, these goals are located inside an AADL annex subclause associated with a component instead of a package.

3 LAMP use cases

The predefined rules provided by LAMP Lib enable quick and easy development of simple exploration and processing features by the end user. This chapter shows a few more advanced LAMP use cases that can also be adapted for other

similar usages, as all the corresponding Prolog source code is made available in LAMP Lib.

In this chapter, we address some of the most important design issues in critical real-time software development and show how LAMP can help solving them:

- Functional chain latency
- System to software model bridging
- Reverse engineering
- Cyber security analysis
- Composite assurance cases

3.1 Scheduling aware flow latency analysis

One of the main high level analysis goals for critical real-time systems is the ability to estimate the end-to-end latency of a global system functional chain. Such a functional chain typically involves a distributed architecture composed of sensors, software processing and actuators. All that can be properly described in AADL and the logical link between the ultimate source of information (usually a sensor) and the ultimate destination of the same information (usually an actuator) can be described by AADL flows.

Moreover, the AADL architecture can carry all the details that are required to perform precise timing analysis, and in particular to evaluate the response time of each concurrent thread and bus message.

These two individual analyses of the same AADL architecture, i.e., end-to-end flow analysis and response time analysis can be efficiently combined to offer a more precise estimate of function chains latency. We named this approach SAFLA (Scheduling Aware Flow Latency Analysis) and implemented it as a LAMP service.

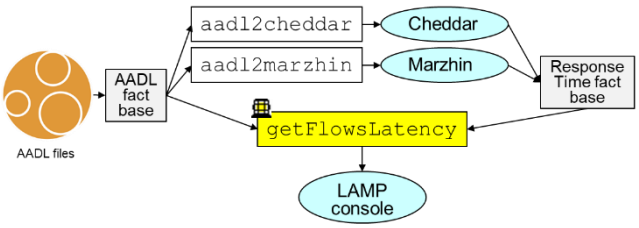

Figure 2 Scheduling Aware Flow Latency Analysis

Running SAFLA with LAMP Lab consists in automatically executing the following sequence of actions:

- Launching one of the timing analysis tools that are available with AADL Inspector (Cheddar or Marzhin)
- Computing the response time of each thread and bus message and putting these data into a separate Prolog facts base.
- Performing end to end flow analysis to identify all the contributors (threads and bus messages) of each functional chain.
- Summing up the response time of all the contributors to provide an estimate of the end-to-end flow latency.

Because it considers thread interferences and the behaviour of the real-time scheduler, SAFLA estimates are more accurate than other end-to-end flow latency analysis techniques that are based on static thread properties such as its period, deadline or user defined individual latency.

3.2 Model transformations into AADL

Another important goal while integrating system wide Model Driven Engineering solutions is the ability to implement model to model transformations. In this area, the realization of an efficient path between the system engineering and the software engineering activities is a frequent request of end users and tool-chain integrators.

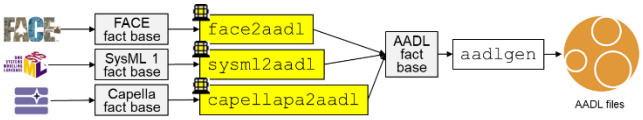

Figure 3 Model transformations into AADL

Let us consider, as a representative example, the transformation between a SysML (v1) global system model and an AADL model representing its software sub-systems. The aim of such a transformation is to ensure that all the useful information already expressed at system level is preserved while going down into the details of software design and analysis.

There are several well documented approaches to address this kind of needs and many technical solutions to implement them.

One of the common approaches consists in overloading the source system model by software details so that the transformation into the destination software model becomes easier. With SysML v1, this is typically realized by defining an AADL profile and associating entities at system level with AADL stereotypes. This approach has two main drawbacks:

- It overloads the system model with data that cannot be processed during system engineering activities.
- It requires an adaptation of the SysML design tools to support the AADL profile.

With LAMP Lab, the approach is different and does not have these drawbacks. It relies on a two steps process. The first step is purely syntactic and converts the source system model into a dedicated Prolog fact base. The second step consists in implementing the model transformation semantics with a Prolog rule base. An initial version of these rules is provided in LAMP Lib as a template. The LAMP approach has the following advantages:

- It preserves a clean separation of the source and destination models.
- It does not require a configuration of the upstream engineering tools.
- The transformation rules can easily be customized to implement a specific conceptual mapping between the two models.
- The same transformation rules can be shared for all input models, and then stored inside the common LAMP Lib, or defined locally within the destination project to address more specific mappings.
- The transformation rules can easily apply to a fusion of source fact bases originating from different modelling languages.

3.3 Textual AADL reverse engineering

The preceding section deals about model transformations ending to the creation of AADL models. Reverse path is also possible, i.e., generating a new kind of model from a source AADL project. An example of that is converting a set of AADL textual files to feed a graphical design tool for reverse engineering purposes.

The primary representation of the AADL language is its textual syntax. A graphical notation is also specified by the standard, and it can be helpful for describing the software architecture in a more visual way. However, this graphical notation may be used in different ways. It may simply mirror the architectural part of the textual representation with graphical artefacts (declarative model) or provide a view of the hierarchical decomposition of the top-level system (instance model). Note that a third possible graphical representation will consider the allocation of software instances onto hardware instances by graphical containment (deployment model).

The AADL variant of the Stood design tool (Stood for AADL) supports graphical edition of both the declarative and the instance models. When associated with the HOOD software design methodology, the instance model approach is required for the main "system to design" and its environment, whereas the declarative model option can be used to describe libraries of reusable components. AADL textual notation can be automatically generated by Stood from the combination of declarative and instance graphical models representing a complete HOOD design. Corresponding AADL files can then be loaded into any compliant AADL text editor such as AADL Inspector to perform static, timing, safety, and security analysis.

Although Stood offers its own AADL import feature, a more flexible reverse path has been implemented within AADL Inspector with LAMP. It consists of doing a model transformation from the original AADL project to a HOOD Standard Interchange Format (SIF) file that represents the AADL instance model.

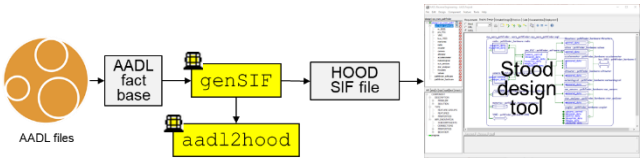

Figure 2 Textual AADL reverse engineering

This transformation is implemented by two LAMP modules. The first one, `genSIF`, provides the entry point for the reverse engineering operation and the production rules for the SIF output file. The second module, `aadl2hood`, implements the mapping between an AADL instance model and the HOOD entities and relationships.

These rules are available in LAMP Lib and can thus be easily customized if needed. Moreover, they can be used as a template to implement other reverse engineering approaches and target different design tools.

3.4 Security policy checker

Cyber security has become a very important concern for embedded real time critical systems. The level of confidentiality, integrity and availability of these system must thus be evaluated as early as possible during the development process to reduce the threats of unauthorized disclosure, modification, or loss of data when the software becomes operational.

Ideally, cyber security criteria should be addressed "by construction" thanks to a well-defined modular architecture of the "system to design" associated with suitable visibility rules. The "data hiding" and "low coupling" principles supported by the HOOD methodology can be a way to reach this goal. However, it is also recommended to perform dedicated early analysis and verification activities to check that the applicable security policy is respected.

As opposed to safety engineering where the verification processes have been well formalized and standardized for decades, security engineering approaches remains much more specific to limited scopes, at corporate, project, or sometimes team level. The online, flexible, and rigorous solution offered by LAMP is thus very beneficial to adapt the security verification process to each practical context.

Although significant work has been done on that topic, the AADL standard does not include a published security annex to extend the language specification for cyber security analysis purposes. Moreover, no general recommendation about the AADL security verification process is proposed by the standard. All that must thus be done at a tool, project, or user level. With LAMP, we have the ability to manage that at a project level, i.e., to embark the security verification rules inside the AADL model itself.

Concretely, supporting cyber-security analysis features with AADL requires the three following preparatory actions that are illustrated below with fragments of a simple example.

Define a security model, typically by specifying a dedicated property set to add security related information to AADL artefacts. Such additional information will thus become directly available as LAMP facts.

```
property set Lamp_Security_Model is
  Security_Level : aadlinteger 1 .. 5
    applies to (Data);
end Lamp_Security_Model;
```

Formalize a security policy that is appropriate for the current project and its use cases. Such a policy can take the form of a list of verification goals that may need to be reached according to more or less complex logics. LAMP goals fit well this need.

```
checkSecurityRules :-
/* Sec_R1 */ checkFlowSecurity,
/* Sec_R2 */ checkMaxSecurityLevel,
/* Sec_R3 */ checkNoWriteDown.
```

Implement each individual security policy rules associated with the verification goals. LAMP rules are a good solution to achieve that.

```
/* Sec_R3: When two components are */
/* connected via a shared Bus,     */
/* they must comply with the       */
/* No-Write-Down rule.             */
checkNoWriteDown :-
  isAADLBusBinding(C),
  isAADLConnection(P,T,I,C),
  getConnectionEnds(P,T,I,C,Xs,Xd),
  getMaxSecurityLevel(Xs,Ls),
  getMaxSecurityLevel(Xd,Ld),
  Ls > Ld,
 printMessageSec_R3(C,Ls,Ld).
```

With this solution, the security model, security policy and security rules can be fully customized to fit each project requirements. The rules can either be stored inside LAMP Lib for a higher reusability or kept within the AADL model for a higher confidentiality.

3.5 Composite assurance cases

In the previous sections, we have presented four individual model processing use cases that take benefit of the LAMP technology. Compared to other frameworks, LAMP inherits the advantages of the Prolog language, and especially its capacity to provide at the same time powerful model agnostic data representations (fact bases) providing implicit accessors and iterators, as well as rigorous logic programming features (rule bases).

In addition to these characteristics that are common to all usages of Prolog, LAMP leverages its deep integration with the AADL ecosystem. The exhaustive representation of an AADL project, including annexes, under the form of a fact base can be enriched by other sources of information also handled as additional mergeable fact bases. These complementary facts may come from any other phases of the development process (requirements engineering, system engineering, coding, testing…), or represent results previously obtained by other modelling, analysis, or verification tools.

All that makes LAMP a good candidate to formalize complex composite assurance cases that may need to be put in place to comply with high demanding V&V processes.

As opposed to the preceding use cases that are based on pre-existing features offered by LAMP Lib, such assurance cases implementation requires that new sets of rules are added by the development team.

4 Related works

A few dedicated languages like REAL [8] or RESOLUTE [9] have been developed to offer advanced online processing solutions for AADL.

The main difference with LAMP is that they are not leveraging an existing standard solution like Prolog. They thus require the specification of a complete language syntax, semantics and run-time environment, and its maintenance over the time.

5 Conclusion

This paper shows how the LAMP framework leverages the Prolog language to propose a powerful and flexible solution to implement online model exploration and processing features. LAMP is currently strongly focused on the realization of AADL related analysis and verification activities and is implemented as a part of the AADL Inspector tool.

However, its general principles (Prolog rules embedded inside the model to be processed) could be applied to any other modelling language and especially those supporting a textual notation such as SysML v2 [3].

References

[1] David S. Warren and all (2023), *Prolog: The Next 50 Years*, LNAI 13900, Springer

[2] SAE International (2022), *Architecture Analysis & Design Language (AADL)*, https://www.sae.org/standards/content/as5506d/.

[3] OMG (2023), *System Modeling Language™ v2*, *https://github.com/Systems-Modeling/SysML-v2-Release.*

[4] P. Dissaux and P. Farail (2014), *Model Verification: Return of Experience*, 7th European Congress on Embedded Real Time Software and Systems (ERTS).

[5] P. Dissaux and B. Hall (2016), *Merging and Processing Heterogeneous Models*, 8th European Congress on Embedded Real Time Software and Systems (ERTS).

[6] P. Dissaux (2020), *LAMP: A new model processing language for AADL*, 10th European Congress on Embedded Real Time Software and Systems (ERTS).

[7] AADL Inspector: *https://www.ellidiss.fr/public/wiki/inspector*

[8] Gilles O., Hugues J (2010), *Expressing and Enforcing User-Defined Constraints of AADL Models*, ICECCS 2010, pp 337-342.

[9] A. Gacek and all (2014), *Resolute: an assurance case language for architecture models*, HILT.

Challenges in Model Synchronization for Information Preservation Illustrated with the FACE and AADL Standards

Dominique Blouin, Anish Bhobe, Laurent Pautet
LTCI Lab, Télécom Paris, Institut Polytechnique de Paris, France; email: <firstname>.<lastname>@telecom-paris.fr

Abstract

This article explores the challenges in Model Synchronization in Model Driven Engineering, focusing on Information Preservation. It introduces modern architecture description languages such as FACE and AADL and describes the challenges in reliably maintaining consistency of their models, as well as determining how to propagate changes of one model to the other. It describes ongoing efforts including improving Information Preservation in model synchronization approaches such as Triple Graph Grammars, as well as the required foundations and ideas for the establishing effective Model Management and Synchronization frameworks.

Keywords: Model Synchronization, Model Management, Model Driven Engineering, AADL, FACE

1 Introduction

Modern engineered systems are increasingly complex and expensive to develop and this is further exacerbated in the case of safety-critical systems where certification and reliability of systems is paramount. Using conventional waterfall or V-cycle development methods lead to errors being discovered late during the testing and integration phase. Significant time and financial investment is wasted to fix the issues in an already implemented system.

This has increased the adoption of Model Driven Engineering (MDE) in development of critical systems. In MDE process, the developers build Models i.e. abstractions of the system, which are used for validation and generation of the software code. The various components and subsystems are modeled using different models which are brought together through the process of Virtual Integration during the end of the design phase. This allows integration testing early in the development process, highlighting any flaws in design before any effort is committed to it.

The models are described in Domain Specific Modeling Languages (DSML) that can capture the different but complementary aspectsof a system. Thus we have to use multiple DSMLs to describe all the aspects of the entire system. However, there is a significant overlap in information captured by the many aspects of the system. When a model is edited, any changes in the information in the overlapping part must be propagated to the other models.

A well known error caused by the lack of synchronization was in the development of the Airbus A380 wherein the CAD software used between the teams in France (CATIA 5) and in Germany (CATIA 4) had differences in calculation of cable lengths. As such, while each team independently could create models that passed validation, the final aircraft had cables that were shorter that required by the airframe and thus cost an additional estimated 4.8 billion euros in development [1].

Model Management approaches attempt to solve this problem by applying changes and validation globally to the set of models instead of locally to each model, and synchronize changes across models to ensure consistency (Model Synchronization). They also can provide virtual integration capability to verify and validate the system as a whole, with all models working together similar to the integration step within conventional software development. These processes are generally implemented as a set of Model Transformations (MT) which contain the rules to generate target (unmodified) models from source (modified) models. However, not all the information in the target model is available to the source and thus may be erased by the model transformation. As such, ensuring that the information is preserved during the MT process is important for the reliability of the MDE process.

Model Management and Model Synchronization approaches are increasingly attempting to reduce the information loss by using methods such as Incremental Model Transformation [2] as well as newer Generalized Concurrent Rules [3] to reduce the information lost. However, there is no way of comparing the methods and their preserved information as no metric for analysing Information Preservation exists.

2 Motivation

The increasing use of MDE in aviation, automobiles and other safety-critical fields puts increasing demand on improvements in the space of Model Synchronization and Information Preservation. One such challenging example is the case of synchronization of models of the Architecture Analysis and Design Language (AADL)[1] and the Future Airborne Capabilities Environment (FACE)[2].

[1] `https://www.sae.org/standards/content/as5506d/`
[2] `https://www.opengroup.org/face`

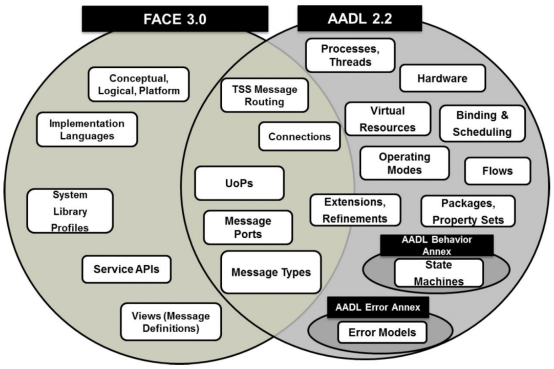

Figure 1: FACE and AADL information overlap [6]

2.1 FACE

FACE is a technical standard for the reference architecture for portable software components. It is an Architecture Description Language (ADL) that specifies different Units of Conformance (UoC) or Units of Portability (UoP) and the data that they exchange, as well as rich set of software data structures. However, it lacks language constructs to define component behaviors or non-functional properties such as timing, resource consumption, etc.

2.2 AADL

AADL is an SAE international Aerospace Standard (AS). It is an ADL that supports performance and safety analyses and code generation (synthesis) on descriptions of computer hardware, software representations, and hierarchical system compositions. It allows specific description of threading, processes, scheduling etc. of systems.

2.3 FACE and AADL

FACE describes the various Units of Conformances and Units of Portability and the data exchanged between them, while AADL can use these UoCs and UoPs to model the entire Cyber-Physical System with the hardware and operational properties such as behaviors, threading, task timings, etc. This allows analysis tools to detect problems of the integrated system such as memory over-usage, overrun task deadlines, unhandled error propagation etc. from AADL models of systems, while FACE focuses on reusability and modularity, software requirement conformance, and the interoperation of system or components through shared protocols.

Information such as the message types, the details on portability and connections, message routing etc. as shown in fig. 1. Any change in a FACE model that changes these properties will invalidate the corresponding AADL model. Thus, such changes must be propagated to the generated AADL model and vice versa in order to keep the models consistent. Such propagation is described by Model Transformations that provide rules defining how the synchronization must take place.

To this end, there is a need to find the best Model Transformation (MT) approaches for implementing such synchronization based on computing performance, as well as Information Preservation capability. There is work on benchmarking and comparing of MT approaches [4] [5] but no such benchmark or metric for analyzing information preservation exists.

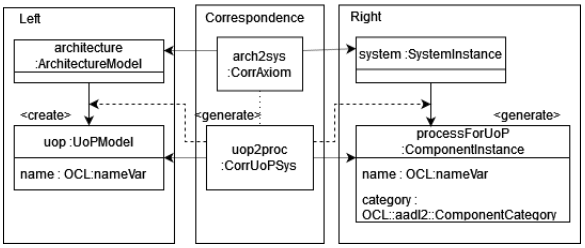

Figure 2: A Triple Graph Grammar rule for mapping a FACE unit of portability with an AADL process

3 Background

3.1 Model Transformations

Model Transformations (MT) can translate a model to another model in the same or a different DSML. A MT generally define these as a set of transformation rules or procedures. MTs that create entire target models from the source models at every change are known as Batch Model Transformations, while MTs that can only update the set of inconsistent model elements by applying only the relevant subset of rules are known as Incremental Model Transformations.

3.2 Triple Graph Grammar

Triple Graph Grammars (TGG) [7] [8] are a well formalized method for bi-directional incremental Model Transformations. A TGG is a set of 3 Graph Grammars, each of which describes how a graph can be constructed by a set of transformation rules. All models in a TGG are interpreted as graphs. A TGG describes the construction of 3 models simultaneously: the Left Model (e.g. FACE), the Right Model (e.g. AADL) and a Trace Model that links the elements that are constructed together. A TGG thus contains rules that define how a source model is edited, which rules to correspond to target model and which rules to apply on the target model. Either Left or Right model may be a source. As such, the TGG rules can construct the entirety of the Left Model from the Right Model, or vice versa, as well as only synchronize updates by applying only a subset of rules. For illustration, fig 2 shows simple left and right graphs being connected via a Correspondence Graph in the center. Within such a set of models, in case the *uop* element is created in the left graph (as marked by <*create*>) then, the set of provided rules can automatically generate (marked by <*generate*>) the *processForUoP* element in the right graph, and also add the *uop2proc* correspondence element between the two graphs. *uop* only corresponds to *processForUoP* if their parents i.e. *architecture* and *system* correspond to each other. As such, we can say that in the correspondence model, *uop2proc* depends on *arch2sys*

While this method is well formalized and robust, it often applies rules redundantly. For example, consider a case where a new node is added as a root in the source model. TGG will revoke all the relations, create a root node in the target model and then apply the required rules to create the sub-tree of the target model from the source model, instead of re-using the existing model elements. Any time the types of an element or its hierarchy is changed, the correspondence relations are revoked and the tree has to be recreated.

4 Challenges

4.1 Information Preservation (IP)

There has been work on improving both the computational performance as well as IP by using Incremental Model Transformations which preserve information by simply preserving the target model's unedited elements. However, there may still be information loss in the edited sections. Examples of incremental MT tools are VIATRA[3], YAMTL[4], MoTE (TGG)[5] and eMoflon (TGG)[6].

A well known case in information loss in TGGs occurs for fields in target model's modified elements that were not captured by the source model. Whenever a delete and re-create takes place, for example in fig. 2, if *architecture* is re-parented. The *system* will get re-parented and *processForUoP* will be deleted and re-created. During the delete the *category* field will be deleted and the information in it will be lost.

4.2 Non-Bijective Relations

Languages like AADL use concepts of Inheritance and Refinement commonly seen within Object-Oriented Programming. These concepts allow multiple AADL components to derive from the same base AADL system properties. As processes and other components in AADL are linked to elements from FACE such as UoC, any addition of elements or structural changes in the FACE model need to be propagated to the AADL model. However, this gives rise to the question of where in the inheritance hierarchy should the change take place, as a modified element may require a modification of the base class, or a refinement of the same. Mittal et al [9] tackle this problem specifically for AADL instancing transformation.

4.3 Recovery from Inconsistent States

AADL and FACE ecosystems also contain specialized editors and tools that help develop these models. At certain times, concurrent accesses or modification to both models being synchronized can leave the system in an inconsistent state. As such, the synchronization process needs to identify the information from the two models that is to be preserved in preference to the other. Virtual Integration processes such as System Architecture Virtual Integration (SAVI) [10] define and require a Single Source of Truth (SSoT) that contains the information that is considered correct or has authority over other sources of information. These approaches generally use models that can be demarcated into SSoT and dependent models. But as AADL and FACE capture different aspects better, they must be considered as both Sources of Truth. Thus we cannot necessarily define a single source of truth, especially in a synchronization process where the two models can be considered to be separate but equally important.

We can often identify cases of information loss and correct it and find specific solutions for particular uses. However, the lack of any formal metric on the amount and type of information preserved poses significant challenge in comparing approaches with each other to find the most appropriate method.

The following research questions are of great importance:

1. How do we measure Information Preservation in Model Transformations?
2. Under what circumstance may information be lost in each approach?
3. What is the appropriate location for an information?
4. Which information in a conflict is correct?

5 State of work on reducing information loss in TGG

To tackle the problem of information loss during the process of synchronization due to the delete and create operations, Kosiol et al. [3] propose the inclusion of *generalized concurrent rules* (GCR). The GCR use Negative Application Conditions (NACs) to recognize elements that will be deleted and re-created and instead preserves and re-uses such elements to avoid the loss of uncaptured fields. The work further provides proofs that for TGGs that have completely specified NACs and consist of monotonic graph transformations that are covered by the approach, information loss of such kind is prevented.

However, this adds additional constraints on the developers to add the NACs as well as to ensure monotonic transformations. Monotonic transformations involve rules that square such as *architecture* and *arch2sys* in fig 2 form with *uop* and *uop2proc* i.e. for each pair of rules (f, g) that transform a model M into M^1 and M^2, there exists a pair (f', g') that transforms M^1 and M^2 into the same model M'. As such, the Information Preservation is limited to a subset of the set of possible TGGs and also requires additional work to specify the NACs.

6 Perspective

There is a need for identifying the cases and causes of Information loss, as well as developing generalized frameworks that can patch the limitations of the approaches, or at least recognize and warn the users in case a change would lead to Information Loss.

6.1 Information Preservation Metric

Information Preservation in a primitive model element is non-probabilistic - it is either preserved, lost, or imprecised but is not corrupted in the MT process so long as the MT approach and the rules are correctly implemented to match the required synchronization. Using this we analyze patterns or cases where information is always preserved, deliberately deleted or sometimes lost. Such a metric will be useful to evaluate the previous body of work in Information preservation, and also provide a scaffolding for the future work.

6.2 Architecture Centric Model Management (ACMoM)

ACMoM[7] is a framework built to support the Architecture Centric Virtual Integration Process (ACVIP) [11] The framework is based on Hierarchical Megamodels [12] and manages the relationships and consistency between different models. It considers the relations for model synchronization as

[3] https://projects.eclipse.org/projects/modeling.viatra

[4] https://yamtl.github.io/

[5] https://www.hpi.uni-potsdam.de/giese/public/mdelab/mdelab-projects/mote-a-tgg-based-model-transformation-engine/

[6] https://emoflon.org/

[7] https://mem4csd.telecom-paristech.fr/blog/index.php/acmom/

black-boxes and can use different Model Synchronization approaches for different relations. ACMoM also maintains a well typed understanding of the models that are managed and can thus analyse the set of models as well as the relationships at large scale. As such, it can act as a platform to analyse Information Preservation as well as being able to warn users or conditionally apply the transformations if required i.e. apply Change Policies as described in section 6.3.

6.3 Model Change Policies

Completely solving the problem of Information Preservation in Model Synchronization approaches is difficult and may not be possible or scalable for all the approaches. However, Information Loss can be identified during the use of a tool and the cases can be noted. In such cases, even if an approach is lossy, an external intervention can allow preserving or restoring information. As such, Model Management methods can analyse the used models and transformations to recognize loss cases automatically, or based on heuristics created by the users and apply specific strategies to counter the issues. The strategies can be based on:

- Using different approaches based on the IP metrics or other criteria.
- Adding additional conditional rules to store and preserve information that is known to be lost.
- Disallowing/Replacing changes on the input that lose information in favor of other changes that have the same effect but better preserve information.
- Warning the user of the information loss for manual preservation.

6.4 Authoritative Source of Truth

To solve the problem of Single Source of Truth noted in section 4.3, there is a notion of Authoritative Source of Truth (ASoT)[8] which defines which model(s) is the Source of Truth for the given information. As such, a model management framework can use the ASoT to decide the transformation to use, and which information is dropped and updated in favor of the ASoT. The ASoT must be organized on the basis of the technological and conceptual domains instead of the various models or programs in use. Such ASoTs can also include requirement models, engineering costs and other data, and can be leveraged by the model synchronization as well as the validation involved.

7 Conclusion

In summary, the increasing complexity of modern safety-critical systems requires MDE for early design verification and validation and development. The use of multiple models gives rise to the challenge of model synchronization, especially in cases such as AADL and FACE in the aviation domain. We look at TGGs as a Model Synchronization approach as well as the effort made to improve Information Preservation with methods such as Generalized Concurrent Rules (GCR). However, a crucial need exists for a metric to quantify and identify information loss. Ongoing and future work includes development of Model Change Policies to automate identification and/or mitigation of information loss.

[8] https://www.omgwiki.org/MBSE/doku.php?id=mbse:authoritative_source_of_truth

References

[1] Admin, "Why do projects fail?." "https://calleam.com/WTPF/?p=4700", Jan 1970.

[2] D. Hearnden, M. Lawley, and K. Raymond, "Incremental model transformation for the evolution of model-driven systems," in *Model Driven Engineering Languages and Systems: 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006. Proceedings 9*, pp. 321–335, Springer, 2006.

[3] J. Kosiol, "Formal foundations for information-preserving model synchronization processes based on triple graph grammars," 2022.

[4] H. Mkaouar, D. Blouin, and E. Borde, "A benchmark of incremental model transformation tools based on an industrial case study with aadl," *Software and Systems Modeling*, vol. 22, no. 1, pp. 175–201, 2023.

[5] E. Leblebici, A. Anjorin, A. Schürr, S. Hildebrandt, J. Rieke, and J. Greenyer, "A comparison of incremental triple graph grammar tools," 01 2014.

[6] A. Labs, "Introduction to aadl analysis and modeling with face units of conformance." "https://camet-library.com/sites/default/files/documents/2018.08.28_Introduction_to_AADL_with_FACE.pdf", 2018.

[7] A. Schürr, "Specification of graph translators with triple graph grammars," in *Graph-Theoretic Concepts in Computer Science* (E. W. Mayr, G. Schmidt, and G. Tinhofer, eds.), (Berlin, Heidelberg), pp. 151–163, Springer Berlin Heidelberg, 1995.

[8] H. Giese and R. Wagner, "From model transformation to incremental bidirectional model synchronization," *Software & Systems Modeling*, vol. 8, pp. 21–43, 2009.

[9] R. Mittal, D. Blouin, A. Bhobe, and S. Bandyopadhyay, "Solving the instance model-view update problem in aadl," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, (New York, NY, USA), p. 55–65, Association for Computing Machinery, 2022.

[10] P. Feiler, J. Hansson, D. de Niz, and L. Wrage, "System architecture virtual integration: An industrial case study," Tech. Rep. CMU/SEI-2009-TR-017, Nov 2009. Accessed: 2023-Dec-8.

[11] A. Boydston and P. H. Feiler, "Architecture centric virtual integration process (acvip) : A key component of the dod digital engineering strategy," 2019.

[12] A. Seibel, *Traceability and model management with executable and dynamic hierarchical megamodels*. PhD thesis, Hasso-Plattner-Institut fur Softwaresystemtechnik, Universität Potsdam, 2013.

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/

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