The journal for the international Ada community

Ada User Journal

Volume 45
Number 3
September 2024

Produced by Ada-Europe

Ada-Europe General Secretary

Dene Brown SysAda Limited Signal Business Center 2 Innotec Drive BT19 7PD Bangor Northern Ireland, UK	Tel: +44 2891 520 560 Email: Secretary@Ada-Europe.org URL: www.ada-europe.org

Information on Subscriptions and Advertisements

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

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

ADA USER JOURNAL

Volume 45
Number 3
September 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

As we welcome you to this September issue of the Ada User Journal, I would like to begin by extending my warmest congratulations to Luís Miguel Pinho, our Associate Editor, on his appointment as President of Ada-Europe. I would also like to express my appreciation for Tullio Vardanega, who, after serving as President of Ada-Europe for so many years, now takes on the leadership of the newly established Ada User Society. Both assume significant responsibilities in shaping the future of the Ada language, and I wish them every success in their new roles. Their dedication and expertise will undoubtedly contribute to ensuring the continuity and, hopefully, the growth of the Ada community.

Regarding the technical content, this issue features the Proceedings of the Ada Developers Workshop, which was co-located with the 28th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2024) last June in Barcelona. The proceedings open with an introductory note by the workshop organizers, Fernando Oleo Blanco and Dirk Craeynest, followed by nine concise articles summarizing the respective presentations at the workshop. These contributions explore various aspects of the Ada language, including language-specific features, related tools, and systems written in Ada, as well as an especially thought-provoking article reflecting on the present and future of Ada. We hope these proceedings will offer valuable insights and perhaps inspire participation in future editions for those who could not attend the workshop —or even for those who did—.

This issue also features a contributed article on the formal verification of safety-critical software written in Ada, authored by a team of researchers from the Vikram Sarabhai Space Centre, IIT Bombay, and IIT Madras in India. The article explores the use of the SPIN model checker, combined with a custom toolchain, to perform static analysis of Ada code and help ensuring software reliability in critical applications.

As always, we include our News Digest and Calendar sections, expertly curated by their respective editors, Alejandro R. Mosteo and Dirk Craeynest. Following the calendar, readers will find the Call for Papers for the 29th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2025), set to take place from June 10–13, 2025, in Paris, France. With multiple tracks and submission formats available, we encourage everyone to contribute and be part of this exciting event!

Antonio Casimiro
Lisboa
September 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,

Today, I want to highlight two conversations in the Digest that interested me particularly. Firstly, documentation about the 'Red' language has been found online, and it seems that this completes the availability of all contestants from which Ada emerged victorious [1]. Curiously, I found the style of the code not that unfamiliar for an Ada programmer.

Secondly, an animated discussion emerged around the idea of "what Jean Ichbiah would want to find in Ada 2022" [2]. Therein you can also find the reservations he had about preliminary versions of Ada 95 [3], which is in itself worth a read if you have not read them before (as I had not).

Sincerely,
Alejandro R. Mosteo.

[1] "'Red' and the DoD Language Competition", in Ada and Other Languages.

[2] "Ichbiah 2022 Compiler Mode", in Ada Practice

[3] https://web.elastic.org/~fche/mirrors/old-usenet/ada-with-null

Ada-related Events

[AEiC 2024] Ada Developers Workshop Videos and Slides

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Subject: [AEiC 2024] Ada Developers Workshop videos and slides are public
Date: Thu, 8 Aug 2024 20:49:18 +0200
Newsgroups: comp.lang.ada

Dear Ada community,

the recordings of the talks that were held in the Ada Developers Workshop have been made available in the AEiC 2024 website [1]. The slides for each presentation can also be found there. The links can be found just under the title for each entry.

Also, huge thanks to Dirk, Nam, Fabien, the organisers of the conference; Ada-Europe and AdaCore for their sponsorship and their funding to get the technology ready to record the Workshop.

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

Best regards,
Fer & the Ada Developers Workshop team

P.S: any kind of feedback is more than welcome!

Ada Monthly Meetup, September 2024

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Subject: Ada Monthly Meetup, September 2024
Date: Sun, 11 Aug 2024 17:21:09 +0200
Newsgroups: comp.lang.ada

I would like to announce the September (2024) Ada Monthly Meetup which will be taking place on the 7th of September at 13:00 UTC time (15:00 CEST). As always the meetup will take place over at Jitsi. The Meetup will also be livestreamed/recorded to Youtube.

If someone would like to propose a talk or a topic, feel free to do so! We currently have no proposals. Nonetheless, I would like to talk about the AEiC 2024 Ada Developers Workshop, remind people about the 2024 Crate of the Year Award and maybe talk a bit about the Ada Users Society :)

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

Ada Monthly Meetup, 5th October 2024

From: Fernando Oleo / Irvise <irvise_ml@irvise.xyz>
Subject: Ada Monthly Meetup, 5th October 2024
Date: Mon, 16 Sep 2024 22:43:14 +0200
Newsgroups: comp.lang.ada

I would like to announce the October (2024) Ada Monthly Meetup which will be taking place on the **5th of October at 13:00 UTC time (15:00 CEST).** As always the meetup will take place over at Jitsi. The Meetup will also be livestreamed/recorded to Youtube.

If someone would like to propose a talk or a topic, feel free to do so! We currently have no proposals. Nonetheless, I would like to bring some topics that were left off during September's Meetup.

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

Best regards and see you soon!
Fer

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

P.S: you can see the September summary in https://forum.ada-lang.io/t/ada-monthly-meeting-september-2024/1073/6 or in YouTube (with audio issues) https://www.youtube.com/live/i_bVoiDlw5E

Ada-related Resources

[Delta counts are from July 11th to November 13th. —arm]

Ada on Social Media

From: Alejandro R. Mosteo <amosteo@unizar.es>
Subject: Ada on Social Media
Date: 13 Nov 2024 19:35 CET[b]
To: Ada User Journal readership

Ada groups on various social media:

- Reddit: 8_868 (+127) members [1]
- LinkedIn: 3_549 (+28) members [2]
- Stack Overflow: 2_426 (+15) questions [3]
- Ada-lang.io: 287 (+46) users [4]
- Gitter: 271 (+13) people [5]
- Telegram: 208 (+3) users [6]
- Libera.Chat: 69 (-4) concurrent users [7]

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

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

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

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

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

[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: 13 Nov 2024 19:39 CET[c]
To: Ada User Journal readership

GitHub: >1_000* (+260) developers	[1]
Rosetta Code: 1_005 (+26) examples	[2]
42 (=) developers	[3]
Alire: 483 (+71) crates	[4]
1_268 (+200) releases	[5]
Sourceforge: 251 (-1) projects	[6]
Open Hub: 214 (=) projects	[7]
Codelabs: 60 (+3) repositories	[8]
Bitbucket: 37 (=) 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: 13 Nov 2024 19:48 CET
To: Ada User Journal readership

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

- TIOBE Index: 25 (-1) 0.71% (-0.08%) [1]
- PYPL Index: 15 (+2) 1.03% (+0.07%) [2]
- Languish Trends: 153 (+39) 0.01% (+0.01)% [3]
- Stack Overflow Survey: 40 (+2) 0.9% (+0.13%) [4]
- IEEE Spectrum (general): 50 (-14) Score: 0.0014 0107 (-0.093) [5]
- IEEE Spectrum (jobs): 55 (-26) Score: 0.0 (-0.0173) [5]
- IEEE Spectrum (trending): 46 (-16) Score: 0.0022 (0.01) [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/2024/

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

Re: Ada-Lang and Its Forum

From: Randy Brukardt <randy@rrsoftware.com>
Subject: Re: Ada-Lang and it's (more active than CLA) forum
Date: Tue, 2 Jul 2024 02:55:49 -0500
Newsgroups: comp.lang.ada

[Cont'd from AUJ 45-2, April 2024 —arm]

> Ada-Lang is a community maintained and supported webpage whose intent is to give a nice "landing page" to anybody wanting to learn Ada and become a hub for all Ada users.

I was adding this site to AdaIC's "Learn" pages (I think it disappeared some years ago, it is good to see it back), and noted that nowhere does it identify itself as "Ada-Lang" or any other short name on the site itself. It just calls itself "Ada Programming Language", which is a bit grandiose (there are a number of sites that can lay claim to part of that title, but surely none that can lay claim to all of it). Within the Ada Community in particular, it helps to identify the site more precisely. And I don't think that many people really look at the links that they click on, I doubt many people using AdaIC do, so just using the domain name and assuming people know what it is without any identification elsewhere is not ideal.

My two cents worth. (Humm, given prices these days, I don't think you can actually buy anything with two cents. That's probably one cliche that needs updating. ;-)

AWS-friendly Web Hosting

From: Marius Alves <marius2023pt@gmail.com>
Subject: Ada/GNAT/AWS-friendly web hosting
Date: Thu, 12 Sep 2024 15:25:41 +0100
Newsgroups: comp.lang.ada

Researching how to build an HTTP server (serving a website) on a local machine (MacOS) using AWS (Ada Web Server) and deploy it on a web hosting provider (e.g. 1dollar-webhosting.com).

Anyone done that? I've searched but could not find [anything].

Thanks.

Some specific questions on my mind follow.

Is a macOS host required (e.g. Ultahost 15 euros/month; I'd rather stay with 1dollar)?

If the host runs on Linux then cross-building (from macOS to Linux) is required, right? GNAT does that, right?

Or, must the program be built in the host? (Thus requiring GNAT to be there.)

The host is already running an HTTP server program (probably Apache). Must it be turned off? How?

In general, can the executable be launched on a VPS (Virtual Private Server)? Which port?

Will dynamic linking work? I'm guessing not, so, static; but then, will GNAT integrate the right libraries for Linux in the executable?

Will "Community GNAT" do? (Instead of GNAT Pro.)

Are those the right questions?

Thanks, thanks, thanks, thanks, thanks, thanks and thanks.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 12 Sep 2024 16:48:40 +0200

Adalog's site (https://www.adalog.fr/) is a standalone program written in Ada with AWS.

So are the sites for the various Ada-Europe conferences (see https://www.ada-europe.org/conference2024/ for example).

And many others...

> Is a macOS host required

No

> If the host runs on Linux then cross-building (from macOS to Linux) required, right?

Never tried, but no reason it shouldn't be possible

> Or, must the program be built in the host?

That's what I do

> The host is already running an HTTP server program (probably Apache). Must it be turned off? How?

Of course, you cannot have two programs listening on the same port, so if you want to listen to 80 or 8080, you'd better stop Apache (or any other program) to do that. As for me, I don't run Apache at all.

> In general, can the executable be launched on a VPS (Virtual Private Server)? Which port?

The port is given by the initial data of AWS

> Will dynamic linking work?

You just compile your program like any other Ada program

> Will "Community GNAT" do? (Instead of GNAT Pro.)

Yes, that's what I do

> Are those the right questions?

All questions are right....

> Thanks, thanks, thanks, thanks, thanks, thanks and thanks.

You're welcome

From: Drpi <314@drpi.fr>
Date: Thu, 12 Sep 2024 16:54:45 +0200

> The host is already running an HTTP server program (probably Apache). Must it be turned off? How?

The usual way is to use Apache (or nginx or another one) as a front end. Your application uses port 1080 (or something else) and the front end relays this port to the external 80 port.

This way, the security stuff is managed by the front end, not your application. You can also run multiple applications, each being redirected to its domain name/path.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 12 Sep 2024 18:22:28 +0200

> Researching how to build an HTTP server (serving a website) on a local machine (MacOS) using AWS (Ada Web Server) and deploy it on a web hosting provider (e.g. 1dollar-webhosting.com).

In my experience, this would be easier done with Gnoga (https://sourceforge.net/projects/gnoga/) than AWS. On a web-based system using AWS quite a while ago, we had to have a number of JS files. Although we had a lot more Ada than JS, we spent a lot more effort correcting JS errors than Ada errors.

Gautier de Montmollin has made Gnoga programs publicly available, such as his Pasta! game (http://pasta.phyrama.com/), so might be able to help with your hosting questions.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 12 Sep 2024 19:06:08 +0200

> This way, the security stuff is managed by the front end

But security breaches mainly use known bugs in Apache... If you write your own server with AWS, the attacker knows nothing about the software that answers! And as for buffer overflow attacks... Well, it's Ada. You'll see some handled Constraint_Error in the log file, end of story!

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Thu, 12 Sep 2024 17:16:29 -0000

> But security breaches mainly use known bugs in Apache… [...]

AWS uses OpenSSL or a fair bit better LibreSSL for TLS, written in C and quite often found vulnerable. You could isolate the nginx proxy to another machine though.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 12 Sep 2024 20:48:29 +0200

> Researching how to build an HTTP server (serving a website) on a local machine (MacOS) using AWS (Ada Web Server) and deploy it on a web hosting provider (e.g. 1dollar-webhosting.com).

That depends on what the provider would allow you to upload to the host. Likely nothing executable... (:-))

> If the host runs on Linux then cross-building (from macOS to Linux) is required, right?

It is possible, but far simpler would be a virtual machine running Linux. E.g. I compile for Linux targets on virtual machines. Only for ARM I am using physical machines. You must know what kind of Linux your provider has in order to choose the right version of the libc etc. [...]

> Will dynamic linking work?

If you ship the libraries together with the server. Then if the host runs Apache it must have some TLS library installed. You must learn the version and link against it. In any case you need either OpenSSL or else GNUTLS. The HTTP server from Simple Components can use both. I believe that either can be built as a static library. I see no reason why AWS could not be linked statically. BTW you must maintain certificates on the server.

> Will "Community GNAT" do?

I am not sure if all-static builds were possible, e.g. libc, libgnat.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Thu, 12 Sep 2024 22:29:36 -0000

> we spent a lot more effort correcting JS errors than Ada errors.

Did you “use strict”?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 13 Sep 2024 11:03:03 +0200

> Did you “use strict”?

I don't know. It was quite a while ago and I didn't work on the JS. But the point is that when you use Gnoga, you don't need any to create any JS.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Thu, 12 Sep 2024 22:35:20 -0000

> But security breaches mainly use known bugs in Apache…

That’s called “security through obscurity”. Not recommended.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Thu, 12 Sep 2024 22:40:35 -0000

> The usual way is to use Apache (or nginx or another one) as a front end.

Yup, I do things this way for my Python+ASGI code, too. This is called a “reverse proxy”, though I don’t know why -- I think “server-side proxy” would be more accurate.

Make sure your back-end server is listening only on a loopback address: 127.0.0.0/8 (IPv4) or ::1 (IPv6). That way the only access to it from outside the machine is through the public web-server front end.

(Question to ponder: why does Ipv4 offer over 16 million different loopback addresses, while IPv6, with its much larger address space, has to make do with only one?)

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 13 Sep 2024 08:46:33 +0200

> That’s called “security through obscurity”. Not recommended.

No, AWS is public and there is nothing hidden. Just that, since there are wayyyyy more users of Apache than of AWS, attackers will not bother to try to break in

From: Stéphane Rivière <stef@genesix.org>
Date: Fri, 13 Sep 2024 15:15:03 +0200

As a professional web hoster, I strongly advise you to forget Apache and use only Nginx, both as a proxy (in your case) and as a web server (generic case). Not only does Apache have security problems, but its performance is pitiful compared to Nginx.

If you have several sites, the ideal solution is to enter everything in https/port 443 on the nginx proxy (which will be able to manage X509/TLS https certificates) and exit on as many ports 8080, 8081, 8082, etc. as you have websites.

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Fri, 13 Sep 2024 16:33:15 +0200

> Researching how to build an HTTP server (serving a website) on a local machine (MacOS) using AWS (Ada Web Server) and deploy it on a web hosting provider (e.g. 1dollar-webhosting.com).

I don't know about 1dollar, but a typical web hosting provider will only let you upload static files (HTML, pictures et cetera), limited snippets of web server configuration, and certain kinds of programs that run under their web server's control. PHP is common. Some might run Perl programs with mod_perl, or Python programs using WSGI.

Maybe some web hosts support CGI or FastCGI. Those interfaces can be implemented in Ada. I think you'll have limited use for AWS in that case, as the HTTP parsing is handled by the web server.

I think it would be hard to find a web host that lets you run arbitrary network-facing daemons. To run your own web server you want a VPS (or a physical server in a colocation facility, but if your security needs don't rule out a web host, then a VPS is also fine).

> The host is already running an HTTP server program (probably Apache). Must it be turned off? How?

A typical web host won't let you turn off their web server. They serve many customers' content from the same Apache instance, so turning that off would break all those websites.

> In general, can the executable be launched on a VPS (Virtual Private Server)?

Sure. In a VPS you have the whole operating system to yourself (maybe except for the kernel if the VPS provider uses OpenVZ). You install and run whatever programs you want, just like on your own physical computer. Maybe you'll be able to get a VPS with macOS, if that's your preference.

In a VPS it's also your responsibility to install updates regularly, and upgrade to a new major OS version from time to time. If you fail to keep up, then criminals will take over your VPS and use it as a relay when attacking others. Make sure that you'll be notified automatically when there are updates to install.

> If the host runs on Linux then cross-building (from macOS to Linux) is required, right?

GCC – and thus GNAT – can be built as a cross-compiler. Perhaps you can find one that someone has built and packaged for MacOS. Otherwise you'll need to build your own from the GCC source code, configuring it to be a cross-compiler. (That's theoretical knowledge. I have no practical experience with cross-compilation).

> Or, must the program be built in the host? (Thus requiring GNAT to be there.)

No, but in my opinion it's much easier that way. Either build on the computer you'll run on, or on another computer of the same processor architecture, running the same version of the same operating system. That way you don't need to worry about getting the wrong version of some library or build tool.

> Will dynamic linking work?

Cross-compilation should be able to work with shared libraries. Regardless of whether the libraries are shared or static, libraries for the target machine must be available on the build host. I guess you would either install packaged libraries on the target machine, and copy those to the build host, or else cross-compile the libraries too. You need to configure search paths carefully so that both the compiler and the linker find the cross-libraries instead of the native ones. This is one of the complications you avoid by building natively.

> Which port?

Normally port 443, because of course you'll use HTTPS, won't you? Optionally you can also have an HTTP server on port 80 that responds to every request with a redirection to HTTPS.

If you choose to put AWS behind a reverse proxy like DrPi suggested, then the reverse proxy listens on port 443 on your public IP address, and you tell AWS to listen on some other port and only on the localhost address, ::1 or 127.0.0.1.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 14 Sep 2024 01:38:16 -0500

> That's called "security through obscurity". Not recommended.

That's the wrong way to look at it. An Ada program is better thought of as "security by simplicity and correctness", because you are running an Ada that only does a few things (and which can be thoroughly tested, checked with static analysis, and so on) rather than a general program that does a zillion things (with many combinations that can't be tested).

The only place "obscurity" comes into it is that no one else is running the exact same program as you. So attacks that depend on any sort of knowledge of the program cannot succeed.

In any case, there is no such thing as "secure", there are only levels, and for the sorts of non-critical stuff that we're doing, an Ada program is certainly secure enough. I wouldn't try to run a storefront on it (although that would be more because you'd have a hard time convincing your bank that it is OK than any real problems), or anything that needs high-level security.

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Sat, 14 Sep 2024 12:02:05 -0000

> work with Gnoga (https://v22.soweb.io).

Runs on Android/IOS. Does that require an internet web server?

From: Stéphane Rivière <stef@genesix.org>
Date: Sat, 14 Sep 2024 15:00:00 +0200

> Runs on Android/IOS.

Yes, v22.Gui/Gnoga is responsive. Tested with 5" smartphones as old as Nexus 5 (with a browser more recent than the stock one to handle websockets). Also tested on 43” 4K ;)

On some iOS devices, the menu bar is slightly offset. I didn't look too hard. It's a Safari problem. It works fine with Firefox and Chrome.

> Does that require an internet web server?

Not necessarily. v22.Gui/Gnoga supports itself X509 TLS https certificates (tested). However, for various reasons (such as the possibility of having several web applications on the same instance and on the same 80/443 input port), in production, I've always chosen to have a Nginx proxy on the front end, which is also more flexible and handles automatic switching from http/80 to https/443.

adaic.org; Is There a Problem?

From: John Mccabe <john@nospam.mccabe.org.uk>
Subject: adaic.org; is there a problem?
Date: Tue, 17 Sep 2024 16:19:47 -0000
Newsgroups: comp.lang.ada

Sorry to ask here; I wasn't sure where else to go, but is www.adaic.org ok for everyone? I'm just seeing a mostly white screen with a blackish bar at the top on Firefox, Chrome and Edge (Chrome on both Windows and Android). It might just be me, but I thought I'd ask in case anyone else sees it like that and can prod the right people to fix it or, alternatively, just let me know that it's a problem at my end!

From: Bill Findlay <findlaybill@blueyonder.co.uk>
Date: Tue, 17 Sep 2024 18:35:16 +0200

FOOBAR.

It looks as though a significant part of the HTML is missing.

From: John Mccabe <john@nospam.mccabe.org.uk>
Date: Tue, 17 Sep 2024 16:42:37 -0000

Thanks for that Bill; at least I'm not going mad then :-)

From: Dirk Craeynest <dirk@orka.cs.kuleuven.be>
Date: Tue, 17 Sep 2024 17:59:37 -0000

I noticed the problem with adaic.org as well, and have informed Randy, its webmaster, yesterday already. Stay tuned until he kicks the server back into action... ;-)

From: Blady <p.p11@orange.fr>
Date: Mon, 23 Sep 2024 20:31:42 +0200

I noticed a similar problem with www.ada-auth.org?

From: Luke A. Guest <laguest@archeia.com>
Date: Tue, 24 Sep 2024 13:56:47 +0100

Yup both down, just tried them.

Ada-related Tools

GNAT Studio 25.0 for macOS Ventura.

From: Blady <p.p11@orange.fr>
Subject: [ANN] GNAT Studio 25.0 for macOS Ventura.
Date: Fri, 26 Jul 2024 12:02:15 +0200
Newsgroups: comp.lang.ada

Here is a very preliminary version of GNAT Studio 25.0wa as a standalone app for macOS:
https://sourceforge.net/projects/gnuada/files/GNAT_GPL Mac OS X/2024-ventura

NEW:

The GNATStudio launcher looks for a gnatstudio_launcher.rc file in the .gnatstudio folder from either $HOME or $GNATSTUDIO_HOME locations. If it exists, we can define some environment variables with the standard syntax VAR=VALUE. If the VAR exists then VALUE is appended to it. If not, VAR is created with VALUE. Thus, it permits to set extra PATH to GNAT compiler and builder folders or GPR_PROJECT_PATH. If a line begins with '#' then it is not considered. An example file of gnatstudio_launcher.rc is provided in the archive. Modify the content and put it in your .gnatstudio folder.

See readme for details.

Limitation: Ada Language Server has some latences and doesn't respond when parsing source code with more than 1000 lines. It may be due to some compilation options I missed.

There could be some other limitations that you might meet.

Feel free to report them here.

Any help will be really appreciated to fix these limitations.

KDF9 Pascal, Thanks to Ada

From: Moi <findlaybill@blueyonder.co.uk>
Subject: KDF9 Pascal, thanks to Ada
Date: Tue, 2 Jul 2024 01:27:43 +0100
Newsgroups: comp.lang.ada

Some time ago it occurred to me that the best way to illustrate the remarkable architecture of the EE KDF9 would be to write a cross-compiler that generates idiomatic KDF9 Usercode (assembly language) and displays it in association with the source code.

I chose Pascal as the source language, having compiler texts available for retargeting.

PASKAL, which implements a large subset of Pascal, is now available.

The only parts of Pascal not implemented are file types and packed types, including the 'text' type, which means that there is no Standard Pascal I/O. However, I provide some basic KDF9-oriented output routines as a stopgap. They are more than adequate to show the correct execution of, for example, the Whetstone Benchmark, and many other classic codes, such as Quicksort.

PASKAL is written in Pascal, using the fpc compiler, and in Ada 2012, and is included with V11.2c of ee9, my KDF9 emulator (also in Ada 2012).

Included with it are the following documents:

* PASKAL: Users' Guide
* PASKAL: Object Program Structure.
* PASKAL: Implementation Overview

Compiled binaries are available for:

* Apple Silicon Macs
* Intel Macs
* 64-bit Intel (Debian Bookworm) Linux
* 64-bit Raspberry Pi (Debian Bookworm) OS

The Intel Linux binary should run under WSL on MS Windows 10 or 11.

Get your copy here:
http://www.findlayw.plus.com/KDF9/#PSK

There is a direct link there to the Users' Guide. It includes an example of a complete Pascal program and the corresponding KDF9 Usercode, should that be the extent of your interest.

Gnoga's 10th Anniversary - 2.2 Released.

From: Blady <p.p11@orange.fr>
Subject: Gnoga's 10th anniversary - V2.2 released.
Date: Sun, 8 Sep 2024 18:30:49 +0200
Newsgroups: comp.lang.ada

Gnoga was born on SourceForge [1] on September 8, 2014.

Gnoga (GNU Omnificent Gui for Ada) is the multi-platform graphics library created natively in Ada. I immediately liked Gnoga for the coherence and simplicity of these APIs naturally fitting together. The programmer can rely on Ada for his business code and on the multitude of Javascript libraries for the graphical interface.

For 10 years Gnoga has evolved in maturity to fulfill its founding principles:

- providing a framework and associated tools for developing GUI applications using the Ada language, leveraging web technologies for application developers
- developing native applications for desktop and mobile just as easy to create, all using the same code base
- providing better tools means better application quality
- offering the application developer a powerful toolset for secure cloud based computing, mobile apps, desktop apps and web apps the combination not found in any other set of tools in any other language

Gnoga statistics:

- 1031 commits
- 2200 downloads
- 2196 posts on the mailing list
- 56 tickets

You'll find a special Gnoga's wiki anniversary page [2] with some materials and my testimony.

Feel free to post your testimony, your own story with Gnoga.

On this occasion, Gnoga V2.2a has been released [3] and [4], with main changes:

- Added key field to keyboard event
- If present command line options gnoga-host, gnoga-port, gnoga-boot and gnoga-verbose will override host, port, boot file and verbosity programmed in source code (see TIPS).
- Improve logging implementation in a separate package in order to allow user defined logging handlers.
- Add a backslash compatibility mode on the behavior of Escape_String for SQLite with the one for MySQL.
- Change MYSQL_Real_Connect profile to better match with documentation

This version has been tested on macOS 13.6 and GNAT 14.1. Please provide feedback of other environments.

[1] https://sourceforge.net/p/gnoga/code/ci/45c76779e7af7b869deacc698478eb3ef25cfe91

[2] https://sourceforge.net/p/gnoga/wiki/Gnoga-Anniversary

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

[4] https://sourceforge.net/p/gnoga/code/ci/dev_2.2/tree

Ada Inside

Canal+ Crash

From: Nicolas Paul Colin De Glocester <master_fontaine_is_dishonest @strand_in_london.gov.uk>
Subject: Canal+ crash
Date: Fri, 19 Jul 2024 23:41:44 +0200
Newsgroups: fr.comp.lang.ada, comp.lang.ada

Canal+ uses Ada but one is alleging that Canal+ suffered a crash today with Windows. Cf.

https://www.UniversFreeBox.com/article/568957/orange-canal-et-bouygues-telecom-annoncent-a-leurs-abonnes-etre-touches-par-la-panne-informatique-mondiale

Cf. a complaint by Mister Brukardt that Ada cannot control non-Ada software on a shared system.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 20 Jul 2024 09:23:11 +0200

It is not about Ada. It is about the fundamental principle that security cannot be added on top of an insecure system. The lesson never learned is that security levels impose safety problems not solving security issues. Modern security architectures are nothing but a huge scam.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Sat, 20 Jul 2024 07:43:18 -0000

> It is about the fundamental principle that security cannot be added on top of an insecure system.

Actually, it can. Notice how the Internet itself is horribly insecure, yet we are capable of running secure applications and protocols on top of it.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 20 Jul 2024 11:08:47 +0200

> we are capable of running secure applications and protocols on top of it.

Of course we can. That is the whole idea of the scam. Why on earth do we need security updates? Do you update your screwdriver each week?

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Sun, 21 Jul 2024 01:04:44 -0000

> Why on earth do we need security updates?

Because computer systems are complex, and new bugs keep being discovered all the time.

> Do you update your screwdriver each week?

I don’t depend on my screwdriver to keep my bank account secure.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 21 Jul 2024 09:22:06 +0200

> Because computer systems are complex, and new bugs keep being discovered all the time.

This does not make sense. You can create a very complex system out of screwdrivers and still each screwdriver would require no update.

Systems consist of computers and computers of software modules. There is nothing inherently complex about making a module safe and bug free. Security interactions are primitive and 100% functional. There are no difficult issues with non-functional stuff like real-time problems. It is purely algorithmic while all mathematical complexity of cryptography is NOT what gets updated. It is complex only because it was designed as a Wood Block Tumbling Game.

> I don’t depend on my screwdriver to keep my bank account secure.

I don't need a bank account to fasten the screws. Application area is irrelevant.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sun, 21 Jul 2024 11:00:36 +0300

> Security interactions are primitive and 100% functional. There is no difficult issues with non-functional stuff like real-time problems.

Well, several recent attacks use variations in execution timing as a side-channel to exfiltrate secrets such as crypto keys. The crypto code can be functionally perfect and bug-free, but it may still be open to attack by such methods.

But certainly, most attacks on SW have used functional bugs such as buffer overflows.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sun, 21 Jul 2024 11:10:06 +0200

> But certainly, most attacks on SW have used functional bugs such as buffer overflows.

A problem that has been solved since 1983, and even before (Pascal had bounds checking). Sigh…

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 21 Jul 2024 11:19:30 +0200

> Well, several recent attacks use variations in execution timing as a side-channel to exfiltrate secrets such as crypto keys.

It is always a tradeoff between the value of the information and costs of breaking the protection. I doubt that timing attack are much more feasible in that respect than brute force.

> But certainly, most attacks on SW have used functional bugs such as buffer overflows.

Exactly. Non-functional attacks are hypothetical at best. They rely on internal knowledge which is another problem. An insider work is the most common case of all breaches. So, maybe, it is better to update the staff? (:-))

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 21 Jul 2024 11:34:14 +0200

> A problem that has been solved since 1983, and even before (Pascal had bounds checking). Sigh...

Yup, however some crackpot could always suggest an attack on bounds checking, e.g. exception vs. not, index to bounds comparison dependent on the actual values etc., and then produce a lengthy paper on a constructed absolutely unrealistic scenario... (:-))

From: Niklas Holsti
<niklas.holsti@tidorum.invalid>
Date: Sun, 21 Jul 2024 14:31:27 +0300

> I doubt that timing attack are much more feasible in that respect than brute force.

Security researchers and crypto implementers seem to take timing attacks quite seriously, putting a lot of effort into making the crucial crypto steps run in constant time.

> Non-functional attacks are hypothetical at best. They rely on internal knowledge which is another problem.

As I understand it, the "internal knowledge" needed for timing attacks is mostly what is easily discoverable from the open source-code of the SW that is attacked.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Sun, 21 Jul 2024 18:49:27 +0200

> Security researchers and crypto implementers seem to take timing attacks quite seriously

Cynically: they certainly know how to butter their bread...

> the "internal knowledge" needed for timing attacks is mostly what is easily discoverable from the open source-code

Considering many many layers of software to predict timing from code in uncontrolled environment would be a challenge.

From: Lawrence D'Oliveiro
<ldo@nz.invalid>
Date: Sun, 21 Jul 2024 21:52:58 -0000

> You can create a very complex system out of screwdrivers and still each screwdriver would require no update.

There is an old engineering adage, that the complexity of a system arises, not so much from the number of individual components, as from the number of potential interactions between them.

If you have a box full of screwdrivers, then all you have is a box full of screwdrivers.

If you have a computer system made up of a bunch of modules interacting with each other, then you could have, potentially, quite a complex system indeed.

Look up the term “combinatorial explosion” to learn more.

From: Lawrence D'Oliveiro
<ldo@nz.invalid>
Date: Sun, 21 Jul 2024 21:53:46 -0000

> A problem that has been solved since 1983, and even before (Pascal had bounds checking). Sigh...

Pascal had no checking for memory leaks or double-frees.

Rust certainly seems to be a next-generation solution to these sorts of memory problems.

From: Lawrence D'Oliveiro
<ldo@nz.invalid>
Date: Sun, 21 Jul 2024 21:55:10 -0000

> Considering many many layers of software to predict timing from code in uncontrolled environment would be a challenge.

And yet it has been successfully done on the hardware itself, right down under all those layers of software (cf Spectre/Meltdown).

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 22 Jul 2024 08:36:08 +0200

> Pascal had no checking for memory leaks or double-frees.

> Rust certainly seems to be a next-generation solution to these sorts of memory problems.

We were talking about bounds checking, that Pascal had. Nowadays, you should not use pointers directly, but containers. Pointers are necessary only for writing containers, thanks to Ada's features not found in other languages, like allocating dynamically sized arrays on the stack.

Note that in Rust, containers are written using unsafe Rust, therefore Rust is not better than Ada on that aspect, it is a complicated solution to a problem that Ada doesn't have.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Mon, 22 Jul 2024 09:16:09 +0200

> If you have a computer system made up of a bunch of modules interacting with each other, then you could have, potentially, quite a complex system indeed.

Tight coupling = bad design. No difference to screwdrivers. However you can take integer arithmetic if you dislike screwdrivers. However complex system you build, there is no need to update integers.

> Look up the term “combinatorial explosion” to learn more.

Bad design leads to explosion of non-trivial unanticipated system states making it unpredictable. This is what happens when you add security on top. You patch holes drilling new ones to fix the patches.

From: Lawrence D'Oliveiro
<ldo@nz.invalid>
Date: Tue, 23 Jul 2024 01:48:12 -0000

> We were talking about bounds checking, that Pascal had.

Which is only one potential pitfall for bugs with security implications.

Ada and Other Languages

“Red” and the DoD Language Competition

From: Lawrence D'Oliveiro
<ldo@nz.invalid>
Subject: “Red” And The DoD Language Competition
Date: Fri, 6 Sep 2024 01:55:00 -0000
Newsgroups: comp.lang.ada

While browsing around for Ada-related docs some years ago, I came across this site:
https://iment.com/maida/computer/redref/index.htm
which collects info on the DoD’s “Strawman”, “Woodenman”, “Tinman”, “Ironman” and “Steelman” series of RFPs, and the specs for the “Red” language that didn’t become Ada.

From: Luke A. Guest
<laguest@archeia.com>
Date: Sat, 7 Sep 2024 17:43:26 +0100

We have all the colours now:
https://www.reddit.com/r/ada/comments/165f5zg/common_hol_phase_1_reports/

Ada Practice

Accessing the Command Line

From: Lawrence D'Oliveiro
<ldo@nz.invalid>
Subject: Accessing The Command Line
Date: Thu, 4 Jul 2024 00:08:56 -0000
Newsgroups: comp.lang.ada

```
with Ada.Command_Line;
with Ada.Text_IO;
procedure Echo is
package cli renames
         Ada.Command_Line;
  package tio renames Ada.Text_IO;
  package int_io is new tio.Integer_IO
        (Num => Integer);
begin
  tio.put("my name: ");
  tio.put(cli.Command_name);
  tio.Put_Line("");
  tio.Put("nr args: ");
  int_io.Put(cli.Argument_Count, width => 1);
  tio.Put_Line("");
  for i in 1 .. cli.Argument_Count loop
    tio.put("[");
    int_io.put(i, width => 1);
    tio.put("]: ");
    tio.put(cli.argument(i));
    tio.put_line("");
  end loop;
end Echo;
```

Comments:

Ada, like Python, offers the convenience of being able to specify local “nicknames” for imported packages, to save some typing.

Having become used to the convenience of printf-style formatting in C and other languages that have adopted it (including Lisp and Python), I don't miss the tedium of having to format and output one item at a time. Though I recognize that there is no way to do printf style in a type-safe fashion, short of going to a fully-dynamic language.

Being able to access the POSIX command line via some globally-defined entity instead of arguments to a "mainline" procedure is something that just about every decent language offers. C is notably absent from this list.

From: Paul Rubin <no.email@nospam.invalid>
Date: Wed, 03 Jul 2024 18:16:01 -0700

> Though I recognize that there is no way to do printf style in a type-safe fashion, short of going to a fully-dynamic language.

C++ does that polymorphism stuff in iostream so you'd write std::cout << "my name: " << command_line << ...

Haskell does something sort of similar with additional machinery.

printf for some people is one of the motivations for dependent types.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Thu, 4 Jul 2024 01:50:59 -0000

> C++ does that polymorphism stuff in iostream so you'd write std::cout << "my name: " << command_line << ...

I know. The disadvantage of the C++ scheme is you cannot easily reorder items as necessary to fit the grammar of localized messages. That, I think, is why lots of other languages (including Python and Lisp) have copied the printf scheme, yet none have seen fit to copy the C++ scheme.

> printf for some people is one of the motivations for dependent types.

I wonder how you would express such a thing in an Ada-friendly fashion?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 4 Jul 2024 13:27:05 +0200

Remember that you can concatenate strings:

```
>    tio.put("my name: ");
>    tio.put(cli.Command_name);
>    tio.Put_Line("");
```

```
Tio.Put_Line (Item => "my name: " & Cli.Command_Name);
```

Image functions thus allow similar simplifications. 'Image is one such function, if you can accept the initial space for non-negative values:

```
>    tio.Put("nr args: ");
>    int_io.Put(cli.Argument_Count, width => 1);
>    tio.Put_Line("");
```

```
Tio.Put_Line (Item => "nr args:" & Cli.Argument_Count'Image);
```

For simple cases you can roll your own:

```
function Image (Value : in Integer) return String is
   Raw : constant String := Value'Image;
begin -- Image
   return Raw ( (if Value < 0 then 1 else 2) .. Raw'Last);
end Image;
```

```
>        tio.put("[");
>        int_io.put(i, width => 1);
>        tio.put("]: ");
>        tio.put(cli.argument(i));
>        tio.put_line("");
```

```
Tio.Put_Line (Item => '[' & Image (I) & "]: " & Image (Cli.Argument (I) ) );
```

For more complex uses, you can use something like PragmARC.Images[.Image]

(https://github.com/jrcarter/PragmARC).

You probably should review the definition of Ada.Text_IO

(http://www.ada-auth.org/standards/aarm12_w_tc1/html/AA-A-10.html), especially

for procedure New_Line.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 4 Jul 2024 15:01:05 +0200

> with Ada.Command_Line;

> with Ada.Text_IO;

[...]

A general advice processing strings, any strings: messages, commands, payload etc.

Always read a complete string into a fixed size buffer (safety). Never use streams. Process the whole string consequently. Never tokenize. Never copy anything. Ada has slices.

The same is true for the output. Build a complete substring in a buffer. Consequently. Flush the complete substring to the output.

Do not use Unbounded_String.

From: Rod Kay <rodakay5@gmail.com>
Date: Fri, 5 Jul 2024 01:13:36 +1000

> I wonder how you would express such a thing in an Ada-friendly fashion?

There is the 'GNAT.Formatted_String' package, which provides 'printf' functionality.

Unfortunately, its formatting is somewhat buggy and has been so for many years. Usage is quite simple and reasonably elegant but the occasional incorrect formatting is a major problem, essentially rendering the package useless.

There is also the new 2022 f"X = {An_X_Variable} notation for embedding Variable'Image into strings, which is very nice. However, it does not allow for formatting, so not useful for your needs. Just thought I'd mention it, as it is now available in GCC 14.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 4 Jul 2024 18:15:54 +0200

>> printf for some people is one of the motivations for dependent types.

> I wonder how you would express such a thing in an Ada-friendly fashion?

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

From: Ben Bacarisse <ben@bsb.me.uk>
Date: Thu, 04 Jul 2024 20:42:00 +0100

> ... Though I recognize that there is no way to do printf style in a type-safe fashion, short of going to a fully-dynamic language.

Not so. Haskell has Text.Printf.

From: Paul Rubin <no.email@nospam.invalid>
Date: Thu, 04 Jul 2024 15:06:00 -0700

> Not so. Haskell has Text.Printf.

Text.Printf is not fully type safe. printf "%d\n" "foo" throws a "bad formatting character" exception, really amounting to a runtime type exception.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Thu, 4 Jul 2024 23:54:49 -0000

> Remember that you can concatenate strings:

> Tio.Put_Line (Item => "my name: " & Cli.Command_Name);

I'm sure I can, but I'm not sure what the point is. Let Ada collect the pieces in its own buffers. That saves copying steps.

> PragmARC.Images[.Image] (https://github.com/jrcarter/PragmARC).

I don't really feel the need to resort to third-party libraries just to do simple I/O.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 5 Jul 2024 10:58:00 +0200

> I'm sure I can, but I'm not sure what the point is. Let Ada collect the pieces in its own buffers. That saves copying steps.

Agreed. I don't understand why people dislike printing piece by piece. In the old FORTRAN, you could write only line by line, but this time is long gone…

With the various Put procedures, you have individual formatting options that you don't have otherwise. Moreover, there is a nice property that few people noticed: if you have an algorithm writing data to a file, with loops and so on, you can keep the exact same structure replacing every Put with the matching Get, and you will read your data correctly. This feature goes away as soon as you have a 'Image.

Reduction Expressions

From: Simon Wright <simon@pushface.org>
Subject: Reduction expressions
Date: Tue, 13 Aug 2024 13:36:54 +0100
Newsgroups: comp.lang.ada

Are the Accum_Type & Value_Type (ARM 4.5.10(9/5)) of a reduction attribute reference required to be definite?

ARM 4.5.10(24/5) & (25.5) seem to imply so, which explains why GNAT doesn't support e.g. String.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 19 Aug 2024 22:59:04 -0500

Accum_Subtype (we changed the name since it is a subtype, not a type; various clarifications were made to the wording as well in AI22-0011-1, AI22-0047-1, and AI22-0069-1) most likely has to be definite since the accumulator is of that type, and the bounds/constraints of the accumulator are thus defined by the initial value. In most uses, the first call on Reduce would then raise Constraint_Error (because the bounds/constraints are incorrect). I don't think there is any reason that the Value_Subtype has to be definite for a sequential reduce (a parallel reduce requires the two subtypes to statically match).

Note that if someone has a clever way to use an indefinite result, it is allowed. For instance, I could see a class-wide result making sense in some limited circumstances. But I don't think String would do anything useful, since the bounds are determined by the initial value.

BTW, this answer is essentially topic #1 of AI22-0011-1.

From: Simon Wright <simon@pushface.org>
Date: Tue, 20 Aug 2024 22:23:27 +0100

> Accum_Subtype (we changed the name since it is a subtype, not a type;

Amazing how a person (I) can have used Ada for ~40 years and still be hard put to it to describe the difference, at least in a case like this one, where the ARG members clearly see meanings that leave me lukewarm if not cold. Maybe "the heart of twilight"?

> But I don't think String would do anything useful

String was just the simplest indefinite type for an example.

> BTW, this answer is essentially topic #1 of AI22-0011-1.

Thanks for the pointer.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Tue, 20 Aug 2024 23:30:54 -0000

> Amazing how a person (I) can have used Ada for ~40 years and still be hard put to it to describe the difference

I thought the difference was obvious. “subtype” is the C equivalent of “typedef”, just giving a new name to an existing type. So

```
subtype A is B;
```

(where A and B are simple identifiers) is valid, whereas

```
type A is B;
```

is not: a “type” declaration always creates a new type: you have to write at least

```
type A is new B;
```

and now you have two types with different names that are structurally the same, but not compatible.

From: Keith Thompson <keith.s.thompson+u@gmail.com>
Date: Tue, 20 Aug 2024 16:41:55 -0700

> I thought the difference was obvious. “subtype” is the C equivalent of “typedef” [...]

A subtype with no added constraint is similar to a C typedef, but given

```
subtype Digit is Integer range 0..9;
```

Digit is distinct from Integer (though they're both the same type).

C doesn't have anything directly corresponding to Ada subtypes.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Wed, 21 Aug 2024 01:37:22 -0000

> Digit is distinct from Integer (though they're both the same type).

“Integer range 0..9” is a subtype of Integer, and is valid for example as a return type where Integer is expected. The “subtype” declaration doesn’t actually create the subtype: “Digit” is just a shorthand name for that, just like a C typedef.

From: Simon Wright <simon@pushface.org>
Date: Wed, 21 Aug 2024 08:47:49 +0100

> I thought the difference was obvious. [...]

Yes, I've understood that for a long time but ... ARM22 4.5.10(8,9)[1] say

(8) The expected type for a reduction_attribute_reference shall be a single nonlimited type.

(9) In the remainder of this subclause, we will refer to nonlimited subtypes Value_Type and Accum_Type of a reduction_attribute_reference. ...

and in AI 22-0011-1 [2] starting at 22-Oct-2021 5:25 PM,

* SB: raises a series of observations,

* STT: "... You really need to think of Accum_Type as a particular *subtype*"

* SB: "Ok, I was confused - Accum_Type is a subtype, not a type. So a lot of my message was noise."

If SB can be confused, so can I!

[1] http://www.ada-auth.org/standards/22rm/html/RM-4-5-10.html#p8

[2] http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai22s/ai22-0011-1.txt?rev=1.2

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 23 Aug 2024 23:27:48 -0500

> If SB can be confused, so can I!

Which is why we changed the name - if SB can be confused, it is a good bet that there is something wrong with the wording. That's why I usually recommend bleeding edge users use the bleeding edge RM - no point in rediscovering all of the bugs that we already know about. Unfortunately, in this case, I'm the only one that has the bleeding edge RM because I haven't finished adding all of the approved AIs to it. This group is some that I've done, which is why the answer to your question was relatively easy to find.

Ichbiah 2022 Compiler Mode

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Subject: Ichbiah 2022 compiler mode
Date: Thu, 5 Sep 2024 11:52:37 -0000
Newsgroups: comp.lang.ada

I guess this is a very subjective question.

A number of Ada users have expressed that they would rather Ada was simpler whilst others desire more features.

I appreciate Ada 83 portability but also like a lot of modern Ada features.

Out of interest. Could anyone help me with what a GNAT or other compiler Ichbiah_2022_Mode might look like. Perhaps it might be possible to use pragmas to get an estimated mode of what features he might keep or drop.

I can continue research but currently I do not have the details of his objections to Ada 95 and how those may have continued through to today is perhaps a nuanced question.

What do you think Ichbiah would jettison from Ada 2022? All comments welcome.

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 5 Sep 2024 15:40:35 +0200

> I do not have the details of his objections to Ada 95

Ichbiah's objections to Ada 95 are in https://web.elastic.org/~fche/mirrors/old-usenet/ada-with-null

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Thu, 5 Sep 2024 16:08:01 -0000

What does this mean?

"elimination of accuracy constraints in subtypes"

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Thu, 5 Sep 2024 21:24:05 +0200

> "elimination of accuracy constraints in subtypes"

See ARM-95 J.3 (https://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-J-3.html),

Reduced Accuracy Subtypes.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 5 Sep 2024 19:03:22 -0500

> What do you think Ichbiah would jettison from Ada 2022?

My recollection is that he wanted a more complex "class" feature, which IMHO would have made Ada more complex, not simpler.

In any case, I can't guess what Ichbiah would have suggested after 40 years of experience. (He probably would have moved on to some other language anyway, you have to be somewhat resistant to change to stick with a single language for your entire career. I seem to resemble that remark... ;-)

What I can do is suggest what an RLB_2022 mode would look like, as I did the exercise when we all were cooped up during the early days of the pandemic. My philosophy is that Ada has a lot of combinations of features that cause a lot of implementation trouble, but which are not very useful. So I want to reduce the combinations that cause trouble. I note that every feature is useful for something (else it wouldn't be in Ada in the first place). But some things are not useful enough for the trouble that they cause. Also note that I am not worrying about compatibility with Ada, which is always a problem when updating Ada itself.

Here's some highlights off the top of my head:

(1) Simplify the resolution model; essentially everything resolves like a subprogram. For instance, objects resolve similarly to enumeration literals. This substantially reduces the danger of use clauses (having matching profiles and names is less likely than just matching names), and eliminates the subtle differences between a constant and a function (they should really act the same).

(2) Operator functions have to be primitive for at least one of the types in the profile. (Operators in a generic formal part have a pseudo-primitive requirement.) That includes renamings. In exchange for that, operators have the same visibility as the type (which means they are always directly visible when any object of the type is visible). One then can eliminate "use type" (since it would literally do nothing).

(3) A number of syntax options are eliminated. Matching identifiers are required at the end of subprograms and packages. Initializers are always required (<> can be used if default initialization is needed). Keyword "variable" is needed to declare variables (we do not want the worst option to be the easiest to write, as it is in Ada).

(4) Anonymous types of all sorts are eliminated. For access types, we would use aspects to declare properties (static vs. dynamic accessibility, "closure" types, etc.). For arrays, see next item.

(5) The array model would be greatly simplified. New Ada users (and old ones as well) have a hard time dealing with the fact that the lower bound is not fixed in Ada. Additionally, the existing Ada model is very complex when private types are involved, with operators appearing long after a type is declared. The more complex the model, the more complex the compiler, and that means the more likely that errors occur in the compiler. There also is runtime overhead with these features. The basic idea would be to provide the features of an Ada.Containers.Vector, and no more. Very little is built-in. That means that arrays can only be indexed by integers, but that is a good thing: an array indexed by an enumeration type is really a map, and should use a map interface. So I would add a Discrete_Map to the Ada.Containers packages. Bounded_Arrays are a native type (most of the uses of arrays that I have are really bounded arrays built by hand).

A side-effect of this model change is to greatly simplify what can be written as discriminant-dependent components. Discriminant-dependent arrays as we know them are gone, replaced by a parameterized array object that has only one part that can change. Much of the nonsense associated with discriminant-dependent components disappears with this model.

(6) Static items have to be declared as such (with a "static" keyword rather than "constant"). Named numbers are replaced by explicit static constants. (I would allow writing Universal_Integer and Universal_Real, so one could declare static objects and operations of those types.)

(7) Types and packages have to be declared at library-level. This means that most generic instances also have to be declared at library-level. Subtypes, objects, and subprograms still can be declared at any nesting level. I make this restriction for the following reasons:

(A) Accessibility checks associated with access types are simplified to yes/no questions of library-level or not. The only cases where accessibility checks do any real good is when library-level data structures are constructed out of aliased objects. These would still be allowed, but almost all of the complication would be gone. Even if the check needs to be done dynamically, it is very cheap.

(B) Tagged types declared in nested scopes necessarily require complex dynamic accessibility checks to avoid use of dangling types (that is, an object which exists of a type that does not exist).

(C) Reusability pretty much requires ODTs to be declared in library-level packages. Mandating that won't change much for most programs, and you'll be happier in the long run if you declare the types in library packages in the first place.

(D) There are a lot of semantic complications that occur from allowing packages in subprograms, but this is rarely a useful construct.

(8) Protected types become protected records (that is, a regular record type with the keyword "protected"). Primitive operations of a protected record type are those that are protected actions. (Entries can be declared and renamed as such, they would no longer match procedures, which leads to all kinds of nonsense.) This would eliminate the problems declaring helper types and especially *hiding* helper types for protected types. (See the problems we had defining the queues in the Ada.Containers to see the problem.) The protected operations would allow the keyword "protected" in order to make the subprograms involved explicit.

(9) Strings are not arrays! Strings would be provided by dedicated packages, supporting a variety of representations. There would be a Root_String'Class that encompasses all string types. (So many

operations could be defined on Root_String'Class).

(10) Variable-returning functions are introduced. They're pretty similar the semantics of anonymous access returns (or the aliased function returns suggested by Tucker). This means that a variable can easily be treated as a function (and indeed, a variable declaration is just syntactic sugar for such a function).

(11) Various obsolete features like representation_clauses, representation pragmas, and the ability to use 'Class on untagged private types are eliminated or restricted.

There were a couple of areas that I never made up my mind on:

(A) Do we need tasks at all? Parallel and task are very much overlapping capabilities. But the parallel model would need substantial changes if we were to allow suspension of parallel threads (Ada 2022 does not allow this). Suspension seems necessary to support intermittent inputs of any type (including interrupts) without wasting resources running busy-wait loops.

(B) Should type conversions be operators or remain as the type name as in Ada? A type conversion operator, automatically constructed, would allow user-defined types to have the same sort of conversions to numeric and string types that the predefined do. But an operator would make conversions easier, which is probably the wrong direction for a strongly typed language.

(C) I wanted to simplify the assignment model, but my initial attempt did not work semantically. I'm not sure that simplification is possible with the Ada feature set (I'm sure Bob and Tuck tried to do that when creating Ada 95, but they failed). The main issue is that one would like to be able to replace discriminant checks on user-defined assignment. (Imagine the capacity checks on a bounded vector; Ada requires these to match, but that's way too strong; the only problem is if the target capacity cannot hold the actual length of the source object. A user-defined replacement would be helpful.)

My $20 worth (this was a lot more work than $0.02!!). I probably forgot a number of items; my actual document is about 20 pages long.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Fri, 6 Sep 2024 00:58:05 -0000

> Keyword "variable" is needed to declare variables

One language idea I toyed with years ago was that

```
«name» : «type»;
```

declared a variable, while

```
«name» : «type» := «value»;
```

declared a constant. So, no initialization of variables at declaration time allowed.

> (10) Variable-returning functions are introduced.

Is this like updater functions in POP-11, or "setf" in Lisp? So you have a procedure

```
set_var(«var», «new value»)
```

which is declared to be attached to «var» in some way, such that when you write

```
«var» := «new_value»
```

this automatically invokes set_var?

From: Jeffrey R.Carter <spam.jrcarter.not@spam.acm.org.not>
Date: Fri, 6 Sep 2024 13:07:27 +0200

> Out of interest. Could anyone help me with what a GNAT or other compiler Ichbiah_2022_Mode might look like.

I have no idea what he would have done. For an idea of what I think a language should have, you can look at my informal description of King (https://github.com/jrcarter/King).

From: Simon Wright <simon@pushface.org>
Date: Fri, 06 Sep 2024 22:22:08 +0100

> (A) Do we need tasks at all? Parallel and task are very much overlapping capabilities.

I don't think I've ever wanted parallel. Most embedded system tasks are one-off, aren't they?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 7 Sep 2024 20:13:03 +0300

> I don't think I've ever wanted parallel. Most embedded system tasks are one-off, aren't they?

More and more embedded systems use multi-core processors and do heavy, parallelizable computations. "Parallel" is intended to support that in a light-weight way. In a recent discussion with the European Space Agency, they expressed interest in using OpenMP for such computations on-board spacecraft with multi-core processors, which is an "embedded" context.

Regarding tasks in embedded systems, I agree that most are one-off, but I have occasionally also used tens of tasks of the same task type.

I disagree with Randy's view that tasks and "parallel" are much overlapping. Tasks are able to communicate with each other, but AIUI parallel tasklets are not meant to do that, and may not be able to do that. Tasks can have different priorities; tasklets cannot.

From: Nioclás Pól Caileán De Ghloucester <master_fontaine_is_dishonest @strand_in_london.gov.uk>
Date: Sat, 7 Sep 2024 22:34:52 +0200

"[...] they expressed interest in using OpenMP for such computations [...]"

Hei!

Most of the languages which are referred to by WWW.OpenMP.org/resources/ openmp-compilers-tools facilitate bugs. (The Spark which is referred to thereon is not the Ada-related Spark language.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 11 Sep 2024 23:39:27 -0500

>> (10) Variable-returning functions are introduced.

> this automatically invokes set_var?

No, it is a function that returns a variable, meaning you can assign into the function result. If you have:

```
function Foo return variable Integer;
```

then you can use Foo on either side of an assignment:

```
Foo := 1;

Bar := Foo + 1;
```

Essentially, this idea treats:

```
Var : variable Integer;
```

as syntactic sugar for

```
function Var return variable Integer;
```

The worth of that is two-fold: (1) Objects and functions now resolve the same; (2) one can write a function that acts exactly like an object, and thus can replace it in all uses.

Note that Ada currently has generalized reference objects and functions that return anonymous access types, and both of these act similarly to a variable returning function. But neither is quite a perfect match.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Thu, 12 Sep 2024 22:24:29 -0000

> No, it is a function that returns a variable, meaning you can assign into the function result.

I think an updater function would be more generally useful. Because some things you want to update might not (depending on the implementation) live independently in an explicit variable. And it seems good not to constrain implementations unnecessarily.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 14 Sep 2024 01:18:25 -0500

> I think an updater function would be more generally useful.

Unfortunately, "updater" functions don't work with the Ada model of components, because you can't tell what to do when a component appears or disappears in an assignment. (That's why Ada doesn't allow overloading ":=".) And composition is very important to Ada -- stand-alone objects are pretty rare outside of those for scalar types. I don't think something that only worked with stand-alone objects would be very useful (can't use those with ODTs, for instance).

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Sat, 14 Sep 2024 07:18:29 -0000

> Unfortunately, "updater" functions don't work with the Ada model of components [...]

But it's just syntactic sugar, nothing more. Instead of

```
a := obj.get_prop()

obj.set_prop(a)
```

(both of which have valid Ada equivalents), you can unify them into

```
a:= obj.prop

obj.prop := a
```

What difference does writing it differently make?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 11 Sep 2024 23:46:18 -0500

> Tasks are able to communicate with each other, but AIUI parallel tasklets are not meant to do that, and may not be able to do that. Tasks can have different priorities; tasklets cannot.

I was (of course) presuming that "tasklets" would get those capabilities if they were to replace tasks. That's what I meant about "suspension", which is not currently allowed for threads in Ada (parallel code is not allowed to call potentially blocking operations). If that was changed, then all forms of existing task communication would be allowed.

I'm less certain about the value of priorities, most of the time, they don't help writing correct Ada code. (You still need all of the protections against race conditions and the like.) I do realize that they are a natural way to express constraints on a program. So I admit I don't know in this area, in particular if there are things that priorities are truly required for.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 12 Sep 2024 10:42:38 +0300

> I was (of course) presuming that "tasklets" would get those capabilities

Ok, I understand. In that case, what "parallel" adds to the current tasking feature is an easy way to create a largish and perhaps dynamically defined number of concurrent threads from a "parallel" loop, where the threads are automatically created when the loop is started and automatically "joined" and destroyed when the loop completes.

I don't mind at all if a future Ada evolution merges tasks and "parallel", although it might defeat the easier access to multi-core true parallelism that is the goal of the "parallel" extension, AIUI.

> I'm less certain about the value of priorities

Priorities (or the equivalent, such as deadlines) are absolutely required for real-time systems where there are fewer cores than concurrent/parallel activities so that the system has to schedule more than one such activity on one core.

If Ada did not have tasks with priorities, most of the Ada applications I have worked on in my life would have had to avoid Ada tasking and retreat to using some other real-time kernel, with ad-hoc mapping of the kernels's threads to Ada procedures.

Despite the transition to multi-core processors, I think that there will continue to be systems where scheduling is required, because the number of concurrent/parallel activities will increase too.

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 12 Sep 2024 11:04:58 +0200

> I was (of course) presuming that "tasklets" would get those capabilities [...]

Well, tasks are not only for speeding up code. They can be a very useful design tool (active objects, independent activities). I think the Ada model is clean and simple, I would hate to see it disappear.

> I'm less certain about the value of priorities [...]

If you had as many cores as tasks, you would not need priorities. Priorities are just optimization on how to manage cores when there are not enough of them.

I know that people use priorities to guarantee mutual exclusion, and other properties. All these algorithms were designed at the time of mono-CPU machines, but they fail on multi-cores. Nowadays, relying on priorities for anything other than optimization is bad -and dangerous- design.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 12 Sep 2024 11:07:01 +0200

> I don't mind at all if a future Ada evolution merges tasks and "parallel" [...]

To me usefulness of "parallel" is yet to be seen, while tasks proved to be immensely useful on all architectures available.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 12 Sep 2024 14:35:27 +0300

> Well, tasks are not only for speeding up code. They can be a very useful design tool (active objects, independant activities). I think the Ada model is clean and simple, I would hate to see it disappear.

I agree.

> Priorities are just optimization on how to manage cores when there are not enough of them.

In some contexts it could be optimization -- for example, to increase throughput in a soft real-time system -- but in hard real-time systems priorities (or deadlines) are needed for correctness, not just for optimization.

> I know that people use priorities to guarantee mutual exclusion, and other properties. All these algorithms were designed at the time of mono-CPU machines, but they fail on multi-cores.

In SW for multi-core systems it can be beneficial to collect tasks that frequently interact with each other or with the same single-user resources in the same core, and then the mono-core mutual-exclusion algorithms like priority ceiling inheritance can be used for that group of tasks, while using other algorithms for mutual exclusion between tasks running in different cores.

From: Kevin Chadwick <kc-usenet@chadwicks.me.uk>
Date: Thu, 12 Sep 2024 12:36:19 -0000

>If Ada did not have tasks with priorities, most of the Ada applications I have worked on in my life would have had to avoid Ada tasking and retreat to using some other real-time kernel, with ad-hoc mapping of the kernels's threads to Ada procedures.

Counter intuitively it is possible that this is holding Ada back. A lot of Ada code cannot run without some fairly complex runtime support due to tasks, protected objects, finalization etc.. Runtimes have to be developed for each chip instead of each cpu. At Least I assume that that is why these features are not available to e.g. the light cortex-m33 or cortex-m4 or cortex-m0+ runtimes. This requires rewriting code which isn't required with equivalent C code such as containers and ip stacks etc.. Even support for the Ada interrupt package is missing but it looks like porting that support to chips is less work and research.

If you need advanced multi core support then using an OS seems like a more suitable situation to be in to me.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Thu, 12 Sep 2024 18:43:45 +0300

> Counter intuitively it is possible that this is holding Ada back. [...]

True, however an Ada RTS can implement many of the tasking features with moderate effort on top of non-Ada real-time kernels such as FreeRTOS, VxWorks, etc., as AdaCore have done for some kernels. At least for the Ravenscar and Jorvik profiles. AIUI, the processor-specific stuff is then mainly in the kernel, not in the RTS.

> If you need advanced multi core support then using an OS seems like a more suitable situation to be in to me.

Using a large OS like Linux would not be acceptable for many embedded systems. Fortunately the smaller real-time kernels are adding multi-core support too.

The great advantage of using the standard Ada tasking feature, special syntax and all, is that your embedded Ada program can then be executed on a PC or other non-embedded computer, for testing or other purposes, tasking and all. It can also be analysed by static-analysis tools such as AdaControl for race conditions and other tasking-sensitive issues.

From: Nioclás Pól Caileán De Ghloucester <master_fontaine_is_dishonest @strand_in_london.gov.uk>
Date: Fri, 13 Sep 2024 22:45:03 +0200

> Counter intuitively it is possible that this is holding Ada back [...]

A book by Burns and Wellings unsensibly boasts that the demanding runtime demands of Ada are an advantage because if you are with them then you are with them, whereas as Kevin Chadwick points out - they are not easy to make.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 14 Sep 2024 01:13:28 -0500

> [...] in hard real-time systems priorities (or deadlines) are needed for correctness, not just for optimization.

This I don't buy: priorities never help for correctness. At least not without extensive static analysis, but if you can do that, you almost certainly can do the correctness without depending upon priorities.

I view priorities as similar to floating point accuracy: most people use them and get the results they want, but the reason for that is that they got lucky, and not because of anything intrinsic. Unless you do a lot of detailed analysis, you don't know if priorities really are helping or not (and similarly, whether your results actually are meaningful in the case of floating point).

Anyway, I don't see any such changes coming to Ada, but rather to some separate follow-on language (which necessarily needs to be simpler), and thus some things that are sometimes useful would get dropped.

(Different message)

...

> [...] what "parallel" adds to the current tasking feature is an easy way to create a largish and perhaps dynamically defined number of concurrent threads from a "parallel" loop [...]

I think the parallel block is more useful for general tasking. The advantage of using parallel structures is that they look very similar to sequential structures, and one lets the system do the scheduling (rather than trying to figure out an organization manually).

One of the advantages of the model I'm thinking about is that it separates concerns such as parallel execution, mutual exclusion, inheritance, organization (privacy, type grouping), and so on into separate (mostly) non-overlapping constructs. Ada started this process by having tagged types a separate construct from packages; you need both to get traditional OOP, but you can also construct many structures that are quite hard in traditional "one construct" OOP. I think that ought to be done for all constructs, and thus the special task and protected constructs ought to go. We already know that protected types cause problems with privacy of implementation and with inheritance. Tasks have similar issues (admittedly less encountered), so splitting them into a set of constructs would fit the model.

In any case, this is still a thought experiment at this time, whether anything ever comes of it is unknown.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 14 Sep 2024 08:47:49 +0200

> The advantage of using parallel structures is that they look very similar to sequential structures, and one lets the system do the scheduling (rather than trying to figure out an organization manually).

Tasking is not about scheduling. It is about program logic expressed in a sequential form. It is about software decomposition. Parallel constructs simply do not do that.

> One of the advantages of the model I'm thinking about is that it separates concerns such as parallel execution, mutual exclusion, inheritance, organization (privacy, type grouping), and so on into separate (mostly) non-overlapping constructs.

To me it is exactly *one* construct: inheritance. You should be able to inherit from an abstract protected interface at any point of type hierarchy in order to add mutual exclusion:

```
type Protected_Integer is new Integer and
   Protected;
```

> Ada started this process by having tagged types a separate construct from packages;

I see modules and types as unrelated things.

> you need both to get traditional OOP, but you can also construct many structures that are quite hard in traditional "one construct" OOP. I think that ought to be done for all constructs, and thus the special task and protected constructs ought to go.

Constructs yes, they must go. It must be all inheritance. The concepts must stay.

> We already know that protected types cause problems with privacy of implementation and with inheritance. Tasks have similar issues (admittedly less encountered) [...]

The problems are of syntactic nature, IMO.

There is an issue with an incomplete inheritance model. You need not just complete overriding but also more fine mechanisms like extension in order to deal with entry point implementations. The same problem is with constructors and destructors, BTW. What should really go is Ada.Finalization mess replaced by a sane user construction hooks model for all types, class-wide ones included.

From: Lawrence D'Oliveiro <ldo@nz.invalid>
Date: Sat, 14 Sep 2024 07:19:47 -0000

> ... priorities never help for correctness.

Concurrent programming was never about correctness, it was about efficiency/performance (throughput, latency, whatever is appropriate). And priorities are just another part of this.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 14 Sep 2024 11:12:43 +0300

> This I don't buy: priorities never help for correctness. At least not without extensive static analysis, but if you can do that, you almost certainly can do the correctness without depending upon priorities.

You misunderstood me; perhaps I was too brief.

I said "hard real-time systems", which means that the program is correct only if it meets its deadlines, for which priorities or deadline-based scheduling are necessary if there are fewer cores than concurrent/parallel activities, and the application has a wide range of deadlines and activity execution times.

(To be honest, there is the alternative of using a single thread that is manually sliced into small bits, interleaving all the activities increment by increment, according to a static, cyclic schedule, but that is IMO a horribly cumbersome and unmaintainable design, though unfortunately still required in some contexts.)

I believe we agree that priorities should be used for other things, such as controlling access to shared data, only if there is a well-defined and safe mechanism for it, such as protected objects with priority ceilings and priority inheritance on a single core.

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

October 01-03	2024 **International Conference on Software Engineering Research & Development** (SERD'2024), Oklahoma City, Oklahoma, USA & Online. Topics include: general and social aspects of software engineering (SE); software design, testing, evolution, and maintenance; formal methods and theoretical foundations; programming languages (PLs), systems, and environments; object- oriented (OO) design and analysis; emerging SE technologies and dependability; distribution, componentization, and collaboration; concurrent, parallel and distributed systems; etc.
October 05	**Ada Monthly Meetup 2024 October**, 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.
October 07-08	24th IEEE **International Working Conference on Source Code Analysis and Manipulation** (SCAM'2024), Flagstaff, Arizona, USA. Topics include: abstract interpretation, bad smell detection, clone detection, program comprehension, program slicing, program transformation and refactoring, security vulnerability analysis, source level metrics, source level optimization, source-level testing and verification, static and dynamic analysis, etc.
☺ October 13-16	33rd **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2024), Long Beach, California, USA. Topics include: parallel architectures; compilers and tools for parallel architectures; applications and experimental systems studies of parallel processing; computational models for concurrent execution; support for correctness in hardware and software; reconfigurable parallel computing; parallel programming languages, algorithms, and applications; middleware and run time system support for parallel computing; distributed computing architectures and systems; etc.
October 15-18	24th **International Conference on Runtime Verification** (RV'2024), Istanbul, Türkiye. Topics include: monitoring and analysis of runtime behavior of software, hardware, and cyber-physical systems; program instrumentation; combination of static and dynamic analysis; monitoring techniques for concurrent and distributed systems; fault localization, containment, resilience, recovery and repair; etc.
October 15-18	24th **International Conference on Formal Methods in Computer-Aided Design** (FMCAD'2024), Prague, Czech Republic. Topics include: methods, technologies, theoretical results, and tools for reasoning formally about computing systems; formal aspects of computer-aided system design including verification, specification, synthesis, and testing; etc.
October 16-18	17th **International Conference on Verification and Evaluation of Computer and Communication Systems** (VECoS'2024), Djerba, Tunisia. Topics include: analysis of computer and communication systems, where functional and extra-functional properties are inter-related; cross-fertilization between various formal verification and evaluation approaches, methods and techniques, especially those developed for concurrent and distributed hardware/software systems.
October 20-22	31st **Static Analysis Symposium** (SAS'2024), Pasadena, USA. Co-located with SPLASH'2024. Topics include: static analysis as fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance.
☺ October 20-25	ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2024), Pasadena, California, USA.

October 20-21 17th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2024). Topics include: software language engineering in general rather than engineering a specific software language; software language design and implementation; validation of software language tools and implementations (verification and formal methods, testing techniques, simulation techniques); software language maintenance (software language reuse; language evolution; language families and variability, language and software product lines); software language integration and composition domain-specific approaches for any aspects of SLE; (analysis, design, implementation, validation, maintenance); empirical studies and experience reports of tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.

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

October 21-24 21st **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2024), Kyoto, Japan. Topics include: theoretical and practical aspects of automated analysis, synthesis, and verification of hardware and software systems; program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent systems; verification in industrial practice; applications and case studies; automated tool support; etc.

☺ October 22 **High Integrity Software Conference** (HISC'2024), Newport, South Wales, UK. Topics include: advanced software development for high-integrity and high-assurance systems, including programming languages, verifiable code generation; verification and testing of high-integrity systems; assurance of high-integrity systems; infrastructure and ecosystem for high-integrity software; etc.

October 22-24 22nd **Asian Symposium on Programming Languages and Systems** (APLAS'2024), Kyoto, Japan. Topics include: all areas of programming languages and systems; programming paradigms and styles; methods and tools to specify and reason about programs and languages; programming language foundations; methods and tools for implementation; concurrency and distribution; applications, case studies and emerging topics.

Oct 27 – Nov 01 39th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2024), Sacramento, California, USA. Topics include: foundations, techniques, and tools for automating analysis, design, implementation, testing, and maintenance of large software systems.

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.

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 early registration: October 15, 2024.

☺ Nov 07-08 32nd **International Conference on Real-Time Networks and Systems** (RTNS'2024), Porto, Portugal.

Nov 06 — 17th **Junior Researcher Workshop on Real-Time Computing** (JRWRTC'2024). Topics include: real-time system design and analysis (task and message scheduling; modeling, verification and evaluation; model-driven development; worst-case execution time estimation; distributed systems; fault tolerance; quality of service and security), software technologies for real-time systems (compilers and programming languages, middleware and component-based technologies, tools, ...), real-time applications (automotive and avionics applications; process control; telecommunications and multimedia; medical applications), etc. Deadline for submissions: October 3, 2024 (abstracts), October 8, 2024 (full papers).

November 13-15 — 19th **International Conference on integrated Formal Methods** (iFM'2024), Manchester, UK. Topics include: recent research advances in the development of integrated approaches to formal modelling and analysis; all aspects of the design of integrated techniques, including language design, verification and validation, automated tool support and the use of such techniques in software engineering practice.

November 13-15 — 29th IEEE **Pacific Rim International Conference on Dependable Computing** (PRDC'2024), Osaka, Japan. Topics include: software and hardware reliability, resilience, safety, security, testing, verification, and validation; dependability measurement, modeling, evaluation, and tools; architecture and system design for dependability; reliability analysis of complex systems; dependability issues in computing systems (e.g. high performance computing, real-time systems, cyber-physical systems, ...); emerging technologies (autonomous systems including autonomous vehicles, human machine teaming, smart devices/Internet of Things); etc.

☺ November 17 — **Parallel Applications Workshop, Alternatives to MPI+X** (PAW-ATM'2024), Atlanta, Georgia, USA. Topics include: alternatives to the MPI+X model, novel application development using high-level parallel programming languages and frameworks; examples that demonstrate performance, compiler optimization, error checking, and reduced software complexity; experience with the use of new compilers and runtime environments; etc.

November 21-22 — 23rd **Belgium-Netherlands Software Evolution Workshop** (BENEVOL'2024), Namur, Belgium. Topics include: software evolution and maintenance. Deadline for registration: October 21, 2024.

November 26-29 — 13th **Latin-American Symposium on Dependable Computing** (LADC'2024), Recife, Pernambuco, Brazil. Topics include: all aspects of dependable and secure systems and networks; critical infrastructure protection, cyber-physical systems, safety-critical systems; software (software frameworks and architectures, model-driven engineering, testing, V&V, certification, runtime verification, ...); formal methods for dependable and secure systems; etc.

December 03-06 — 31st **Asia-Pacific Software Engineering Conference** (APSEC'2024), Chongqing, China. Topics include: requirements and design; component-based software engineering; software architecture, modeling and design; middleware, frameworks, and APIs; software product-line engineering; testing and analysis; testing, verification, and validation; program analysis; program repairs; formal aspects of software engineering; formal methods; model-driven and domain-specific engineering; software comprehension and traceability; dependability, safety, and reliability; software maintenance and evolution; refactoring; reverse engineering; software reuse; debugging and fault localization; software repository mining; etc.

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

☺ 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: October 15, 2024 (TCRTS award nominations).

December 13-15 — 25th **International Conference on Parallel and Distributed Computing, Applications, and Techniques** (PDCAT'2024), Hong Kong, China. Topics include: all facets of parallel and distributed computing, task mapping and job scheduling, formal methods and programming languages, parallel/distributed algorithms, security and privacy, high performance systems, etc.

2025

☺ January 19-25 — 52nd ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2025), Denver, Colorado, USA. Topics include: all aspects of programming languages and programming systems, both theoretical and practical; fundamental principles and important innovations in the design, definition, analysis, transformation, implementation and verification of programming languages, programming systems, and programming abstractions.

January 20-21 **International Conference on Certified Programs and Proofs** (CPP'2025). Topics include: research areas related to formal certification of programs and proofs; new languages and tools for certified programming; program analysis, program verification, and program synthesis; program logics, type systems, and semantics for certified code; teaching mathematics and computer science with proof assistants; etc.

January 20-21 26th **International Conference on Verification, Model Checking, and Abstract Interpretation** (VMCAI'2025), Denver, Colorado, USA. Co-located with POPL'2025. Topics include: program verification, model checking, abstract interpretation, static analysis, type systems, program certification, detection of bugs and security vulnerabilities, hybrid and cyber-physical systems, concurrent and distributed systems, analysis of numerical properties, analysis of smart contracts, etc., case studies on all of the above topics. Deadline for submissions: October 1, 2024 (papers).

January 20-22 20th **International Conference on High Performance and Embedded Architecture and Compilation** (HiPEAC'2025), Barcelona, Spain. Topics include: computer architecture, programming models, compilers and operating systems for general-purpose, embedded and cyber-physical systems.

☺ January 20 6th **Workshop on Next Generation Real-Time Embedded Systems** (NG-RES'2025). Topics include: application of formal methods to distributed and/or parallel real-time systems; programming models, paradigms and frameworks for real-time computation on parallel and heterogeneous architectures; dependable systems and networks; compiler-assisted solutions for distributed and/or parallel real-time systems; middlewares for distributed and/or parallel real-time systems; scheduling and schedulability analysis for distributed and/or parallel real-time systems; etc. Deadline for paper submissions: November 17, 2024.

February 02 12th **Ada Developer Room at FOSDEM 2025**, Brussels, Belgium. FOSDEM 2025 is a two-day event (Sat-Sun 1-2 Feb), held in hybrid mode. This year's edition includes once more an Ada Developer Room, held on Sunday morning 2 February.

February 04-06 19th **International Working Conference on Variability Modelling of Software-Intensive Systems** (VaMoS'2025), Rennes, France. Topics include: variability across the software lifecycle, test and verification of variable systems, evolution of variability-intensive systems, runtime variability, variability mining, reverse-engineering of variability, economic aspects of variability, variability and quality requirements, industrial development of variable systems, experience reports from managing variability in practice, etc. Deadline for submissions: October 25, 2024 (abstracts), November 1, 2024 (papers).

March 01-05 ACM/IEEE **International Symposium on Code Generation and Optimization** (CGO'2025), Las Vegas, USA.

March 04-07 32nd IEEE **International Conference on Software Analysis, Evolution, and Reengineering** (SANER'2025), Montreal, Quebec, Canada. Topics include: software tools for software evolution and maintenance; software analysis, parsing, and fact extraction; software reverse engineering and reengineering; program comprehension; software evolution analysis; software architecture recovery and reverse architecting; program transformation and refactoring; mining software repositories and software analytics; software reconstruction and migration; software maintenance and evolution; program repair; software release engineering, continuous integration and delivery; empirical studies on all the above topics; education related to all of the above topics; etc. Deadline for submissions: October 4, 2024 (research track abstracts), October 11, 2024 (research track papers), November 1, 2024 (industrial track abstracts, Reproducibility Studies and Negative Results (RENE) track abstracts), November 4, 2024 (short papers and posters track abstracts, Early Research Achievement (ERA) track abstracts), November 6, 2024 (registered report track), November 8, 2024 (industrial track papers, Reproducibility Studies and Negative Results (RENE) track papers), November 11, 2024 (short papers and posters track papers, Early Research Achievement (ERA) track papers, tool demo track papers), November 17-22, 2024 (workshop paper abstracts), November 8-29, 2024 (workshop papers), December 3, 2024 (Journal-First track papers).

March 12-14 33rd Euromicro/IEEE **International Conference on Parallel, Distributed and Network-Based Processing** (PDP'2025), Turin, Italy. Topics include: embedded parallel systems; dependability, survivability, fault-tolerance; programming languages, compilers, middleware; runtime, systems software; performance prediction and analysis; simulation and modeling of parallel/distributed systems; etc. Deadline for submissions: October 29, 2024 (abstracts), November 5, 2024 (papers).

Mar 30 - Apr 03 20th **European Conference on Computer Systems** (EuroSys'2025), Rotterdam, the Netherlands. Topics include: all areas of computer systems research, such as distributed systems, language support and runtime systems, systems security and privacy, dependable systems, analysis, testing and verification of systems, parallelism, concurrency, and multicore systems, real-time, embedded, and cyber-physical systems, etc. Fall deadline for submissions: October 15, 2024 (abstracts), October 22, 2024 (submissions).

Mar 31 - Apr 04 40th ACM/SIGAPP **Symposium on Applied Computing** (SAC'2025), Catania, Italy. Topics include: latest developments, trends, experiences, and challenges in applied computing. 42 specialized tracks, including: cyber-physical systems; dependable, adaptive, and secure distributed systems; embedded system; IoT and edge computing; interoperability; programming languages; software engineering; computer security; software verification and testing; etc.

☺ Mar 31-Apr 04 **Track on Programming Languages** (PL'2025). 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 4, 2024 (full papers), January 9, 2025 (student travel award program application). Deadline for author registration: December 6, 2024.

Mar 31-Apr 04 **Software Verification and Testing Track** (SVT'2025). 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. Deadline for submissions: October 13, 2024 (regular papers, student research competition research abstracts).

Mar 31-Apr 04 20th **Track on Dependable, Adaptive, and Secure Distributed Systems** (DADS'2025). 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. Deadline for paper submissions: October 4, 2024.

Mar 31 - Apr 04 22nd IEEE **International Conference on Software Architecture** (ICSA'2025), Odense, Denmark. Topics include: linking architecture to requirements and/or implementation; methods to address the intertwining of specification and design; model-driven architecture; component-based software engineering; architecture frameworks and architecture description languages; evaluating quality aspects (e.g., security, performance, reliability, evolvability); automatic extraction and generation of software architecture descriptions; architecture & continuous integration/delivery, and DevOps; refactoring and evolving architecture design decisions and solutions; roles and responsibilities for software architects; training, soft skills, coaching, mentoring, education, and certification; etc. Deadline for submissions: November 8, 2024 (abstracts), November 15, 2024 (full papers).

Mar 31-Apr 01 9th **International Workshop on Formal Approaches for Advanced Computing Systems** (FAACS'2025). Topics include: integration between formal methods and software architecture, architecture description languages and metamodels, model-driven engineering, approaches and tools for verification and validation, reports on practical experience in the application of formal methods to industrial case studies, etc. Deadline for submissions: December 20, 2024.

Mar 31 - Apr 04 18th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2025), Naples, Italy. Topics include: formal verification; replications, empirical studies, case studies, experience reports; software reliability; static and dynamic analysis; test automation; testability, test design, and adequacy criteria; testing and development processes; testing, debugging, and repair tools; testing in specific domains (embedded/cyber-physical systems, concurrent, distributed, real-time systems, ...);

testing of non-functional properties such as security; etc. Deadline for submissions: October 2, 2024 (full papers).

April 07-08 11th **International Conference on Fundamentals of Software Engineering** (FSEN'2025), V ster s, Sweden. Topics include: all aspects of formal methods, especially those related to advancing the application of formal methods in the software industry and promoting their integration with practical engineering techniques; models of programs and software systems; software specification, validation, and verification; software testing; software architectures and their description languages; integration of formal and informal methods; component-based and service-oriented software systems; cyber-physical software systems; model checking and theorem proving; software verification; CASE tools and tool integration; industrial applications; etc. Deadline for submissions: October 7, 2024 (abstracts), October 14, 2024 (papers).

April 07-10 31st **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2025), Barcelona, Spain. Theme: "Social REsponsibility". Deadline for submissions: October 25, 2024 (workshops), November 1, 2024 (research paper abstracts), November 8, 2024 (research papers), February 7, 2025 (papers to workshops, education and training, posters and tools, and doctoral symposium).

April 08-11 20th **European Dependable Computing Conference** (EDCC'2025), Lisbon, Portugal. Topics include: latest ideas and results on theory, experiments, techniques, systems and tools for the design, validation, operation and evaluation of dependable and secure computing systems; hardware and software architecture of dependable systems; dependability and security modelling, evaluation, and tools; safety-critical systems design and analysis; mixed-criticality systems design and evaluation; testing and validation methods; dependability and security of: artificial intelligence systems, cyber-physical systems, e.g. intelligent vehicles, (industrial) Internet of Things, ...; etc. Deadline for submissions: October 7, 2024 (full papers).

Apr 26 – May 04 47th **International Conference on Software Engineering** (ICSE'2025), Ottawa, Ontario, Canada. Topics include: the full spectrum of Software Engineering (SE), trustworthy AI for SE; AI-assisted software design and model driven engineering; mining software repositories; software metrics (and measurements); software design methodologies, principles, and strategies; architecture quality attributes, such as security, privacy, performance, reliability; modularity and reusability; dependency and complexity analysis; patterns and anti-patterns; technical debt in design and architecture; formal methods and model checking; reliability, availability, and safety; resilience and antifragility; design for dependability and security; vulnerability detection to enhance software security; dependability and security for embedded and cyber-physical systems; evolution and maintenance; API design and evolution; software reuse; refactoring and program differencing; program comprehension; reverse engineering; environments and software development tools; human and social aspects (focusing on programming languages, environments, and tools supporting individuals, teams, communities, and companies; focusing on software development processes; ...); modeling and model-driven engineering; variability and product lines; modeling languages, techniques, and tools; empirical studies on the application of model-based engineering; software testing; automated test generation techniques such as fuzzing, search-based approaches, and symbolic execution; testing and analysis of non-functional properties; program analysis; debugging and fault localization; runtime analysis and/or error recovery; etc. Deadline for submissions: October 1 - November 19, 2024 (abstracts co-located conferences), October 8 - December 6, 2024 (papers co-located conferences), October 21, 2024 (TCSE Award nomination intention), October-November, 2024 (special tracks), November 4, 2024 (TCSE Award nomination), November 11, 2024 (workshop papers).

April 27-28 13th **International Conference on Formal Methods in Software Engineering** (FormaliSE'2025). Topics include: approaches, methods and tools for verification and validation; formal approaches to safety and security related issues; scalability of formal method applications; integration of formal methods within the software development lifecycle; model-based engineering approaches; correctness-by-construction approaches for software and systems engineering; application of formal methods to specific domains, e.g., autonomous, cyber-physical, intelligent, and IoT systems; formal methods in a certification context; case studies developed/analyzed with formal approaches; experience reports on the application of formal methods to real-world problems; guidelines to use formal methods in practice; usability of formal methods; etc. Deadline for submissions: November 11, 2024 (abstracts), November 18, 2024 (papers).

Apr 27 - May 03 37th **International Conference on Software Engineering Education and Training** (CSEET'2025), Ottawa, Ontario, Canada. Topics include: all dimensions of learning and teaching in the area of software engineering. Deadline for submissions: October 3, 2024 (abstracts), October 10, 2024 (papers).

May 03-08 28th ETAPS **International Joint Conferences on Theory and Practice of Software** (ETAPS'2025), Hamilton, Canada. Deadline for submissions: October 10, 2024 (TACAS, FoSSaCS, FASE), October 24, 2024 (TACAS artifacts), January 9, 2025 (FASE, FoSSaCS artifacts).

> May 05-08 24rd **European Symposium on Programming** (ESOP'2025). Topics include: fundamental issues in the specification, design, analysis, and implementation of programming languages and systems, such as 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, etc. Deadline for submissions: October 10, 2024 (round 2).

May 06-09 18th **Cyber-Physical Systems and Internet of Things Week** (CPS-IoT Week'2025), Irvine, USA. Event includes: 5 top conferences, HSCC, ICCPS, IoTDI, IPSN, and RTAS, as well as poster and demo sessions, workshops, tutorials, competitions, industrial exhibitions, PhD forums, and summits.

> May 06-09 16th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2025). Topics include: safety and resilience for CPS; software platforms and systems for CPS; specification languages and requirements; design, optimization, and synthesis; testing, verification, certification, assurance; security, trust, and privacy in CPS; tools, testbeds, demonstrations and deployments; CPS applications in power systems, infrastructure networks, transportation, healthcare, automotive, aerospace, etc. Deadline for submissions: November 7, 2024 (abstracts), November 14, 2024 (papers).

May 12-16 28th **Ibero-American Conference on Software Engineering** (CIbSE'2025), Ciudad Real, Spain. Topics include: community-based software engineering (SE) (e.g., open source, crowdsourcing); ethics in SE; industrial experience reports in SE; software architecture and variability; software ecosystems and systems of systems; SE education and training; SE for emerging application domains (cyber-physical systems, Internet of Things, ...); software evolution and modernisation; software modeling and model-driven engineering; software processes; software product lines and processes; software quality, quality models and technical debt management; software reliability; software repository mining and software analytics; software reuse; software testing; etc. Deadline for submissions: January 17, 2025 (abstracts), January 31, 2025 (papers, doctoral symposium, journal first).

May 20-22 17th **Software Quality Days** (SWQD'2025), Munich, Germany. Theme: "Balancing Software Innovation and Regulatory Compliance" Topics include: all topics related to software and systems quality; 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, and automotive systems; continuous integration, deployment, and delivery; project and risk management; secure coding, software engineering, and system design; detection and prevention of vulnerabilities and security threats; etc. Deadline for submissions: October 31, 2024.

June 03-07 39th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2025), Milan, Italy. Topics include: research in high performance computing in parallel and distributed processing; real-world applications that use parallel and distributed computing concepts; experiments and performance-oriented studies in the practice of parallel and distributed computing; 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); etc. Deadline for submissions: October 3, 2024 (abstracts), October 10, 2024 (papers).

♦ June 10-13 29th **Ada-Europe International Conference on Reliable Software Technologies** (AEiC'2025), Paris, France. Organized by Ada-Europe and Ada-France. Deadline for submissions: January 20, 2025 (journal track papers), February 24, 2025 (industrial track and work-in-progress papers, tutorial and workshop proposals). #AEiC2025 #AdaEurope #AdaProgramming

June 12-13 **Software Technologies: Applications and Foundations** (STAF'2025), Koblenz, Germany.

> June 12-13 18th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2025). Topics include: software language engineering in general, rather than

	engineering a specific software language, such as software language design and implementation, software language validation (verification and formal methods for languages, testing techniques for languages, simulation techniques for languages, ...), software language integration and composition, software language maintenance (software language reuse; language evolution; language families and variability, language and software product lines), domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance), empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications), etc. Deadline for submissions: February 7, 2025 (abstracts), February 14, 2025 (papers).
June 23-27	33rd ACM **International Conference on the Foundations of Software Engineering** (FSE'2025), Trondheim, Norway. 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.
September 09-12	44th **International Conference on Computer Safety, Reliability and Security** (SafeComp'2025), Stockholm, Sweden. Topics include: all aspects related to the development, assessment, operation, and maintenance of safety-related and safety-critical computer systems; fault detection and recovery mechanisms; safety guidelines and standards; safety/security co-engineering and trade-offs; safety and security qualification, quantification, assurance and certification; threats and vulnerability analysis; 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; safety-related applications of smart systems and IoT; critical infrastructures, smart grids, SCADA; medical devices and healthcare; surveillance, defensed, emergency & rescue; logistics, industrial automation, off-shore technology; education & training. Deadline for submissions: 7 February 2025 (workshops, abstracts), 14 February 2025 (full papers).
☺ October 12-18	ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2025), Singapore.
	☺ Oct 12-25 **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2025). Deadline for submissions: October 15, 2024 (round 1), March 25, 2025 (round 2).
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

29th Ada-Europe International Conference on
Reliable Software Technologies (AEiC 2025)
10-13 June 2025, Paris, France

General Information

The **29th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2025)** will take place in Paris, France. The conference schedule comprises a journal track, an industrial track, a work-in-progress track, a vendor exhibition, parallel tutorials, and satellite workshops.

- Journal track papers present research advances supported by solid theoretical foundation and thorough evaluation.
- Industrial track contributions highlight the practitioners' side of a challenging case study or industrial project.
- Work-in-progress track 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.

Schedule

20 January 2025	Deadline for submission of journal track papers
24 February 2025	Deadline for submission of industrial track papers, work-in-progress papers, and tutorial and workshop proposals
28 March 2025	First round notification for journal track papers, and notification of acceptance for all other types of submissions
10-13 June 2025	Conference

Scope and Topics

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

The topics of interest for the conference include but are not limited to:

- Formal and model-based engineering of critical systems
- High-integrity systems and reliability
- AI for high-integrity systems engineering
- Real-time systems
- Ada language
- Applications in relevant domains

More specific topics are described on the conference web page.

Conference Chair
Jean-Pierre Rosen
rosen@adalog.fr
Adalog & Ada-France

Journal track Co-chairs
Laurent Pautet
laurent.pautet@telecom-paris.fr
Telecom Paris
Sara Royuela
sara.royuela@bsc.es
Barcelona Supercomputing Center

Industrial track Co-chairs
Daniela Cancila
daniela.cancila@cea.fr
CEA LIST
Laurent Gouzenes
lgouzenes@pactenovation.fr
Pacte-Novation

Work-in-progress track Co-chairs
Hai Nam Tran
hai-nam.tran@univ-brest.fr
University of Brest
Anish Bhobe
anish.bhobe@telecom-paris.fr
Telecom Paris

Workshop Chair
Anish Bhobe
anish.bhobe@telecom-paris.fr
Telecom Paris

Tutorial Chair
Robert Cholay
robert.cholay@free.fr
Systerel

Exhibition & Sponsorship Chair
Ahlan Marriott
ahlan@Ada-Switzerland.ch
White Elephant GmbH

Finance Chair
Paul Duquennoy
paul.duquennoy@free.fr

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

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

Local Chair
Pierre Jouvelot
pierre.jouvelot@minesparis.psl.eu
Mines Paris, PSL University

Ada europe

http://www.ada-europe.org/conference2025

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 3.7). Contributions must be submitted by **20 January 2025**.

JSA has adopted the Virtual Special Issue model to speed up the publication process, where Special Issue (SI) 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. Ada-Europe 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, Laurent Pautet and Sara Royuela.

Call for Industrial Track Submissions

The conference seeks industrial-practitioner presentations that deliver insight on the challenges of developing reliable software. Especially welcome kinds of submissions are listed on the conference web site. Given their applied nature, such contributions will be subject to a dedicated practitioner-peer-review process. Interested authors shall submit a one-to-two pages abstract, by **24 February 2025**.

The abstracts 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, Daniela Cancila and Laurent Gouzenes.

Call for Work-in-progress Track Submissions

The work-in-progress (WiP) 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 **24 February 2025**.

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, Hai Nam Tran and Anish Bhobe.

Call for Tutorials

The conference seeks tutorials in the form of educational seminars on themes falling within the conference scope, with an academic-for-practitioner slant, including hands-on or practical elements. Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half-day or full-day), the intended level of the contents (introductory, intermediate, or advanced), and a statement motivating attendance. Tutorial proposals shall be submitted at any time but no later than **24 February 2025** to the respective chair, Robert Cholay. 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 **24 February 2025** to the respective chair, Anish Bhobe. Once submitted, each workshop proposal will be evaluated by the conference organizers as soon as possible.

Call for Exhibitors

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

Venue

The conference will take place at Mines Paris. Mines Paris - PSL, a founding member of Université PSL, is a leading French engineering school and the French leader institution in research partnerships. Founded 240 years ago to help spur the energy efforts called by the Industrial Revolution, it has been since training engineers in a wide spectrum of scientific disciplines. With about 1,500 students, including 100 PhD graduates per year, Mines Paris - PSL hosts 18 research centres and 5 academic departments. It is located along the Luxembourg gardens, next to the Quartier Latin, and is close to public transportation, including line B of the RER to Charles De Gaulle airport (CDG).

Paris, the capital city of France, is renowned for its rich history, stunning architecture, and vibrant culture. Often referred to as "The City of Light," Paris is home to iconic landmarks such as the Eiffel Tower, the Louvre Museum, and Notre-Dame Cathedral. The city is a global centre for art, fashion, gastronomy, science and culture, attracting millions of visitors each year.

Front picture generated with the assistance of AI (DALL·E 2). Back picture provided by Mines Paris.

First Ada Developers Workshop at AEiC 2024

Fernando Oleo Blanco
Open Source & Ada aficionado; Tel: +34 689 44 27 45; email: irvise@irvise.xyz

Dirk Craeynest
KU Leuven, Dept. of Computer Science, B-3001 Leuven, Belgium; email: Dirk.Craeynest@cs.kuleuven.be

Abstract

At this year's Ada-Europe conference, the new "Ada Developers Workshop" created an informal platform for Ada developers to meet, share insights, and present their latest projects or project updates. A short overview of the full-day program is given, followed by reports of the presentations by their respective authors.

Keywords: developers, community, Ada.

1 Introduction

During the 28th Ada-Europe International Conference, AEiC 2024 [1], which took place in Barcelona this year, a new hybrid workshop focusing on the Ada programming language, Ada-related technology, and the Ada community was born on Friday 14 June. This *"Ada Developers Workshop"* [2] was proposed and organised to give the Ada community an informal place where projects, ideas and topics could freely be shared and discussed among fellow Ada users.

One of the goals of the workshop was to make it as affordable as possible, so as to lower the barrier of entry and maximize the community's involvement. Thanks to the sponsorship of AdaCore [3] and Ada-Europe [4] the registration fee for on-site participation, including breaks and lunch, was minimal, and there was no fee at all for online participation. Additionally, AdaCore sponsored much of the recording and streaming equipment that was used. The workshop was mainly organised by Fernando Oleo Blanco, Fabien Chouteau and Dirk Craeynest, with the help of the Barcelona Supercomputing Center [5], and saw a good level of interest: 19 in-person participants, plus some 15-20 remotely for part or all of the day.

2 Program overview

In this first workshop edition, 9 presentations took place, given by 8 speakers from 5 countries. In chronological order they were:

- *"SweetAda: a Multi-architecture Embedded Development Framework"*, by Gabriele Galeotti (Italy) and Fernando Oleo Blanco (Spain);
- *"Avoiding Access Types"*, by Jeffrey R. Carter (Belgium);
- "*G-NAV: Soaring the Clouds with AdaWebPack*", by Guillermo A. Hazebrouck (Belgium);
- *"Alire 2.0: a 'Quality of Life' Update*", by Alejandro Mosteo (Spain);
- *"HiRTOS: a Multi-core RTOS Written in SPARK Ada"* presented J. German Rivera (USA);
- *"Ironclad: a Formally Verified OS Kernel Written in SPARK/Ada"*, by Cristian Simon (Spain);
- *"An Ada Story of Time"*, by Jean-Pierre Rosen (France);
- *"Controlled I/O: a Library for Scope-Based Files*", by Jeffrey R. Carter (Belgium);
- *"Ada Community Advocacy"*, by Fernando Oleo Blanco (Spain).

The topics covered by the different presentations were quite diverse: we had two talks focused on Operating Systems: HiRTOS and Ironclad; one on embedded systems development: SweetAda; one about navigation and using Ada for web development and related technologies (WASM): G-NAV; one about package management: Alire; one about Ada and its features: Avoiding Access Types; one about utilities and libraries for Ada: Controlled I/O; one about the theory, measurement and management of time: An Ada Story of Time; and finally, we had a presentation about Ada and its evolving community: Ada Community Advocacy.

3 Informal proceedings

All presenters were invited to prepare short papers. The collection of those mini-papers is provided in this AUJ issue, as informal proceedings of the workshop.

In addition, the slides as well as the video recordings of all presentations are publicly available via the workshop's website [2].

References

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

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

[3] https://www.adacore.com/

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

[5] https://www.bsc.es

SweetAda: a Lightweight Ada-based Framework

Gabriele Galeotti

SweetAda home, 50019 Sesto Fiorentino (FI), Italy; email: gabriele.galeotti@sweetada.org, gabriele.galeotti.xyz@gmail.com

Abstract

This article introduces and describes the status of SweetAda[1] development since the last 0.10 release.

Keywords: Ada, SweetAda, embedded, RISC-V, ARM

1 Introduction

SweetAda is a lightweight development framework whose purpose is the implementation of Ada-based software systems directly on hardware. SweetAda can be thought of as an advanced embedded systems framework, with plenty of functionality and quality-of-life features, that does not reach Operating System levels of complexity nor size. The users of SweetAda are encouraged to further develop and build their own solutions based on the building blocks that SweetAda provides and create tailored solutions for their hardware.

The code produced by SweetAda is able to run on a wide range of machines, from ARM® embedded boards up to x86-64-class machines, as well as RISC-V machines and Virtex®/Spartan® PowerPC®/MicroBlaze® FPGAs. It could theoretically run even on System/390® IBM® mainframes (indeed it runs on the Hercules emulator). SweetAda is not an operating system, however it includes a set of both low- and high-level primitives and kernel services, like memory management, PCI bus handling, FAT mass-storage handling, which could be used as building blocks in the construction of complex software-controlled devices.

More than 2 years have passed since the last official release of SweetAda 0.10 and, after almost 3000 commits, things have changed significantly. Much of the work done has led SweetAda into a usable product in various contexts.

We are going to describe the latest changes, features and additions, which were also shown during the Ada Developers Workshop [1] that took place with the AEiC 2024 event, last June, in Barcelona, Spain.

2 Development Environment

The native environment of SweetAda was primarily the mainstream PC machine running a Linux OS. But this seems almost too limiting, since many users prefer to stick with a Windows machine or a macOS one, so one of the targets was to make the SweetAda environment usable under these very different OSes. This generates a huge set of problems, ending up in dealing with filesystem quirks, shell portability, and so on.

Putting focus on the classical *nix shell, we had to make shell scripts portable by sticking to POSIX rules. This was not simple, since it prevents the use of arrays, and imposes severe limits in the handling of objects. But careful programming was successful and, in the end, a user could use SweetAda nearly in every shell with a Bash-style heritage, like ash, or the shell of a macOS machine. MSYS2 in Windows is working perfectly too. Making it work in a cmd.exe was another different story, and a set of PowerShell scripts were necessary to process the whole build system, due to the intrinsic limitations of this ancient command line processor.

The final results are that, nowadays, SweetAda is usable in many mainstream machines, works with every GNAT toolchain, and is completely agnostic, since it does not use any third-party packages.

3 Build System

GNATMAKE was initially selected due to its wide availability, since it is automatically generated when you build a GNAT compiler from the sources. It is simple to use, and it is fast. The AdaCore GPRbuild system gained popularity, so it was natural to try to make SweetAda use this kind of tool. Nowadays, one can use any of these two tools in SweetAda.

One nice side effects is that SweetAda has a .gpr configuration file that is correctly loaded inside GNAT Studio, and you can use this IDE to do development, like editing, computing metrics, and so on. GNAT Studio detects the build targets inside the SweetAda Makefile and you can issue commands like in an interactive shell.

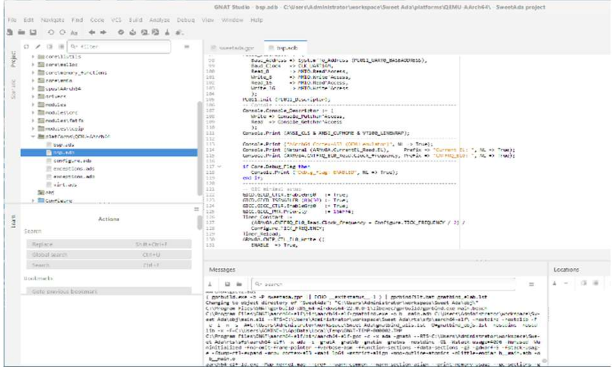

GNAT Studio - bsp.adb - C:\Users\Administrator\workspace\Sweet Ada\platforms\QEMU-AArch64\ - SweetAda project
Debug
View
Window
Help
sweetada.gpr
bsp.adb
Messages
Locations

Figure 1: SweetAda GNAT Studio session in Windows

[1] https://www.sweetada.org, https://github.com/gabriele-galeotti/SweetAda

4 Ada Code Development

A lot of cleanup in the code was also necessary. Many unused and/or redundant subprograms have been deleted, and the code formatting has been made more mainstream.

The core complex has been completely made CPU-independent, and nearly all subprograms are now separate, leading to the possibility of overriding them with user-optimized code.

SweetAda now has a very basic mutex object which is sufficient to do some experiments in SMP. Indeed, you can find a primitive form of SMP in the QEMU-AArch64 and QEMU-RISC-V target platforms, where 4 CPUs run in parallel. Physical boards like Raspberry Pi are in the initial phase of this kind of development.

5 Targets

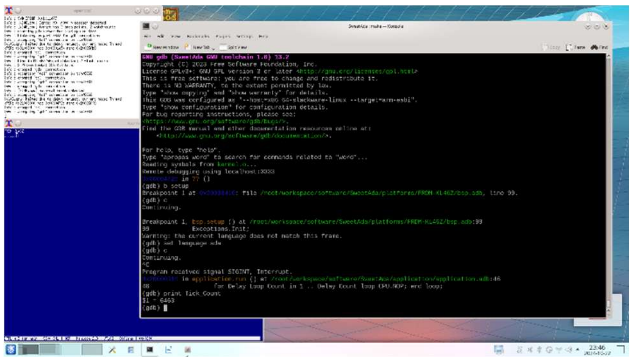
Various targets are new in SweetAda, focusing on those which have popularity and wide attention. So, RISC-V device targets, both virtual and physical, were implemented.

The first attempt was to experiment on a SiFive HiFive 1 board. Then other targets were positively checked, like FPGA implementations, or even complete virtual VHDL environments running under GHDL. As a matter of fact, SweetAda run flawlessly in a NEORV32 device placed in Xilinx or Altera chips. Fernando Oleo Blanco [2] was able to autonomously and easily port SweetAda to a ULX3S FPGA board, and gently showed us this achievement during the AEiC '24 meeting.

The OpenRisc CPU was also implemented, at least in a QEMU platform.

6 Running and Debugging

Real code runs on real platforms. Great effort was put to make SweetAda usable on physical target platform, so an infrastructure to use OpenOCD is now readily available. By simply configuring some script variables, Ada code can be made run on JTAG-enabled hardware, and GDB debugging is fully enabled.



```
GNU gdb (SweetAda GNU toolchain 1.0) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-slackware-linux --target=arm-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from kernel.o...
Remote debugging using localhost:3333
[illegible] in ?? ()
(gdb) b setup
Breakpoint 1 at [illegible]: file /root/workspace/software/SweetAda/platforms/FRDM-KL46Z/bsp.adb, line 99.
(gdb) c
Continuing.

Breakpoint 1, bsp.setup () at /root/workspace/software/SweetAda/platforms/FRDM-KL46Z/bsp.adb:99
99              Exceptions.Init;
Warning: the current language does not match this frame.
(gdb) set language ada
(gdb) c
Continuing.
^C
Program received signal SIGINT, Interrupt.
[illegible] in application.run () at /root/workspace/software/SweetAda/application/application.adb:46
46                  for Delay_Loop_Count in 1 .. Delay_Count loop CPU.NOP; end loop;
(gdb) print Tick_Count
$1 = 6463
(gdb)
```

Figure 2: FRDM-KL46Z target under debugging in an OpenOCD session driven by SweetAda build system

References

[1] https://www.youtube.com/watch?v=pI-WWgJWgnc

[2] https://irvise.xyz/

Trademarks

Linux® is a registered trademark of Linus Torvalds.

POSIX™ is a registered trademark of IEEE.

macOS™ is a trademark of Apple, Inc.

Windows® is a registered trademark of Microsoft Corporation.

SiFive™ is a trademark of SiFive, Inc.

Xilinx™ is a trademark of Advanced Micro Devices, Inc.

Altera® is a registerd trademark of Intel Corporation.

QEMU™ is a trademark of Fabrice Bellard.

Raspberry Pi is a trademark of Raspberry Pi Foundation.

All other trademarks and trade names are properties of their respective owners.

Avoiding Access Types

Jeffrey R. Carter
PragmAda Software Engineering; email: jrcarter@acm.org; https://github.com/jrcarter

Abstract

This paper is a summary of the presentation at the 2024 Ada-Europe International Conference Ada Developers Workshop. Access types and their associated memory management are a constant source of errors, so it is desirable to avoid them. Access types are very rarely needed. Some techniques for avoiding their use are presented.

Keywords: access types, memory management, Ada.

1 Introduction

Ada has two types of access types: access-to-object types and access-to-subprogram types. Here we are dealing with access-to-object types and will call them "access types" for short.

Access types and their associated memory management are a constant source of errors, so it is desirable to avoid them. In Ada, access types are never needed (valid as a first-order approximation, and probably as second). Most uses can be replaced with unbounded components from the standard library; most remaining cases can be encapsulated and hidden within a custom component. Encapsulation and hiding make it easier to get the memory management right.

There are three main cases where access types can be avoided:

- Returning large objects with unknown constraints
- Making private types private
- Self-referential types

2 Returning large objects

Sometimes it is necessary to return an object with constraints that are not known at the point of the call. Typically, this is provided by a function that returns an unconstrained type:

```
type T is array
   (Positive range <>, Positive range <>) of Thing;
function Read (File_Name : in String) return T;
```

Depending on what T represents, the result may be arbitrarily large, so it must be allocated on the heap. But then there is a memory-management issue: how to return the result value and still deallocate the allocated memory.

A common solution is to return the access value, but using access types in the visible part of package specifications is poor design, and forcing the client to do the memory management, as this would, is very poor design.

We could use a smart pointer internally, which would automatically deallocate the memory after the return expression has been evaluated, but then there is still the problem of what the caller can do with such a large object, often leading to the use of access types by the client. We would like to avoid that.

A solution which avoids all these problems is to return a holder:

```
package T_Holders is new
   Ada.Containers.Indefinite_Holders
      (Element_Type => T);
procedure Read
   (File_Name : in String;
    Item : in out T_Holders.Holder);
...
declare
   type T_Ptr is access T;
   Ptr : T_Ptr := new T (...);
begin
   Item.Replace_Element (New_Element => Ptr.all);
   Free (Ptr);
   -- Operate on Item using Update_Element
exception
when others =>
   Free (Ptr);
end;
```

This involves declaring an access type, allocating the necessary space, copying that space into the holder, and freeing the access value. It is easy to get this memory management correct since the allocation and deallocation occur within a few lines. If the copy proves to have unacceptable consequences, one can create a custom holder to avoid it.

3 Making private types private

In Ada 83, the full type of a private type was only visible within the package. Ada 95 added child packages as a form of programming by extension, allowing visibility of the full type in descendant packages. While this is sometimes desirable, there are also cases when it is not [2].

Consider a large record type with constraints between various components that are difficult to enforce automatically. The package would like all modifications of objects of the type to be done by subprograms provided by the package, which ensure that the constraints hold. If the full type is visible to child packages, even the best-intentioned developer of such a child package might forget and assign directly to a component of the object, violating the constraints. It is necessary to make the type invisible to child packages to avoid this.

A question on Stack Overflow a few years ago [5] addressed just this situation. The first solution posted (by Jere) used access types, and included memory management. It is instructive to note that the memory management is incorrect, thus demonstrating yet again why access types should be avoided if possible.

My solution used holders, again:

```
private – Parent
   type Root is abstract tagged null record;
   function Equal
      (Left : in Root'Class; Right : in Root'Class)
   return Boolean is
      (Left = Right);
   package Class_Holders is new
      Ada.Containers.Indefinite_Holders
         (Element_Type => Root'Class, "=" => Equal);
   type Item is record
      Value : Class_Holders.Holder;
   end record;
end Parent;
package body Parent is
   type Real_Item is new Root with record
      Value : Boolean;
   end record;
```

The language prevents instantiating Indefinite_Holders with an incomplete private type, leading to the use of a class-wide type. The function Equal is needed because class-wide types have no primitive operations. Since the full type extends Root, it can be stored in the holder; values have to be converted to their actual type when retrieved:

```
R : Real_Item;
V : Item;
...
R.Value := True;
V.Value.Replace_Element (New_Item => R);
...
R := Real_Item (V.Value.Element);
```

4 Self-referential types

A self-referential type is a type that contains components of itself. Access types are often used to implement them; however, even here access types can be avoided.

A common example of a self-referential type is the binary tree, which contains

- A Value
- Left and Right subtrees

Conceptually

```
type Tree is record
   Value : Element;
   Left  : Tree;
   Right : Tree;
end record;
```

which is clearly not valid Ada. Again, holders can be used [1]:

```
type Root is abstract tagged null record;
function Equal
   (Left : in Root'Class; Right : in Root'Class)
return Boolean is
   (Left = Right);
package Tree_Holders is new
   Ada.Containers.Indefinite_Holders
      (Element_Type => Root'Class, "=" => Equal);
type Tree is new Root with record
   Value : Element;
   Left  : Tree_Holders.Holder;
   Right : Tree_Holders.Holder;
end record;
```

Again, the values obtained from the holders must be converted to type Tree.

All of the examples presented here have used holders, but there are cases, especially of self-referential types, for which another container would be better. For example, S-Expressions (usually shortened to SEXes) can contain a list of SEXes. Implementing that using access types would involve implementing a complete list abstraction. Far better to reuse the implementation in Ada.Containers.Indefinite_Doubly_Linked_Lists, using the techniques presented here. I present this in a draft paper [4] and on comp.lang.ada [3].

5 Summary

Access types are often used unnecessarily. Avoiding access types enhances the correctness of programs, since it prevents the errors associated with memory management. Avoiding access types is often less effort than implementing correct memory management.

References

[1] J. Carter, Binary-tree implementation, https://github.com/jrcarter/Binary_Trees.

[2] J. Carter, "Breaking the Ada Privacy Act", *Ada Letters*, Volume XVI, Number 3 [1996 May/June] [https://github.com/jrcarter/Papers].

[3] J. Carter, Reply to "Recursive algebraic data types", comp.lang.ada, https://usenet.ada-lang.io/comp.lang.ada/ p7mv5s$rig$1@dont-email.me/ [2018-03-06].

[4] J. Carter, "Self-Referential Data Types Without Access Types", https://github.com/jrcarter/Papers/ [2021-09-13].

[5] user15552120, "Hiding record from child packages", https://stackoverflow.com/questions/68838455/hiding-record-from-child-packages/ [2021-08-19]

G-NAV: Soaring the Clouds with AdaWebPack

Guillermo A. Hazebrouck
Aeronautical Engineering; email: gahazebrouck@gmail.com

Abstract

The adventure of developing a new soaring application based on Ada, WASM and WebGL.

Soaring is flying without an engine: circling on the upwards air streams and avoiding the downwards air streams. It is a tactical sport, based on meteorology, aircraft performance, navigation skills, flying skills, and confidence.

Keywords: aeronautics, open source, WebGL, Ada.

1 Presentation overview

Electronic flight instrument systems (EFIS) can effectively increase situational awareness of glider pilots by combining geographic data, aeronautical data, aircraft performance models, geolocation systems and air traffic surveillance systems.

While progressive web applications (PWA) are consolidating in the world of mobile technology, G-NAV explores these new possibilities to provide an alternative EFIS solution, carrying Ada along as the main programming language through the AdaWebPack toolchain.

With a single code base, the solution can be integrated to different systems in different ways, as it can be provided through the internet or through an onboard data acquisition system.

From a developer's perspective, G-NAV implements several particular features, like using its own vector graphics library on top of WebGL to draw shapes, lines and even text on a single canvas. This results in a vivid minimalistic interface that prioritizes awareness and robustness.

Finally, the project seeks to promote open access to a tool that can have a positive impact on safety.

2 Bio

I was born and raised in Córdoba Argentina, where I first attended a technical school and later the university. After graduating as an aeronautical engineer from the National University, I moved to Belgium where I soon started working on computer simulations for civil engineering. After five years I jumped back to aeronautics as Ada developer for the Belgian ANSP.

I love programming and gliding.

References

- https://go-gliding.app
- https://github.com/GuillermoHazebrouck/gnav
- https://github.com/GuillermoHazebrouck/gnav-web

Alire 2.0: a ‘Quality of Life’ Update

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

Abstract

Alire (from Ada Library Repository) is an open source package manager for the Ada and SPARK programming languages. Alire indexes projects in source form, but also binary toolchains ready for use from the GNAT compiler suite. Thus, starting a project with multiple dependencies and using a variety of native or cross-compilers becomes straightforward. This short paper discusses the most relevant new features in its 2.0 release, published in March 2024.

Keywords: package manager, dependency management, Ada, SPARK.

1 Introduction

Current development practices involve languages being supported by tools that simplify tasks such as dependency management, among others; that is, downloading software libraries and integrating them in the build of a larger system. In the case of open source ecosystems, such a tool is particularly important to ease the task of locating libraries scattered all over the Internet which otherwise could be hard to find and put into shape for integration.

Sometimes, such a tool is an integral part of the official language development kit, such as `cargo` for the Rust language [1]. For older languages, originated when such concepts were not ubiquitous, unofficial package managers often emerge, vying for widespread use.

Package managers can deal with dependencies in a variety of ways. In the Linux world, many distributions have a package manager as a cornerstone that typically uses pre-compiled binary libraries, suitable only for the intended environment. This kind of management, although integral to the success of open source Linux distributions, has the drawback of packages being only available on particular distributions where a maintainer has taken the time to adapt the sources to the particular distribution procedures (see, e.g., the Debian Ada policy [2]).

For that reason, language-oriented package managers try to be independent of the distribution or even operating system on which they are going to be used, to maximize the availability of their library and application catalogs.

Alire belongs to this latter class: it is portable across mainstream operating systems, providing a *community index* that points to over 400 Ada/SPARK open source projects ready for use.

In the next section, Alire is characterized in more detail. Section 3 follows by addressing the new features in Alire 2.0. Conclusions close this document in Section 4.

2 Standard features

To better characterize Alire, and contextualize its new features, its basic properties are presented first. Alire is a user-oriented package manager, intended to be used without replacing a system-provided package manager. For example, on Debian, it may leverage `apt` to reuse Debian-provided libraries, but will not interfere with it.

Alire packages are called “crates”, to avoid confusion with Ada’s package concept. A crate may contain one or more GNAT project files, and as many Ada packages as needed, although it is encouraged to use a root package with the same name as the crate. This limits the risk of name clashes and simplifies identifying a crate’s code during development.

Crates are downloaded in source form, given that compilation can be tweaked to use three main modes and any number of custom variations. These modes are: development, where switches suitable for debugging are used; validation, where switches suitable to exhaustive checking of contracts and other validity checks are enabled; and release, where compilation is optimized for performance.

Despite the main focus on sources, binaries can be distributed by Alire in cases where this is advisable. The main example is GNAT toolchains, which can be very complex and time-consuming to bootstrap and build. Alire provides native compilers for Windows, Linux and macOS. For the latter, both AMD64 and AARCH64 architectures are supported. Also available is a growing selection of cross-compilers suitable for embedded development, for targets like ARM, AVR and RISCV64.

When Alire debuted in February 2021 with its 1.0 release, it already had the core features expected from a package manager: ability to search for keywords in its repertoire of crates, automatic download and inclusion of dependencies into the build, a full solver able to find a complete solution whenever it exists, to name the main ones. Further developments within the 1.x branch enhanced the ability to work with several in-progress libraries with a flexible pinning system.

Publishing a user’s work to the community index was at the time a manual process for the most part: Alire would generate a manifest file (which provides the necessary metadata to locate sources and drive the build), but this file would have to be submitted manually through a standard GitHub pull-request.

This latter process entails a number of manual steps that can be daunting for a first-time user: forking and cloning the community index repository, committing and uploading the manifest to the appropriate location in the repository, and finally opening the pull-request.

Another feature that was missing and often requested was a way to reuse dependencies: initially, Alire offered only a sandbox mode in which every user workspace for a crate contains also all of its dependencies. While this is convenient to ensure proper isolation, the need to download and build multiple times a large number of possibly heavy dependencies could become a noticeable nuisance.

3 New features

Considering the expression 'quality of life' in the title, it will be no surprise that the main enhancements in the 2.0 release addressed these limitations just mentioned. These and other improvements are described now.

3.1 Automated publishing

Alire's publishing features now leverage the GitHub API under the hood to automate all steps previously described. Cloning, forking and pull-request creation is done by Alire. The user can then monitor the server-side integration checks from the command-line with `alr publish --status`. Assuming these checks are successful, the final approval[1] can also be requested simply with `alr publish --request-review`. This also opens the way to full automated submission of new crate releases without any user intervention (barring any failures in the integration checks).

3.2 Shared dependencies

In addition to the sandboxed mode described in Section 2, Alire 2.0 offers a new shared mode (enabled by default), in which sources of dependencies are downloaded once to a single read-only *vault* location. To avoid interferences between builds with different configuration or compiler switches, a unique hash is computed to characterize a build, and a unique build is performed within a *cache* location, common to all user projects.

In this way, as long as two projects use a dependency with the same build settings, they will reuse the build, reducing redundant source downloads and compilations.

The aforementioned hash includes recursively the hashes of all dependencies, but also all other factors that might cause a difference in the build: compiler version and target, switches and environment variables.

[1]At this time, the community index is curated, requiring final approval by a maintainer as the last step.

3.3 Lazy index loading

Another problem reported by users was long startup times of Alire's command-line tool, `alr`. This was caused by the load of the full community index on most operations, which was becoming time-consuming as the amount of releases grew, but also due to the attempts to detect system libraries described also in the index.

In Alire 2.0, release metadata is only loaded on demand, when needed by the solver or other look-up operations into the index. Not only does this feature significantly reduce the amount of information loaded for solving and searching, but in most operations that rely on a previously computed dependency solution, no index loading is needed at all.

3.4 Unicode defaults

Although the GNAT compiler provides comprehensive configuration options to tackle source and output encodings, its defaults are not in line with modern practices in which every text is considered Unicode, and often UTF-8-encoded. This caused interoperability problems in projects where several crates used different configurations, as specifications could be included in the build several times with inconsistent configurations, causing difficult-to-diagnose issues.

In what was seen as a risky but necessary move at the time, Alire 2.0 defaults to full Unicode mode, which is also fully Ada compliant. So far, no significant problems have been reported as a consequence of this breaking change in the default build options.

3.5 Nested crate detection

A concern when using a new library is the existence of examples, tests or demos that can help to understand and learn its usage. Of course, proper documentation will go a long way in this regard, but often such examples may exist and go unnoticed.

Alire 2.0 tries to help with this issue by automatically detecting nested crates within a dependency the first time one is downloaded. Any such nested crates (which are essentially projects nested within the main library of interest) are reported with their description, which may point the user towards useful resources.

4 Conclusion

Alire, the Ada/SPARK package manager, has now reached a new level of maturity with a suite of features that address unpolished or missing aspects that were detected during its 1.0 days. These include a full publishing workflow, reuse of dependency builds while preserving isolation, and a standardized Unicode configuration, besides other less impactful improvements.

References

[1] S. Klabnik and C. Nichols, *The Rust programming language*. No Starch Press, 2023.

[2] L. Brenta, "Debian policy for Ada." `https://people.debian.org/~lbrenta/debian-ada-policy.html`, 2014. Accessed: 2024-10-23.

HiRTOS: A Multi-core RTOS Written in SPARK Ada

J. Germán Rivera
Austin, TX, USA; email: jgrivera67@gmail.com

Abstract

This paper describes the design of HiRTOS *(High-Integrity RTOS), a simple real-time operating system kernel and separation kernel written in SPARK Ada. HiRTOS targets safety-critical and security-sensitive embedded software applications that run in multi-core microcontrollers.*

Keywords: RTOS, multi-core, high-integrity, Ada, SPARK.

1 Introduction

Although there are several popular RTOSes for embedded applications that run on small microcontrollers, most of them are not designed with high-integrity applications in mind, and as such are written in C, an unsafe language. So, it would be desirable to have an RTOS specifically designed for high-integrity applications, and written in a safer language, even if application code is written in C/C++.

HiRTOS (*High Integrity* RTOS) [1] is a small RTOS kernel and separation kernel for embedded single-core and multi-core platforms that have a memory protection unit (MPU). HiRTOS has a preemptive thread scheduler with fixed priorities and round-robin for threads with the same priority. On a multi-core platform, each core has its own thread scheduler and threads always execute in the same core where they were created. No HiRTOS resources are shared between CPUs and no communication/synchronization across CPU cores is supported by HiRTOS. Mutexes and condition variables [2] are the only synchronization primitives in HiRTOS, and they only support synchronizing threads running on the same core. HiRTOS mutexes support both priority inheritance and priority ceiling protocols [5]. Unlike traditional condition variables, HiRTOS condition variables can also be waited on while having interrupts disabled, not just while holding a mutex.

Given the fact that there is a separate instance of the HiRTOS scheduler per CPU, and there is no interaction between CPUs at the HiRTOS level, we just need to do formal verification of the HiRTOS thread scheduler for the single-core case.

HiRTOS is written in the SPARK subset of Ada [3], without using pointers (Ada access types). All RTOS objects such as threads, mutexes and condition variables are allocated internally by HiRTOS, from statically allocated internal object pools. These object pools are RTOS-private global arrays of the corresponding RTOS object types, sized at compile time via configuration parameters, whose values are application-specific. RTOS object handles provided to application code are just indices into these internal object arrays. No actual RTOS object pointers are exposed to application code. No dynamic allocation/deallocation of RTOS objects is supported and no static allocation of RTOS objects in memory owned by application code is supported either.

2 HiRTOS Overview

2.1 Major Design Decisions

- For API simplicity, inspired by the thread synchronization primitives of the C11 standard library [4], mutexes and condition variables are the only synchronization primitives in *HiRTOS*. Other synchronization primitives such as semaphores, event flags and message queues can be implemented on top of mutexes and condition variables.

- Unlike C11 mutexes, *HiRTOS* mutexes can change the priority of the thread owning the mutex. *HiRTOS* mutexes support both priority inheritance and priority ceiling [5].

- Unlike C11 condition variables, *HiRTOS* condition variables can also be waited on while having interrupts disabled, not just while holding a mutex.

- *HiRTOS* atomic levels can be used to disable the thread scheduler or to disable interrupts at and below a given priority or to disable all interrupts.

- In a multi-core platform, there is one *HiRTOS* instance per CPU Core. Each *HiRTOS* instance is independent of each other. No resources are shared between *HiRTOS* instances. No communication between CPU cores is supported by *HiRTOS*, so that the *HiRTOS* API can stay the same for both single-core and multi-core platforms. Inter-core communication would need to be provided outside of *HiRTOS*, using doorbell interrupts and mailboxes or shared memory, for example.

- Threads are bound to the CPU core in which they were created, for the lifetime of the thread. That is, no thread migration between CPU cores is supported.

- All RTOS objects such as threads, mutexes and condition variables are allocated internally by *HiRTOS* from statically allocated internal object pools. These object pools are just RTOS-private global arrays of the corresponding RTOS object types, sized at compile time via configuration parameters, whose values are application-specific. RTOS object handles provided to application code are just indices into these internal object arrays. No actual RTOS object pointers exposed to application code. No dynamic allocation/deallocation of RTOS objects is supported and no static allocation of RTOS objects in memory owned by application code is supported either.

- All application threads run in unprivileged mode. For each thread, the only writable memory, by default, is its own stack and global variables. Stacks of other threads are not accessible. MMIO space is only accessible to privileged code, by default. Application driver code, other than ISRs, must request access (read-only or read-write permission) to *HiRTOS* via a system call.

- Interrupt service routines (ISRs) are seen as hardware-scheduled threads that have higher priority than all software-scheduled threads. They can only be preempted by higher-priority ISRs. They cannot block waiting on mutexes or condition variables.

2.2 Separation Kernel Major Design Decisions

Besides being a fully functional RTOS, HiRTOS can be used as a separation kernel [6].

- In a multi-core platform, there is one separation kernel instance per CPU Core. Each instance is independent of each other. No resources are shared between separation kernel instances. No communication between CPU cores is supported, so that the separation kernel API can stay the same for both single-core and multi-core platforms. Inter-core communication would need to be provided outside of HiRTOS, using doorbell interrupts and mailboxes or shared memory, for example.

- Each separation-kernel instance consists of one or more partitions. A partition is a spatial and temporal separation/isolation unit on which a bare-metal or RTOS-based firmware binary runs. Each partition consists of one more disjoint address ranges covering portions of RAM and MMIO space that only that partition can access. Also, each partition has its own interrupt vector table, its own set of physical interrupts and its own global machine state. So, the firmware hosted in each partition has the illusion that it owns an entire physical machine, with is own set of of physical peripherals, dedicated memory and CPU core. The CPU core is time-sliced among the partitions running on the same separation kernel instance.

- Partitions are bound to the CPU core in which they were created. That is, no partition migration between CPU cores is supported.

- Partitions are created at boot time before starting the partition scheduler on the corresponding CPU core. Partitions cannot be destroyed or terminated.

- The separation kernel code itself runs in hypervisor privilege mode. All partitions run at a privilege lower than hypervisor mode. Partitions can communicate with the separation kernel via hypervisor calls and via traps to hypervisor mode triggered from special machine instructions such as WFI. The separation kernel can communicate with partitions, by forwarding interrupts targeted to the corresponding partition.

2.3 HiRTOS Code Architecture

To have wider adoption of an RTOS written in bare-metal Ada, providing a C/C++ programming interface is a must. Indeed, multiple interfaces or “skins” can be provided to mimic widely popular RTOSes such as FreeRTOS [7] and RTOS interfaces such as the CMSIS RTOS2 API [8]. As shown on figure 1, HiRTOS has a C/C++ interface layer that porvides a FreeRTOS skin and and a CMSIS RTOS2 skin. Both skins are implemented on top of a native C skin. The native C skin is just a thin C wrapper that consists of a C header file containing the C functions prototypes of the corresponding Ada subprograms of the SPARK Ada native interface of HiRTOS.

In addition to the C/C++ interface, HiRTOS should also provide an Ada runtime library (RTS) skin, as shown on figure 2, so that baremetal Ada applications that use Ada tasking features can run on top of HiRTOS. This can be especially useful, given the limited number of microcontroller platforms for which there is a bare-metal Ada runtime library available with the GNAT Ada compiler. an all platforms where is avaiable now or in the future.

HiRTOS has been architected to be easily portable to any multi-core microcontroller or bare metal platform for which a GNAT Ada cross compiler is available. All platform-dependent code is isolated in the HiRTOS porting layer, which provides platform-independent interfaces to the rest of the HiRTOS code. To avoid any depdendency on a platform-specific bare-metal Ada runtime library, provided by the compiler, HiRTOS sits on top of a platform-independent portable minimal Ada runtime library.

Figure 3 shows the major code components of HiRTOS. The HiRTOS code base is structured in three conceptual layers. The *HiRTOS API* layer, the *HiRTOS internals layer* and the *HiRTOS porting layer*.

The *HiRTOS API* layer contains the HiRTOS public interface components. The `HiRTOS_Interrupt_Handling` Ada package contains the services to be invoked from top-level interrupt handlers to notify HiRTOS of entering an exiting interrupt context. `HiRTOS_Memory_Protection` contains the services to protect ranges of memory and MMIO space. `HiRTOS_Thread` contains the services to create and manage threads.

The *HiRTOS internals layer* contains HiRTOS-private components that are hardware-independent.

The *HiRTOS porting layer* contains hardware-dependent components that provide hardware-independent interfaces to upper HiRTOS layers.

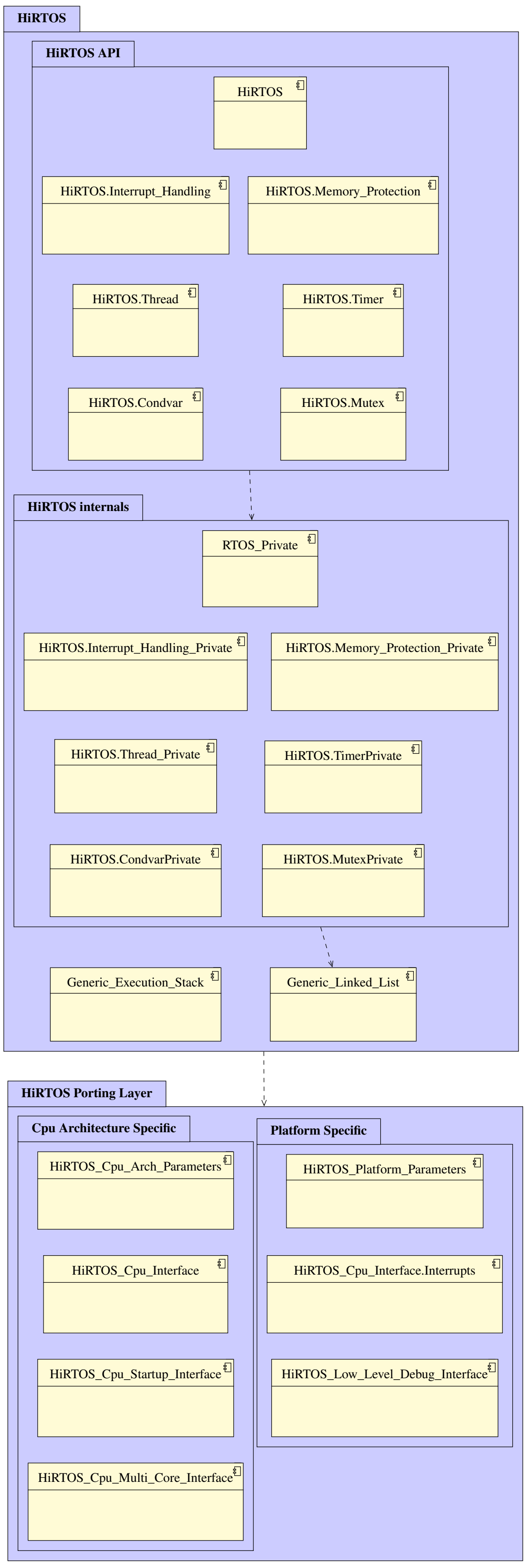


Figure 3: HiRTOS Code Components

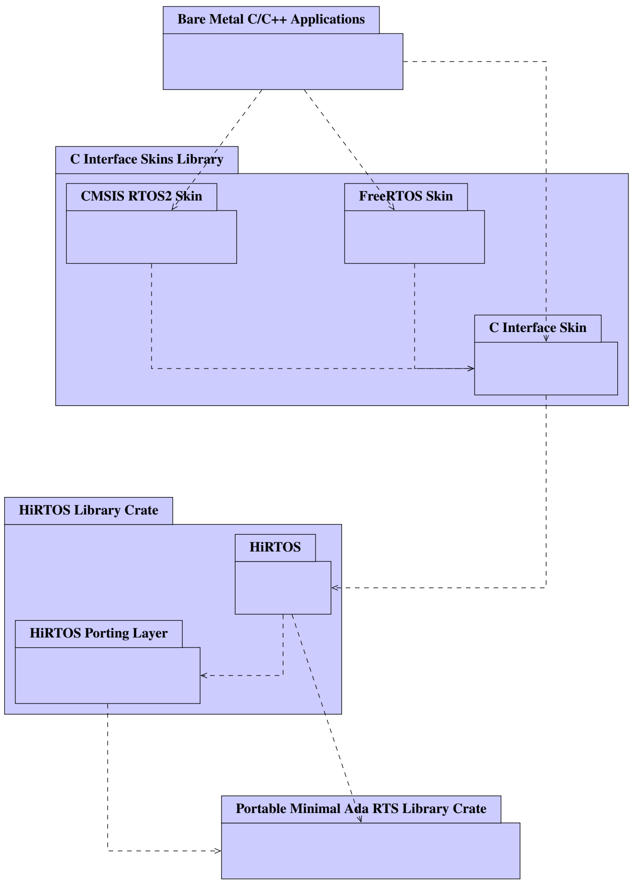

Bare Metal C/C++ Applications
C Interface Skins Library
CMSIS RTOS2 Skin
FreeRTOS Skin
C Interface Skin
HiRTOS Library Crate
HiRTOS
HiRTOS Porting Layer
Portable Minimal Ada RTS Library Crate

Figure 1: HiRTOS Code Architecture for C/C++ Applications

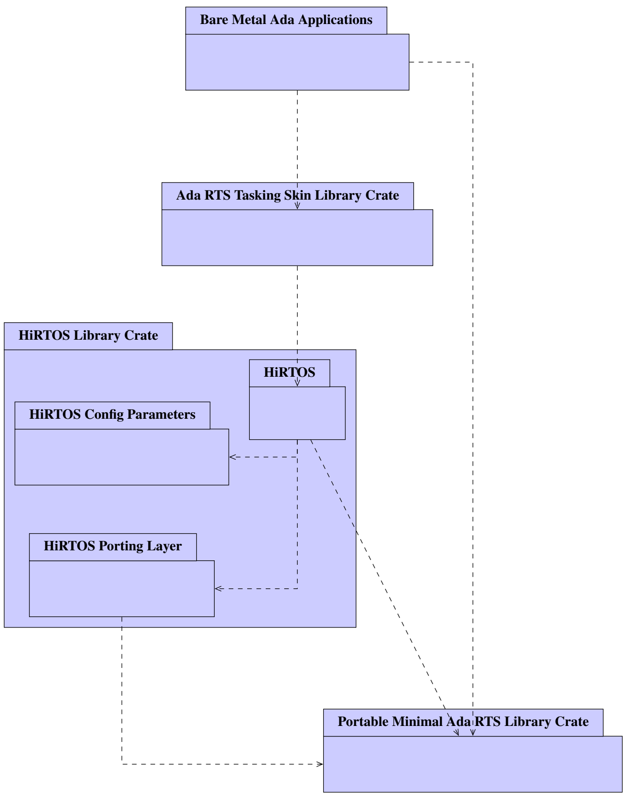

Bare Metal Ada Applications
Ada RTS Tasking Skin Library Crate
HiRTOS Library Crate
HiRTOS
HiRTOS Config Parameters
HiRTOS Porting Layer
Portable Minimal Ada RTS Library Crate

Figure 2: HiRTOS Code Architecture for Ada Applications

References

[1] J. German Rivera, “HiRTOS: a high-integrity multi-core RTOS kernel and separation kernel written in SPARK Ada” `https://github.com/jgrivera67/HiRTOS`

[2] Andrew Birrell et al, “Synchronization primitives for a multiprocessor: a formal specification”, Proceedings of the 11th Symposium on Operating System Principles, November, 1987

[3] John W. McCormick, Peter C. Chapin,“Building High Integrity Applications with SPARK”, Cambridge University Press, 2015 `https://www.amazon.com/Building-High-Integrity-Applications-SPARK/dp/1107040736`

[4] ISO, “N2731: Working draft of the C23 standard, section 7.26”, October 2021 `http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2596.pdf#page=345&zoom=100,102,113`

[5] Lui Sha et al, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization”, IEEE Transactions on Computers, September 1990 `https://www.csie.ntu.edu.tw/~r95093/papers/Priority%20Inheritance%20Protocols%20An%20Approach%20to%20Real-Time%20Synchronization.pdf`

[6] John Rushby, “Design and Verification of Secure Systems”, ACM SIGOPS Operating Systems Review, 1981 `https://www.csl.sri.com/users/rushby/papers/sosp81.pdf`

[7] FreeRTOS `https://www.freertos.org/`

[8] CMSIS-RTOS API v2 (CMSIS-RTOS2) `https://www.keil.com/pack/doc/CMSIS/RTOS2/html/group__CMSIS__RTOS.html`

Ironclad: A Formally Verified OS Kernel Written in SPARK/Ada

Cristian Simon

email: streaksu@mailbox.org; website: https://ironclad.nongnu.org, https://ironclad.cx

Abstract

Ironclad is a partially formally verified, hard real-time capable kernel for general-purpose and embedded uses, written in SPARK and Ada. This paper delves into what the kernel is capable of, what makes it special, and the future of the project.

Keywords: operating systems, scheduling, SPARK, Ada.

1 Introduction

Ironclad was started as a fully free software [1] hobbyist project to develop a kernel for powering general purpose operating systems, but, with time, broadened its scope to the field of formal verification and real-time computing. After two years of development, Ironclad is an actively developed feature rich operating system kernel capable of running complex workloads with a focus on system security and pragmatic formal verification.

Currently, Ironclad features ports to several architectures in different levels of completion, the most complete one being the x86_64 port. It is featured as the kernel of choice for Gloire [2], a fully featured operating system for general use.

2 Operating system architecture

Ironclad at its heart is a UNIX-like kernel, that means, it attempts to follow the model set forth by the original UNIX® [3] operating system family within reason.

Like so many UNIX-like kernels before it, Ironclad is of monolithic design, meaning that all the facilities of the kernel lie alone in kernel space, while the rest of userland consumes them thru syscalls and other APIs. (See Fig. 1).

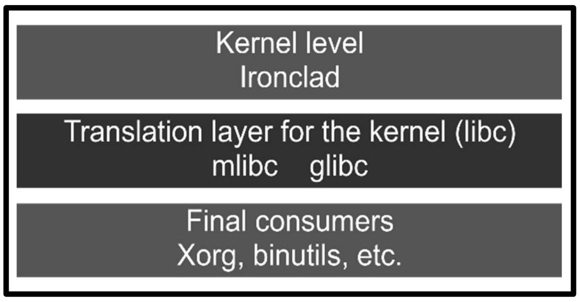


Figure 1: Kernel / Userland separation in Ironclad

The monolithic configuration of Ironclad allows it to remain performant and eases up development and deployment, while its formal verification, even if partial, mitigates in part the security concerns that arise from said design.

Ironclad does not support kernel modules at this point, but the feature might be implemented in the future, as the growing number of drivers might necessitate it.

3 Mandatory Access Control (MAC)

Ironclad has a strong focus in security, and one of the main ways the kernel has to maintain security is Mandatory Access Control (MAC), which manages access to devices and resources, like memory, and handles inheritance of permissions from process to process. MAC is always enabled, and works independently from root (UID 0) access, thus making sure that undesired root access will not be able to compromise the whole system.

3.1 Mechanism

Ironclad's MAC implementation works by imposing permission checks when accessing a protected resource, be it when allocating memory, or when accessing a path, by checking previously configured data on what resources can be accessed, on which quantity, and quality of access.

These checks are done against four types of access permission that a process can have. These are grouped in what is called a "MAC Context". A context consists of:

- Capabilities of the process: An array of broad permissions akin to Linux's capabilities [4], which allow to configure on broad strokes what a process can and cannot do, examples are: access to kernel-based entropy and clock sources, the ability to allocate shared memory, or the ability to modify the mandatory access control data associated with itself.
- Paths the process can access: MAC works by specifying which paths the process can access, any path not explicitly specified will not be accessible, that includes virtual devices.
- Resource limits for countable resources: Maximum amount of memory a process can allocate, amount of file descriptors a process can hold, among others.
- The failure policy to follow if a MAC violation is detected, options include killing the offending process, or logging the event and denying the request (the default).

MAC contexts can only be modified to lower permissions, and permission escalation is not allowed, that means, no resource limits may be raised, no filesystem paths added to allowed path lists, and no capabilities may be regained, only lost.

The MAC context associated with a process is copied from its parent at time of creation, the copy then is independent from the parent's context. Contexts and their added restrictions are inherited this way starting from to the first process of the system, which is created with a context featuring no restrictions, this starting context is denominated "zero MAC context".

3.2 MAC user interfaces

Ironclad provides a family of syscalls to interface with MAC data that allow for setting and retrieving information. If desired, these may be used to change children process permissions before handing off execution when using the UNIX syscalls fork() / exec(), by utilizing the window between them. spawn() equivalents also exist that allow for passing MAC configuration data as an argument.

The Ironclad project provides, as well, a series of userland programs, compiled under the name util-ironclad, that provide utilities to, among other things, execute programs with specific MAC settings from the command line.

4 Scheduling and real-time guarantees

Due to the generalist intentions of Ironclad, the system supports general purpose computing while providing real time facilities, this is accomplished by use of its thread scheduler, and the introduction of thread clusters.

On top of the classic thread and process, Ironclad adds a third entity, thread clusters, which bundle threads and their scheduling requirements. These bundles then can be assigned chunks of the overall system execution time, along with other settings, which will result in highly deterministic scheduling policies (See Fig. 2).

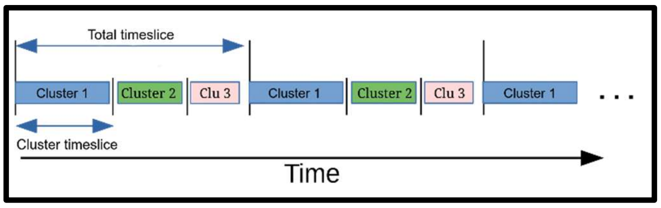


Figure 2: Scheduling of 3 thread clusters

A thread is assigned a thread cluster at creation, and may be migrated from a cluster to another at a later time.

Clusters are identified using a Thread Cluster ID (TCID). In order to provide a familiar environment for developers, at boot, Ironclad has only one cluster, which is configured to use 100% of the CPU time with "sane" configuration flags, going by TCID 1. Users may then configure that cluster, remove it, or add others, up to a system defined limit. As of writing, this limit is 10 clusters.

5 Formal verification

With formal verification, Ironclad aims to ensure correct execution of the kernel's most important components without hindering kernel development, by consuming too much development time with our current limited team. To this end, the project features a mix of verified SPARK modules with unverified plain Ada components. As time goes on, based on availability and priority, unverified modules are "promoted" to verified ones via small rewrites and modifications.

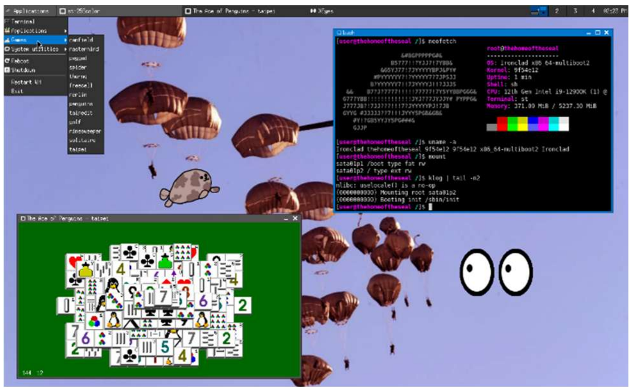
The verification process is ongoing, and currently centered around security-critical components, like Mandatory Access Control (MAC) routines, Address Space Layout Randomization (ASLR) routines, and randomization and entropy sources, to then proceed into the verification of architecture independent components, like process management, to then end at architecture-specific code.

As of writing, all security critical facilities along with a big chunk of architecture independent components are formally verified to SPARK's gold level [5], as reflected by gnatprove's automated checks, which ensures absence of runtime errors. Architecture-specific verification is scarce.

6 What is Ironclad capable of?

Ironclad is featured as the kernel of Gloire, an operating system using it alongside utilities of the GNU [6] project for its core userland, and mlibc from the managarm [7] project as its C library. Attempts to port GNU's glibc as the system's C library have been done, but have not yet been successful due to glibc's convoluted porting process.

Along with the basic GNU utilities like GCC, coreutils, and binutils, Gloire features graphical applications and environments based on Xorg, all running using Ironclad (See Fig 3).


```
[user@thehomeoftheseal /]$ neofetch
                                   root@thehomeoftheseal
                                   ---------------------
                                   OS: Ironclad x86_64-multiboot2
                                   Kernel: 9f54e12
                                   Uptime: 1 min
                                   Shell: sh
                                   CPU: 12th Gen Intel i9-12900K (1) @
                                   Terminal: st
                                   Memory: 371.89 MiB / 5237.30 MiB

[user@thehomeoftheseal /]$ uname -a
Ironclad thehomeoftheseal 9f54e12 9f54e12 x86_64-multiboot2 Ironclad
[user@thehomeoftheseal /]$ mount
sata0lp1 /boot type fat rw
sata0lp2 / type ext rw
[user@thehomeoftheseal /]$ klog | tail -n2
mlibc: uselocale() is a no-op
(0000000393) Mounting root sata0lp2
(0000000393) Booting init /sbin/init
[user@thehomeoftheseal /]$
```

Figure 3: Screenshot of Gloire as of June 2024

Gloire, just like Ironclad, is fully free software, and disk images ready for use, along with instruction on how to use them, and other relevant documentation, are available online at https://github.com/streaksu/gloire.

7 Future plans and roadmap

In the future, we expect to delve on porting Ironclad and Gloire to RISCV architectures, and improving support for the existing ones. Work on the networking department is due as well, and overall there is a lot to polish in the whole structure of the system.

The next few months of the project will see bug fixing and consolidation of new features to then start work on advanced networking hardware support and other architectures.

Ironclad's documentation, as well as download links, presentation materials, and RSS feeds to follow development, among others, are available on-line at https://ironclad.nongnu.org.

References

[1] Free Software Foundation, "What is Free Software?". https://www.gnu.org/philosophy/free-sw.html.en

[2] Gloire: An OS built with the Ironclad kernel and GNU tools, https://github.com/streaksu/gloire

[3] The Open Group, The UNIX® Standard. https://www.opengroup.org/membership/forums/platform/unix

[4] Linux's capabilities man documentation, from Linux's online documentation, https://www.man7.org/linux/man-pages/man7/capabilities.7.html

[5] AdaCore , "Applying SPARK in practice", part of the SPARK's User Guide, covering assurance levels, https://docs.adacore.com/spark2014-docs/html/ug/en/usage_scenarios.html

[6] GNU Operating System Project, "What is GNU?" https://www.gnu.org/

[7] The Managarm project's GitHub repository, https://github.com/managarm/managarm.

An Ada Story of Time

J-P. Rosen

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

Abstract

This paper is a summary of the presentation that was given at the Ada Developers Workshop during AEiC2024 in Barcelona, Spain.

It introduces the various notions covered by "time" in the context of software, and how Ada addresses the issues.

Keywords: Ada, time, time zones, UTC.

1 Introduction

All programming languages provide constructs dealing with time and durations, either as statements or as predefined libraries.

However, these notions are generally weakly defined, with statements like "Time is as returned by the operating system" or "The duration of *sleep* (*delay* or equivalent) is approximate". Of course, this is not sufficient for demanding applications, like those dealing with real-time constraints. In this paper, we present these issues, and the way Ada deals with them.

2 Some definitions

From a software point of view, *time* is something that varies (generally increasing) during computations; the value of time is given by a *time base* (aka *clock*). Note that computer time varies in jumps (not continuously).

A *duration* is a number (not necessarily an integer number) of seconds between two times.

TAI (*International Atomic Time*) is an absolute time, valid in the whole universe (not related to earth's rotation). It is the only time that makes sense if you are programming a device like a lunar probe, for example! It is measured by atomic clocks, with an extreme accuracy: drift of 1 second in 100 million years.

Since a year is exactly 365.2422 days, a day is added to the year every 4 years (with minor adjustments every 100 years, 400 years, and 4000 years). Years with an extra day are called *leap years*. Moreover, the rotation of the earth suffers minor variations due to meteorites, polar ice melting, etc. To compensate for these (unpredictable) variations, a second is added or subtracted in some years; this second is called a *leap second*.

UTC (*Coordinated Universal Time*) is a time intended to be used on earth (but anywhere on earth). It is equal to TAI + a certain number of leap seconds. Currently, UTC=TAI – 37s.

Legal time (wall-clock time, local time) is the time at a given location on earth. It depends on the location and time, due to changes between summer time (*DST*) and winter time. *Time zones* define the offset, in minutes, between the legal time at a given place and UTC. Note that local time depends on time! It is not necessarily unique for a given place: a local time may happen twice, or not at all.

CPU time is the duration spent by a task while holding the CPU. Although it is referred to as a time, it is actually more like a duration since some arbitrary time 0, often the start of the program.

3 Time types in Ada

Some types are defined in the standard as being *time types*. Some contexts require a value of a time type, but any time type can be used (more on this below).

The core language defines Ada.Calendar.Time as a time type, as returned by the function Ada.Calendar.Clock. The returned value is provided by an implementation defined time base, and its time zone is also implementation defined (it can even vary during execution when switching between summer and winter time… or by passing the border between Spain and Portugal!). The minimum variation of Time between two calls to Clock is given by the constant System.Tick.

In addition, an implementation is allowed to provide other time types. This is intended to provide access to high accuracy hardware clocks on external boards for example. Such time types can be used as replacements for Ada.Calendar.Time.

4 Ada packages dealing with time

The package Standard defines the fixed-point type Duration that corresponds to a duration. The language requires its range to cover at least 86400.0 s. (one day), with a 'Small not greater than 20ms.

As mentioned above, the package Ada.Calendar hosts the Time type (a private type) and the Clock function that returns the current time, as well as various constructors/selectors between Time and day/month/year and hours/minutes/seconds, and arithmetic functions between Time and Duration. However, the package dates back to Ada 83, and no provisions was made at the time for time zones and leap seconds, therefore the time zone is left implementation defined; friendly compilers would use the local time, but this is not guaranteed.

The situation was fixed in Ada 95 with the introduction of new packages:

- Ada.Calendar.Time_Zones defines time zones as an offset given as a number of minutes (some time zones are off by 30mn, but the seconds are always the same);

- Ada.Calendar.Arithmetic provides some arithmetic operations that handle correctly leap seconds;
- Ada.Calendar.Formatting provides not only formatting functions for Time (Image, Value, Day_Of_Week), but also operations similar to the ones in Calendar with an extra parameter Time_Offset to account for the time zone, as well as overloaded versions to handle leap seconds.

In addition, annex D (the real-time annex) offers additional packages:

- Ada.Real_Time provides a time type (also named Time) which is defined as a (non integer) number of seconds since an implementation-defined epoch, and is guaranteed to be non-decreasing (no time-zone effect). The package defines also a *private* type Time_Span (similar to Duration) and a constant Tick, with arithmetic operations between these, and conversions to/from Duration and seconds/subseconds. The child package Ada.Real_Time.Timing_Events allows triggering some actions at a given Time.
- Ada.Execution_Time defines the type CPU_Time, which is *not* a time type, with a Clock function and the usual arithmetic operations. It represents the time spent by a task while holding the CPU. Various child packages provide functionalities needed by some scheduling algorithms (Ada.Execution_Time.Timers, Ada.Execution_Time.Group_Budgets, Ada.Execution_Time.Interrupts).

5 Issues with duration

As mentioned above, the type Duration is a fixed point type. It is worth noting that a duration should never be represented as a floating point type; otherwise, as time passes, the (absolute) inaccuracy between two points in time increases, eventually being bigger than the duration of the main loop. And time would cease to increase!

Some languages deal with this issue by having a duration represented as an integer number of nanoseconds. This is of course arbitrary, in the hope that the accuracy of one nanosecond is enough… Ada is fortunate to have fixed point reals that deal appropriately with that issue.

6 The delay statements

Ada 83 had only one statement to suspend execution: the **delay** statement (now called delay-relative statement). Its argument, of type Duration, gives (in seconds) how long execution should be suspended. However, using this statement to suspend execution until a given time is subject to a race condition. If you want a task to resume at time T, you may write:

```
delay T-Clock;
```

But the following sequence of events could happen:

1. Compute T-Clock = 2s
2. After computing and before entering the delay: A higher priority task is activated
3. The task runs for 5 s. OK, it is of higher priority.
4. The first task resumes and delays for 2s., a*lthough the wake-up time has passed!*

Since there was no way to avoid this race condition, Ada 95 introduced the **delay until** statement, where the parameter is given as the wake-up time. The type of the parameter must be a time type, but it can be any time type, not necessarily Ada.Calendar.Time. Note that it is the only use of the **until** keyword in Ada! And it is sad to see that, 30 years after the problem was fixed in Ada, many languages still provide only a delay relative statement[1]…

How long will a **delay** actually sleep? Many languages just say that the actual suspension time is *approximate*, i.e. you do not know. Ada guarantees that a task is never awaken *before* the required wake-up time. In addition, implementations that conform to annex D must document the maximum time taken by a negative or null delay (a time in the past for a **delay until** statement), and the maximum lateness of a **delay** or **delay until** statement in a situation where the task has sufficient priority to preempt the processor as soon as it becomes ready. This provides an upper bound of the difference between the required and actual delay, thus allowing the computation of worst-case execution time as required by demanding real-time programs.

7 Conclusion and recommendations

Ada offers a more precise definition of time and time-related operations than other languages. However, the issue of time management in computers is tricky, and a certain number of caveats are necessary.

Use Calendar only for casual usage, since its behavior in case of changing time zones or leap seconds is implementation defined; i.e. if you just need the "wall-clock" time for display.

If you are doing serious time-related computations, then keep all times as UTC internally; use time zones for human output only. Use only operations provided by Ada.Calendar.Arithmetic and Ada.Calendar.Formatting for computations, since these have a well-defined behavior related to time zones and leap seconds.

Always use the **delay until** statement; the relative delay has been kept only for compatibility with Ada 83.

And be careful with leap seconds, and time zones, especially since daylight saving time may cause unexpected changes of time zone.

[1] Rust provides a sleep_until function, but the language reference says that is implemented using sleep, therefore presumably subject to the same race condition.

Controlled I/O: a Library for Scope-Based Files

Jeffrey R. Carter
PragmAda Software Engineering; email: jrcarter@acm.org; https://github.com/jrcarter

Abstract

This paper is a summary of the presentation at the 2024 Ada-Europe International Conference Ada Developers Workshop. A recent posting of a wishlist requested "Scope-based files". Controlled I/O is a response to that request.

Keywords: access types, memory management, Ada.

1 Introduction

A recent posting (by user pyj) [3] on the ada-lang.io forum requested "Scope-based files (controlled-type files), that close when they go out of scope". Since wrapping a File_Type in a limited-controlled type, and overriding Finalize to close the file if it is open, is trivial, I concluded that the poster would have spent the few minutes needed to implement it if that was all that was desired. Attempting to think what such a library would include in addition, I created **package** Controlled_IO [1].

2 Specification

Controlled_IO is defined by its specification:

```
with Ada.Directories;
with Interfaces;
package Controlled_IO is
  type File_Handle (<>) is tagged limited private;
  -- Opened/created on creation, closed on finalization
  -- Can be both read and written
  use type Ada.Directories.File_Kind;
  function Opened (Name : in String;
                   Form : in String := "")
  return File_Handle with
    Pre  => Ada.Directories.Exists (Name) and then
          Ada.Directories.Kind (Name) =
          Ada.Directories.Ordinary_File,
    Post => Opened'Result.Position = 1;
  -- Opens an existing file
  function Created (Name : in String;
                    Form : in String := "")
  return File_Handle with
    Post => Created'Result.Position = 1;
  -- Creates a new file named Name (deleting any
existing file named Name)
  function Opened_Or_Created (Name : in String;
                              Form : in String := "")
  return File_Handle is
    (if Ada.Directories.Exists (Name) then
       Opened (Name, Form)
     else
       Created (Name, Form) );
  function End_Of_File (File : in File_Handle)
  return Boolean;
  -- Returns File.Position > File.Size
  subtype Byte is Interfaces.Unsigned_8;
  type Byte_List is array (Positive range <>) of Byte;
  type Count_Value is mod 2 ** 64;
  subtype Position_Value is Count_Value range
    1 .. Count_Value'Last;
  function Size (File : in File_Handle)
  return Count_Value;
  -- Returns the current number of bytes in File
  procedure Set_Position (File : in out File_Handle;
                          Position : in Position_Value)
  with
    Pre => Position in 1 .. File.Size + 1;
  -- Sets the current position of File to Position
  function Position (File : in File_Handle)
  return Position_Value;
  -- Returns the current position in File
  function Next (File : in out File_Handle)
  return Byte with
    Pre => not File.End_Of_File;
  -- Returns the byte in File at the current position and
increments the current position
  procedure Read (File : in out File_Handle;
                  List :    out Byte_List)
  with
    Pre => not File.End_Of_File and then
         File.Size - File.Position + 1 >= List'Length;
  -- Calls File.Next for every Byte in List
  procedure Write
    (File : in out File_Handle; Value : in Byte);
  -- Writes Value to File at the current position and
increments the current position
  procedure Write
    (File : in out File_Handle; Value : in Byte_List);
  -- Calls File.Write for every Byte in Value
private -- Controlled_IO
...
end Controlled_IO;
```

A File_Handle is open when it exists, and closed when it ceases to exist. All files allow input, output, and seeking.

3 Children

There are two child packages of Controlled_IO: Text and UTF. Text implements text I/O similar to Ada.Text_IO. UTF implements the Universal Text File format [2] (not to be confused with Unicode Transformation Format).

4 Example/test programs

There are also three example programs.

Controlled_Test and Controlled_Text are user-unfriendly file-copy programs. Controlled_Test performs a binary copy; the output should always be identical to the input. Controlled_Text performs a line-by-line copy of text files; the output may have different line terminators than the input.

Controlled_UTF is a user-unfriendly program to convert a native text file to a Universal Text File.

5 Summary

Controlled I/O is a library for scope-based files that is somewhat more complete and complex than simply wrapping a File_Type in a limited-controlled type and overriding Finalize to close the file if it is open. Files are open when they exist, and closed when they cease to exist. All files allow input, output, and seeking.

References

[1] J. Carter, Controlled_IO implementation, [https://github.com/jrcarter/Controlled_IO].

[2] J. Carter, Universal Text File implementation, [https://github.com/jrcarter/Universal-Text-File].

[3] pyj, Ada wishlist, [https://forum.ada-lang.io/t/ada-library-wishlist/14/5].

Ada Community Advocacy

Fernando Oleo Blanco
Open Source & Ada aficionado; Tel: +34 689 44 27 45; email: irvise@irvise.xyz

Abstract

As Bob Dylan's famous song says, "The Times They Are A-Changin'", and indeed they are! However, Ada's community, in the opinion of the author, is not keeping up pace with the new ages. While the wider programming world has seen its importance grow and the number of professional programmers multiply, Ada has barely changed in the past couple of decades. The industries that it used to dominate are starting to change their preference to newer or other already proven technologies. The resources available to new programmers seeking out Ada are basically the same in style and format as they used to be forty years ago. On the other hand, the community is slowly changing and updating itself, so there is plenty to be happy about, but more remains to be done!

The following article will summarize the main ideas and goals presented in the Ada Developers Workshop in the talk of the same name as the article.

Keywords: community, sustainability, growth, Open Source, GSoC, Ada.

1 The author's point of view

The author would like to preface the entirety of this article with a disclaimer, *"I am not a programmer, less so an Ada one!"* I believe it is important for me to introduce how I found Ada and what made me stick with it, since my personal experience is likely to be completely different to that of the majority of the readers. This way, you, the reader, will be able to use your own experience and point of view with my own and hopefully create a more perfect narrative and conclusion than the one presented in this article.

I originally found Ada in 2019 after working for a few months on a `FORTRAN 66` code used for thermo-hydraulic transients in nuclear reactor cores. Working with such an old program (originally written in punch-cards!) was a wonderful and painful experience, which taught me a lot about clean and structured code, making sure that it was readable and easy to follow with the proper documentation. However, during that time, I made plenty of mistakes, introduced several bugs and I spent too much time debugging the program and trying to answer the question *"why is this happening?"* After this experience I wondered if there were better programming languages and tools which would greatly aid in the clarity and correctness of programs. I knew `C` and `Matlab` and I was aware of `C++`. All those presented great advantages over `FORTRAN 66` thanks to a better type system and better control flow than the omnipresent `GOTO`. However, I knew that it was still easy to make mistakes in those languages and create cryptic and unreadable code; so I searched for better languages. During this time I learned modern `Fortran 2008-18`, which is a wonderful language for scientific computing and presented a night-and-day change with respect to the `66` standard. And as an aside, the Fortran community had a similar article published [1] discussing similar topics as the one here.

However, I still had a lingering desire for a language without memory issues, without mixing types, with the ability to prove its correctness, with a compiler that would find errors before the program is generated, ready to do high-performance computing, etc. Languages such as Rust [2] or Coq [3], frameworks such as Frama-C [4] or projects as the `seL4` microkernel [5] started popping up quite frequently in my search. They are all incredible projects and basically ticked all the check-boxes that I was looking for. However, I also quickly found Ada, and after seeing its expressive type system and the SPARK language/prover, it had pretty much everything I was looking for and then some more!

1.1 Discovering and learning Ada

After reading some Ada examples, I decided to dive deeper and learn it a bit more systematically. So I jumped into Youtube and searched for some Ada tutorials... I was very negatively surprised that there were basically only two creators making videos about Ada. One is the well-known and reveered AdaCore, the other one was Patrick Kelly. I used Patrick's videos and tutorials in order to understand the language more. After watching them, I wanted to know even more, so I started searching for more information about Ada in a search engine. As I prefer videos for learning computer skills, I quickly found and watched Jean-Pierre Rosen's FOSDEM videos, such as his excellent *An Introduction to Ada for Beginning and Experienced Programmers*. Finally I also watched AdaCore's videos and talks about various topics. All of these videos and the learning experience were enjoyable and helped me greatly in discovering how truly amazing the Ada language is. Though I must also say that I watched videos on `C++`, such as the ones uploaded to CppCon and plenty of Rust tutorials; which were very informative and useful for general programming knowledge.

2 Great features... and lack of success?

If any Ada programmer is asked about why Ada is so great and why should people use it more, the reply would go something like:

- Readable and easy to learn syntax! It is quite similar to that of Pascal, Python or Lua.
- Great type system! Newer languages are realizing the value of types, take a look at TypeScript, modern Python 3 or even PHP! And they barely scratch the surface of what Ada is capable of!
- Great module system!
- Amazing features such as tasking, contract, C-interop and we can even target WASM!
- It is mature, well understood, documented and standardized.
- SPARK.

And the author fully agrees with this assessment. But then, the obvious question becomes... *"If Ada is so good, why isn't it used more widely and why doesn't it get the same amount of praise as other do?"*

2.1 Success in theory and in practice

When analyzing a given object, product, task, project, etc; and how successful it should be, we tend to think about properties such as quality, low-cost, readiness, ease-of-use, features... The better the analyzed system fairs within these aspects, the more successful it should be. Applying this framework to Ada, it is obvious that Ada has to be one of the greatest languages available, back in the 80s as in today's environment. However, that does not seem to be the case in real life...

In thermodynamics, specially in the thermal-energy sector (think of coal, gas and nuclear), there is the concept of ideal thermodynamic performance and the real one. The ideal performance comes from Carnot's equation, but the real one is a lower value that is obtained from multiplying Carnot's equation with an efficiency factor. This efficiency factor represents the reality of the system, its inefficiencies, its limits and compromises. It is the opinion of the author that while Ada's theoretical success should be high, it does not translate to the real world due to a low "efficiency factor", just like in the example given above.

It is the opinion of the author that Ada has some major drawbacks that lower its impact in the wider programming world, even if these are not related to any technical subject. Here are some of the points, that the author believes, lower Ada's efficiency to translate its strengths into real results:

- Ada is less profitable for people to learn. This boils down to Ada having less job positions available, and quite a few professionals focus on investing time on technologies that will yield some kind of return.
  - Newer languages have also suffered from this issue, but continued growth and the commitment of companies to use them, has allowed for the creation of new job positions.
- Lack of momentum/trend. People tend to think something new is better than the old, which is a really good rule of thumb. After all, this is the very definition of human progress. However, as we know, this is not always true, and Ada suffers this fate of being considered old by the general public.
- Lack of monopoly. Ada has no longer a field in which it is the only language that can be used. This is in contrast to Java/Kotlin being the de-facto languages in Android, Swift in the Apple ecosystem, TypeScript/JS in the web, Python for LLMs/AI/scientific computing, etc.
- Ada is less enjoyable or "smart", as it focuses on getting stuff done with a good level of quality. Quite a few programmers focus on personal enjoyment rather than project results when considering a new language to learn.
- Lack of (new) learning resources and ease of access. As the author indicated in the introduction, there are few Youtube videos, which are now a common form of learning for the younger people. Blogs are nowadays another common way of learning. Up until recently, the main discussion forum was still in Usenet, and once again, such system is no longer used by the newer generations.
  - Another issue is that some documentation has become slightly obsolete with the appearance of newer standards, tools such as Alire or libraries and bindings. This does not help with easily finding good and up-to-date information. On top of that, a wonderful resource, the Ada Reference Manual ((A)RM), is quite daunting to new programmers as they are not used to reading standards, mostly because other languages do not have them readily available.
- A sense of belonging to a community. Finally, the Ada community is quite small and it is sometimes difficult to interact with it due to differences in expertise or the platform. The conversations and topics that take place are also sometimes self-defeating, such as discussing how Ada does something better than Rust. This just creates an echo chamber and does not propel Ada in a productive direction.

3 Community

The author won't define what the word *community* means in a technical manner. Instead, I want to describe what *kind* of community I am referring to and will discuss in the rest of the article. When the Ada community is brought up, it is tempting to think about companies such as PTC or AdaCore which provide tools to work with Ada. We could also think about users of Ada, such as the US military, Eurocontrol and Codelabs. Or organizations involved with the design of Ada or its members such as WG9 or Ada-Europe. However, none of these groups are what I refer here as the wider Ada community. The community I will be talking about are the general users of Ada and people who interact in informal, everyday platforms. Those people who discuss Ada, share knowledge and are independent with each other. Of course, the companies and groups that I have specifically mentioned before are also part of this wider informal community and they play a very important role there, but I will leave this point to the side for the rest of the article.

3.1 Community as a reflection of success

It is the opinion of the author that the community of a programming language is a reflection of the efficiency and efficacy with which it is able to bring its strengths to the rest of the world. The quality of the community serves as an indirect measurement of the health of the language.

Luckily, the past few years have seen a huge improvement of the Ada community. The creation of Learn.AdaCore has greatly improved the learning experience for new users. AdaLang has modernized the first impression that newcomers to Ada have. It has also provided a modern forum for discussions. There are now group chats in Element, Discord and Telegram. Alire has greatly simplified the discovery of Ada libraries and the installation of a toolchain. The creation of Ada_Language_Server and the support in Visual Studio Code have also brought Ada closer to the mainstream trends and expectations. Nonetheless, the author believes we, the community, can do better in a few areas.

- Focus on the needs of Ada instead of on its already achieved goals. It is quite common to see discussions laying out how Ada is much better than Rust or other languages in one or another way. However, this discussions do very little to improve Ada and its ecosystem. A much healthier approach is to acknowledge what other communities and technologies get right and try an implement those in Ada. Alire is a great example of this.
- Spread the word of Ada in a kind and reasonable manner. Talking about Ada to other Ada programmers is not going to help it grow much. As indicated above, the Ada community knows about other languages, but the opposite is not true. We should bring Ada's nature and knowledge to other people who are outside of Ada circles and may be interested in it. We also have to do so with kindness and without arrogance. We should also try to use the same language that other use and like. For example, in the Rust world, the expression "blazingly fast" is used often to define a fast language. We are as fast as them, so lets use those expressions and selling points too! We should also use generic conferences to try to bring attention to the language and some of the projects that are written in it.
- Help other member of the community, specially when they are alone. As an example, the GNAT compiler for OSX is mostly maintained by a single person, Simon J. Wright. This does not exemplify an ecosystem and community that acknowledges the work that others do and tries to be sustainable. We need to get to work and help each other.
- Encourage younger people to join. The Ada community is filled with knowledgeable and seasoned programmers, but a young and thriving community ensures that it is going to be long lasting and that knowledge is being transmitted. And I would say that a few young people would be very pleased to learn Ada due to its unique set of features.
- Use the opportunities that are available, specially funding. There are programs to get funding which could be utilized by the Ada community to improve its tooling, libraries or major projects. For example, the Ironclad project was recently given a grant by the NLnet foundation. The author has previously talked about the opportunities that the GSoC program presents to open source projects and how Ada could make use of it.

4 Path forward

The list above does not form an exhaustive set of tasks, but a general road-map and set of goals. However, they should serve to tackle the list of shortcomings which was indicated in the previous section. Moreover, this list is not meant to just be a list. The reader may have noticed that all of them are actions, they require a proactive exercise from within the community. The author would also like to point that the desired result of all of these points, to increase the "efficiency", is not a quick process. It takes years to build and foster a thriving community. So lets have patience, energy, a big smile and keep on moving! The author believes there is an important place for Ada/SPARK in the wider ecosystem of languages and people are waiting to use it and build impressive products!

5 Citations and references

References

[1] M. Curcic, O. Čertík, B. Richardson, S. Ehlert, L. Kedward, A. Markus, I. Pribec, and J. Vandenplas, "Toward modern fortran tooling and a thriving developer community," 2021.

[2] S. Klabnik and C. Nichols, *The Rust Programming Language*. USA: No Starch Press, 2018.

[3] T. C. D. Team, "The coq proof assistant," 2024.

[4] L. Correnson, P. Cuoq, F. Kirchner, A. Maroneze, V. Prevosto, A. Puccetti, J. Signoles, and B. Yakobowski, *Frama-C User Manual*.

[5] A. Lyons, K. McLeod, A. Danis, G. Klein, yyshen, A. Heider, C. Millar, S. Sherratt, R. Kolanski, L. Prion, S. Shields, M. Brecknell, J. Beeren, I. Velickovic, J. Brush, I. Zupancic, C. Guikema, E. Pierzchalski, S. Zhuang, N. Spinale, A. Zarrabi, sorear, S. Gauthier, matt rice, branden data61, MattPhillips1, A. Felizzi, michaelmcinerney, A. Boettcher, and J. Millwood, "sel4/sel4: sel4 13.0.0," 2024.

Formal Verification of Safety Critical Software in Ada: Two Approaches

Ranjani Krishnan

Vikram Sarabhai Space Centre, Trivandrum, Kerala, 695022, India.; email: ranjani141@gmail.com

Ashutosh Gupta

IIT Bombay, Powai, Mumbai, Maharashtra, 400076, India.; email: akg@iitb.ac.in

Nitin Chandrachoodan, Lalithambika VR

IIT Madras, Chennai, Tamil Nadu, 600036, India; email: nitin@ee.iitm.ac.in, vrlalithambika@gmail.com

Abstract

The verification of real-time, embedded software in complex, safety critical systems such as crewed space launch vehicles is as significant as the design. Approaches based on formal methods are necessary to ensure exhaustive validation, in addition to the traditional testing and simulation techniques. In this work, the application of software model checking and static analysis in the verification and validation of safety critical software in Ada is explored. With the embedded flight control software in the onboard computer of an aerospace system as the case study, we apply the SPIN model checker and also develop a custom tool chain based on bounded model checking, for formal static analysis of Ada code. The major contributions include the definition of a systematic procedure for model checking with SPIN tool and the development of a new verification framework for formal static analysis of Ada programs. The two approaches are applied to an actual case study through accurate modelling of the execution environment for the concurrent onboard software in a launch vehicle. The results are compared and the advantages and drawbacks of both approaches are summarized.

Keywords: Ada, Safety-critical systems, Formal verification.

1 Introduction

Safety critical systems are those in which failures could lead to loss of human lives, damage to the environment or property. Aircraft, cars, medical devices, nuclear power plants and space systems are examples of such systems. These are large, distributed systems with subsystems developed by different teams. The embedded software in these systems performs critical tasks, typically collecting necessary inputs with various kinds of sensors and generating required outputs to control actuators such as valves, motors etc. It has to meet strict timing deadlines also. The safety requirements for such complex systems are much more than the functional requirements. The redundancy in the systems and very high reliability requirements call for thorough testing and validation, of both the hardware and software. Often, a single changed requirement could necessitate a complete re-verification of the entire project as it could introduce a new error. Hence, for safety-critical systems, both the design and testing are equally important.

The launch vehicle is a safety critical system which is used for injecting satellites into the required orbits as well as for transporting humans to space, with such manned flights becoming more frequent now. In a launch vehicle, the flight software in the onboard computer (OBC) receives attitude, rate and acceleration data through gyros and accelerometers interfaced to navigation computers and fuel tank pressure data from pressure sensors. The OBC communicates with redundant input systems and selects the data based on certain validation criteria. These data are provided to the various software modules such as navigation, rate data processing, digital autopilot, guidance, sequencing, and fuel tank vent algorithm. The Real Time OS (RTOS) in the OBC carries out these data acquisition and interfacing functions as well as the scheduling of the various tasks at the specified time with the required periodicity. The OBC then distributes the outputs - control and sequencing commands – to the control and sequencer systems, which are the output modules. Important parameters and internal states of each software component are monitored through the vehicle telemetry system to assess the system performance.

The software in aerospace systems has been growing exponentially in terms of volume of computations and complexity. Verification of the real-time, embedded software in these systems is therefore as significant as the design. There have been several instances of software bugs causing mission failures. The failure of the first Ariane5 launch [1] in 1996 was attributed to a software error. The software exception, viz. overflow, was caused during the data type conversion from a 64-bit floating point data to a 16-bit signed integer. The Therac-25 radiation therapy machine malfunction [2] was caused due to a race condition. The Patriot missile failure [3] during the first Gulf War was a consequence of a software rounding error that led to incorrect calculation of the time. The Mars climate orbiter crash [4] was due to an error in the thruster control software. The unit used for computation of force was pounds, instead of Newtons, which was the actual NASA specification.

As illustrated by the above examples, the design and verification of onboard, real-time software in aerospace systems like rockets is significantly more intricate compared to the software in desktop systems, user applications or other IT systems. This is mainly due to the interplay of multiple concurrent processes and the inherent complexity of fault-tolerant systems. As the time and effort required for validation of software is directly proportional to its complexity, ensuring the correctness of the code in safety critical systems is extremely difficult. It is also important that verification methods be able to support the increasing size and complexity of the software in such systems.

The traditional techniques employed for validation of embedded software include testing, analysis, review, inspection and simulation. But with these methods, the main drawback is that all combinations of input values cannot be applied to the system to ensure complete verification: they are incomplete and time-consuming. Also, these cannot detect faults early, during the requirements or design phase and cannot be fully automated. Inspecting the code manually is prone to human errors as well as risky, as undetected bugs can lead to failure of the mission or even loss of lives. Hence, for safety-critical systems, it is clear that conventional techniques of verification are inadequate.

Since even a single fault in the software can have catastrophic consequences including mission failure and damage to life, thorough validation of the launch vehicle OBC software is a must. It is tested at different levels such as designer level testing, code inspection, module testing and integrated testing which are all carried out manually by an independent QA team. Numerous simulation tests at subsystem and system levels are also conducted to validate the software. But there is very little automated verification with tools at present. Tool support is essential for testing to provide a guarantee of software quality. To ensure exhaustive validation, approaches based on formal methods are necessary. Certification standards [5,6] for safety critical software also prescribe the application of formal verification. It involves proving that the program semantics (events during the actual program executions) meets its specification (how the program is expected to execute) using mathematical methods.

Advances in tools have made formal verification practical and different methods are being adopted for its application to embedded software. Theorem proving, model checking and static analysis are some of the popular formal techniques. Theorem proving or deductive verification [7] is a powerful method which raises few false alarms; but it demands a high degree of expertise and interaction from the users, for example, by writing contracts for functions. Verification techniques relying on abstract interpretation [8] are used in several tools like the software bounded model checker CBMC [23] and Polyspace [9]; however, such tools generate too many false alarms and the user needs to spend considerable time and effort to identify the actual bugs. Software model checking is an attractive alternative that is automated, simple to understand and learn and can be applied for verification of assertions and temporal properties in various types of software. Verification based on bounded model checking [10] carries out the analysis on a smaller set of program traces with bounded length; hence it overcomes the disadvantages of state explosion and long verification time. It does not require much intervention from the user and scales for large systems, without compromising significantly on accuracy.

In this work, the verification and validation of safety critical software in Ada by applying formal methods to complement traditional testing and simulation is explored. Specifically, software model checking and static analysis are used in this work. With the embedded flight control software in the onboard computer of an aerospace system as the case study, we apply the SPIN model checker [11] and also a custom tool chain developed for formal static analysis of Ada code. The primary goals of this work are to: 1) Formally verify the embedded software in a launch vehicle application using two different approaches 2) Define and develop a systematic procedure for modelling and verification of Ada software 3) Evaluate the challenges involved in both approaches 4) Examine the feasibility of practical application of these methods in the software development lifecycle process.

The major contributions of this work are the following: 1) Definition of a systematic procedure for model checking with SPIN tool incorporating translation of Ada source code to Promela language 2) Application of a verification framework based on bounded model checking for formal static analysis of Ada programs with a pre-defined priority for the tasks 3) Definition of a methodical process to translate functional requirements of a software module into specifications for verification.

The rest of this paper is organized as follows: the related work in formal verification of different safety critical systems using varied approaches is summarized in Section II. The launch vehicle software organization is described in Section III. The overall approach followed for our verification, with model checking and static analysis are explained in Sections IV and V. Section VI describes the experiments carried out using the two techniques and the results and also discusses the merits and demerits of both. Section VII comprises the conclusion and the possible course of future work in this area.

2 Related Work

In [12], the authors describe the application of the Frama-C tool for verification of the embedded control software in a Brazilian launch vehicle. The Frama-C tool has different plug-ins: the value analysis plug-in enables static analysis of software based on the principle of abstract interpretation while the Jessie plug-in permits deductive verification of C programs annotated using the ANSI C specification language. Reference [13] explains the application of the ACL2 theorem prover in various industrial projects for the formal specification of microprocessors, proofs of hardware design models, verification of android apps and several others. The use of formal methods for demonstrating that a software unit serving as a user interface complies with user specifications is elaborated in [14]. A case study from the medical field, a commercial infusion pump is analysed using the PVS theorem prover tool. In [15], the formal specification of a flight control software in Ada using Larch and its formal verification using Penelope, an interactive verification environment based on theorem proving, are attempted by the authors. Reference [16] elaborates on specifying and verifying the requirements of a memory management system using the proof assistant Coq. This method is applied for the verification of an algorithm in a safety-critical embedded OS. The authors report on their use of the Dafny theorem prover to formally verify an algorithm for grasping a spacecraft motor nozzle, part of an autonomous space robotics software in [17]. All of the above are applications of different theorem proving tools to critical systems.

Considering static analysis, in [18], the design of ASTREE, a static analysis tool for C programs, which works on the principle of abstract interpretation is described. It was applied successfully to safety-critical, large sized, embedded control, real-time software in aircrafts for detection of run-time errors. Reference [19] elaborates the application of two formal tools, ASTREE and FLUCTUAT, for analyzing the onboard software of ATV space vehicle in C. These tools are suitable for C programs, but cannot be used for other languages like C++ or Ada. The application of different formal verification techniques, such as theorem proving, static analysis and model checking to automotive embedded software in C is described in [20]. ASTREE and Polyspace (Abstract interpretation-based static analysis), SCADE design verifier (model checking), Frama-C and SPARK (deductive techniques) are the tools employed by the authors. Reference [21] explains the verification of the onboard software in Ariane 5 launch vehicle of ESA manually, as well as using formal static analysis. The real-time kernel ARTK was checked manually for dead code, exceptions handling, dynamic allocation and critical sections. The INRIA static analysis tool was used to check whether all variables are properly initialized, shared data are protected from simultaneous access and other possible errors that could occur at run time. The development of a formal static analysis tool and its application to the software in Programmable Logic Controllers (PLCs) at CERN is discussed in [22].

Reference [23] presents the CBMC tool for the formal verification of ANSI-C programs, based on the principle of bounded model checking. In [24], various formal techniques, such as equivalence and inductive proof methods are combined with bounded model checking using Systerel Smart Solver toolset and used to verify the anti-collision system software onboard a rover. A Counter Example Guided Abstraction Refinement (CEGAR)-based static analysis tool named SDMC for C programs is presented in [25]. The authors explain an algorithm combining k-induction along with invariant inference and evaluate it through verification of the software in unmanned aerial vehicles (UAV) in [26]. These are instances of applying various bounded model checkers for software verification.

Finally, there have been multiple attempts at modeling and verifying critical software using diverse model checking approaches. In [27] the application of UPPAAL, a real-time model checker, for validation of high integrity software developed in Ada95, is discussed. The authors develop a formal definition of a Ravenscar compliant run-time kernel, as the Ravenscar Ada subset [28] supports tasking also. The authors present the specification and model checking of the mode logic in the flight control system of an aircraft using Simulink and NusMV model checker in [29]. Reference [30] describes the development of CASE tools to support design, formal verification and analysis of the digital control system for a nuclear reactor. In [31], the redundancy management system (RMS) in VentureStar rocket is formally verified using the PARAGON toolset suitable for real time systems. Reference [32] explains a systematic method for modeling and formal verification of software case studies in nuclear systems, using the NuSMV and UPPAAL model checkers. The specification of requirements and the formal verification of the level-crossing control system for trains are carried out by the authors in [33] using the mCRL2 tool.

In [34], the NuSMV2 model checker is applied for symbolic model checking of the embedded Ada software in the attitude and orbit control system (AOCS) of a spacecraft. A method for mapping the model of a real-time tasking system that follows the Ravenscar profile, by translating AADL models to LNT specifications and its verification with CADP toolbox is presented in [35]. This technique is applied to a line follower robot, as well as a flight control system, for illustration. Reference [36] proposes a method using timed automata networks to model concurrent, real-time systems and formally verify them using SMV model checker, with the automatic train protection system as a case study. Analysis methods for validation of timing parameters related to safety and security for automotive systems are proposed in [37]. Verification is performed by applying UPPAAL-SMC to a case study in the automotive domain considering various possible attack scenarios. In [38], the authors propose a new protocol, MQTT-CV and validate it formally through SPIN model checker, before testing it on real-world connected cars. A tool named ATOS, that automatically extracts a SPIN model from Ada programs, along with correctness properties is explained in [53]. In [54], the authors describe the design of a bounded model checker for SPARK code.

3 Launch Vehicle Onboard Software

The flight software in the onboard computer (OBC) plays a crucial role in the functioning of the launch vehicle by carrying out various activities like computing the quaternions from the sensed position and velocity, calculating the manoeuvres to steer the vehicle along the optimum trajectory, computing the control commands through the autopilot algorithm, generation of various sequencing commands such as ignition and separation and maintaining the pressures of fuel tanks at required levels. Thus, the software consists of different components like navigation, guidance, digital autopilot, sequencing and fuel tank vent algorithm. To interface the varied software

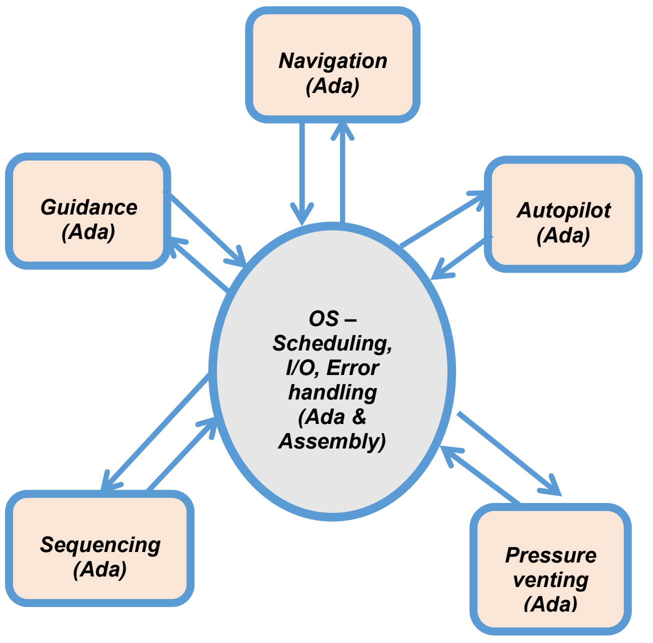


Figure 1: Launch vehicle software organization

components, to manage the hardware resources and also to schedule these tasks as per requirements, a system software is essential. An indigenously developed, 16-bit processor is used as the OBC in the rocket. The embedded software in the onboard computer is developed partly in Ada83 and partly in the assembly language of the indigenous processor. The software components in the OBC are represented in Fig.1 and the inputs/outputs of these are listed in Table 1.

The navigation computer acquires the digital and analog data from gyros and accelerometers, carries out necessary pre-processing and sends the data, along with sensor health status, in real time to the OBC. The navigation software processes this data to generate the outputs for use by other application modules like autopilot, sequencing and guidance. The rate data processing software carries out sensor error compensation, failure detection and isolation and data selection for posting to autopilot. The function of guidance software is to steer the vehicle along the most optimal trajectory from lift-off till satellite injection. The autopilot software carries out control of the vehicle attitude to track the steering commands and ensure collision-free stage separation events. The sequencing software detects the critical flight events, which are triggered by specified dynamic conditions in real time, such as acceleration, lift-off signal and guidance flags within pre-determined time windows. Control of the fuel tank pressures for maintaining it within specified limits is the main function of fuel tank vent algorithm software.

For seamless interfacing of the different OBC software components and to interact with the hardware, a supervisor software is essential. The RTOS software in the OBC is responsible for this. Synchronized operation of all onboard computers with respect to their clocks till the end of the mission is ensured by the RTOS. It collects the inputs, executes the tasks and posts the outputs to the execution modules periodically by scheduling the tasks in real time. It collects the data to be monitored and packs it for telemetry. It services the hardware interrupts, handles the hardware and software errors occurring during the flight and assesses the health of all subsystems in the rocket periodically.

There are tasks running at two periodicities in the embedded software - minor cycle (typically 20ms) tasks and major cycle (typical value 500ms) tasks. The minor cycle tasks follow a pre-defined, static, sequential scheduling and they are executed one after another without pre-emption. In every minor cycle period, after all the minor cycle tasks are completed, the major cycle tasks are scheduled. At the end of every 20 ms cycle, a timer interrupt occurs. Then, the major task is interrupted, its state is saved and the next minor cycle begins. Again, the major task is resumed by restoring its context subsequent to completion of all the minor tasks. This is referred to as the rate monotonic scheduling scheme, in which tasks running at higher frequency always have higher priority than those executing at lower frequencies.

Table 1: Inputs and outputs to different software

Software	Inputs	Outputs
Navigation	Angle and velocity increments, analog rates and acceleration	Quaternions, body rates, velocity, position and orbital parameters
Guidance	Altitude, position, velocity and stage information flags	Commanded quaternions
Autopilot	Attitude, body rates, commanded quaternions	Control commands
Sequencing	Vehicle acceleration, flight time and fuel tank valve status	Sequencing commands
Fuel tank vent algorithm	Commands to start the algorithm and fuel tank pressures	Commands for close/open of valves

4 Verification Approach with SPIN Model Checker

4.1 SPIN Model Checker

SPIN [11] is a model checking tool that is highly popular for formal representation and verification of communication protocols and embedded software. The models are developed in the Process Meta Language (Promela) in this tool. This input language enables the construction of simple system models through three fundamental constituents: processes (asynchronous), channels for messages and global data. Data types that are supported include bit, bool, byte, short and int. The user is permitted to define other structured data types combining the basic ones. It is also possible to embed a restricted subset of C code directly in the model.

For modelling in SPIN, the system should be closed, that is, all possible inputs to the system should be defined in the model, usually done by approximating the input behaviour in a conservative manner. This environment model serves as a test driver for the verification with SPIN. SPIN generates the state vector for the system consisting of all global and local variables, message channels and processes. The statements are interleaved and executed asynchronously. Depending on the process that is selected for execution from the threads that are executable, several different interleavings of process statements are possible. An exhaustive exploration of the state space is extremely complex and time-consuming. To limit the time and memory required for storage of state space, SPIN follows various methods and approximations, including partial order reduction and on-the-fly generation of a minimal recognizer for reachable states.

Given an input model in Promela, SPIN can check for safety and liveness properties like presence of dead code, deadlock, violation of assertions specified by users and Linear Temporal Logic (LTL) formulas representing temporal properties of the system. If a particular property is violated, it gives the error trace for the counterexample, a sequence of transitions that violate the specification. SPIN also has a provision to automatically extract the model from C source code using the FEAVER model extractor [39]. Though SPIN model checker has been successfully used for modelling and verification of numerous satellite systems at NASA [39-43], literature on verification of launch vehicle software with SPIN is not available. Considering the usage history of this tool for aerospace systems, it was chosen for the first part of our work.

4.2 Formal Verification with the SPIN Model Checker

The overall approach employed for the verification of embedded software through model checking is shown in Fig.2. In the context of Indian launch vehicles, the onboard software is developed partly in Ada [44] language and partly in the assembly language of the indigenous processor used as the onboard computer. As there are different components in the software, we focus on specific portions, namely the scheduler, the tank pressure vent algorithm, input and output data processing, clock synchronization module and the 1553B communication. Since the scheduler is developed in the assembly language of the indigenous processor, no existing tools can support its verification and the process is completely manual at present. The modules like tank pressure vent algorithm, the input and output data processing and clock synchronization module are written in Ada. For model checking, the assembly code and the Ada programs were translated into Promela with abstractions and the models were fed as input to SPIN. The safety and liveness properties were derived from the software documentation such as the requirements document and specified in LTL.

The modeling, verification and results of the validation of the scheduler in assembly using SPIN were described in a prior work carried out by us [47]. Here, we focus on our further work: modeling of launch vehicle software components in Ada using Promela and our new tool LLVMBMC and verification of specifications extracted from requirements. The individual components which were

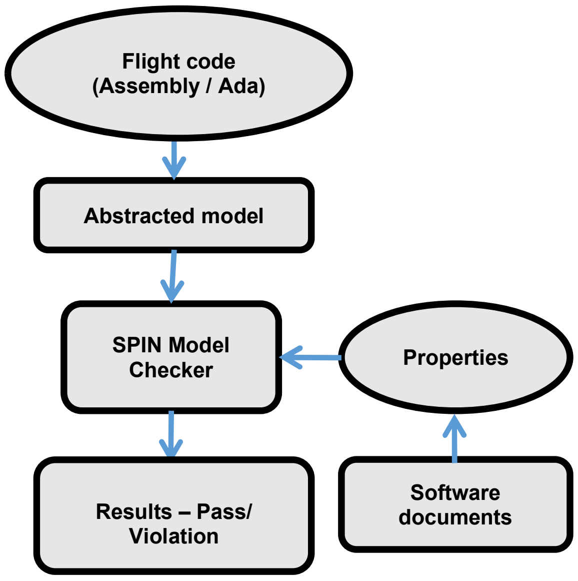


Figure 2: Overall approach with SPIN model checker

verified using one or both methods include input data acquisition module, navigation, guidance, fuel tank vent algorithm, rate data processing, clock synchronization algorithm and output data posting module.

In addition to application of the SPIN model checker for verifying critical software, the process involved a methodical sequence of activities. For the software modules in Ada, a systematic procedure was defined for manual translation of Ada source code to Promela. As shown in Fig. 3, each procedure in the source program was translated into an inline function in SPIN. For global variables, we used variables with simplifications to reduce the state space in

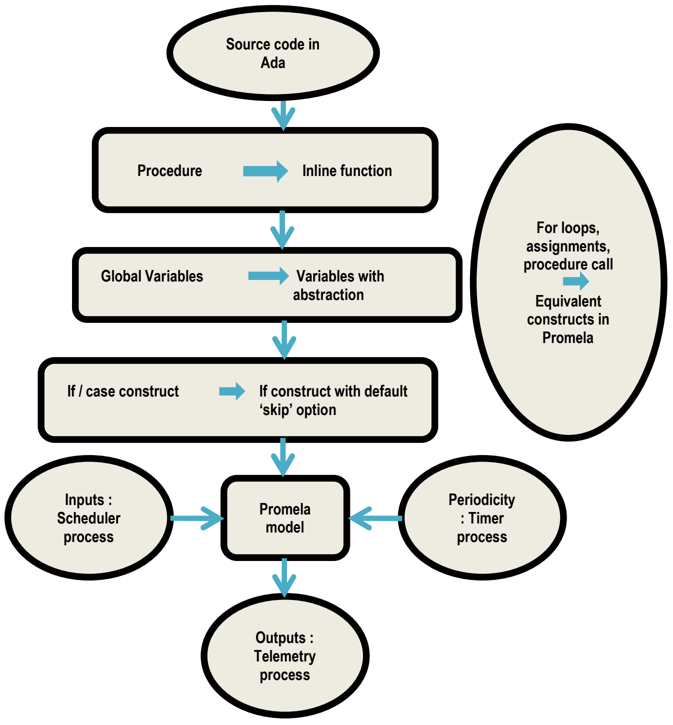

Source code in Ada
Procedure
Inline function
Global Variables
Variables with abstraction
If / case construct
If construct with default 'skip' option
For loops, assignments, procedure call
Equivalent constructs in Promela
Inputs : Scheduler process
Promela model
Periodicity : Timer process
Outputs : Telemetry process

Figure 3: Translation from Ada source code to Promela

Promela. For if/ case statements, for loops, assignments and procedure calls, equivalent constructs in Promela were employed. The execution environment for the high level software components was simulated using a driver scheduler process which supplies the inputs (done by the RTOS in the flight software) and the minimal hardware behaviour (timer and interrupt) was modelled using a timer process. The gathering of outputs, as done by the RTOS, was simulated by means of a telemetry process. A similar procedure can be adopted by other design and testing teams developing safety critical, embedded software for translation to Promela.

Abstraction of the actual system is necessary to enable verification with SPIN. This is a challenging process, which involves derivation of a system with a finite state space from a large program, by simplifying the states that are not germane to the operation we are interested in. It is done through various methods, including removal of code that is irrelevant to the property being verified, substituting infinite data types with finite intervals, restricting the count of tasks executing simultaneously etc. A good understanding of the source program is necessary to perform good simplifications. Separate validation of independent functional requirements is also a kind of reduction. Some of the major simplifications that were applied are listed in Table 2. The portions of the code, independent of and irrelevant to the parts under verification were removed as part of model reduction.

Table 2: Examples of reductions

Construct	Ada code	Abstraction	Promela code
16 bit integer flags that take only 2 possible values	Msg_Sts1, Msg_Sts2, Msg_Sts3: Unsigned_16;	Replace with bit type	bit Msg_Sts1, Msg_Sts2, Msg_Sts3
16 bit integer flags in which only 2 bits are relevant	SFlag, SFlagC: Unsigned_16;	Replace with user-defined type with 2 bits	typedef Bits2 { bit b[2] }; Bits2 SFlag, SFlagC;
Range of values for integer type	Pr1,Pr2,Pr3 : short_integer;	Specify range of values for pressure data during verification	600 – 1200 instead of -32768 - 32767
Remove irrelevant code	Algorithm 1 to be verified	Algorithm 2 code not relevant	Remove code for Algorithm 2

In the real system, several kinds of errors can occur which affect the performance of the software. Aerospace avionics systems normally have built-in redundancy and fault handling mechanisms to mitigate the effect of such errors. To ensure the software behaves as expected in erroneous situations, a procedure was defined for introducing and simulating random errors in the model. After reading each input, it is first checked whether it is relevant for the property being currently verified. If the variable is a part of the property, it is considered significant and its value will affect the correctness of the property. Then, a random error is injected in the variable by assigning a value out of bounds, either intermittently or throughout. Otherwise, it is set to a nominal value, within the permitted ranges. The test results of model checking the launch vehicle software components with SPIN are presented in Section VI.

5 Modeling and Verification with Bounded Model Checking Approach

In Indian launch vehicles, the onboard software is developed in a safe subset of Ada83. Ada language is considered suitable for safety critical concurrent systems, due to its strong typing, ability to catch more errors during compile time rather than during runtime and tasking feature. Ada is mostly used for software development in rockets, missiles and nuclear systems. In the launch vehicle software, there are hardware specific constructs, user-defined data structures and data type conversions/ type casting supported by the in-house compiler for the indigenous onboard computer. These are specified in a library file. Also, each flight software component has to satisfy specific properties depending on its functional requirements. These cannot be verified directly with the available tools, like Polyspace, Vectorcast or GNAT.

There are several tools based on bounded model checking for C programs, such as SMACK [48], SeaHorn [49], CBMC, EsBMC, LLBMC [50], Frama-C [51] and so on. To the best of the authors' knowledge, none of these tools have a frontend support for Ada language. Also, some of these, for instance LLBMC and SeaHorn, do not handle concurrent systems. The embedded software in the OBC is real-time as well as concurrent, with a specific priority for different tasks. The existing tools do not consider the priority of tasks in the modeling. So a custom tool chain which can model the launch vehicle flight software execution environment, translate the Ada code into a suitable IR, accept properties given in a specification file and verify it is essential.

We developed such a tool chain for Ada which is also automated, with some existing tools as the foundation. Except for the requirement of preparing the specification file in a standard format for parsing by the tool, this tool is fully automated and can be run from the command line. It is general enough to model and verify other similar, periodic, safety critical, embedded software, like those in satellites, nuclear systems and aircraft. The overall approach followed for the development of this tool chain is shown in Fig. 4 and the details of each stage of processing are described in subsequent sections. The mathematical background for the tool is explained in our previous work [52]. Here, we focus on its application to a safety critical system.

The first requirement is a front-end/ pre-processor to extract relevant details from the Ada code to an intermediate data structure such as a Control Flow Graph (CFG) or Abstract

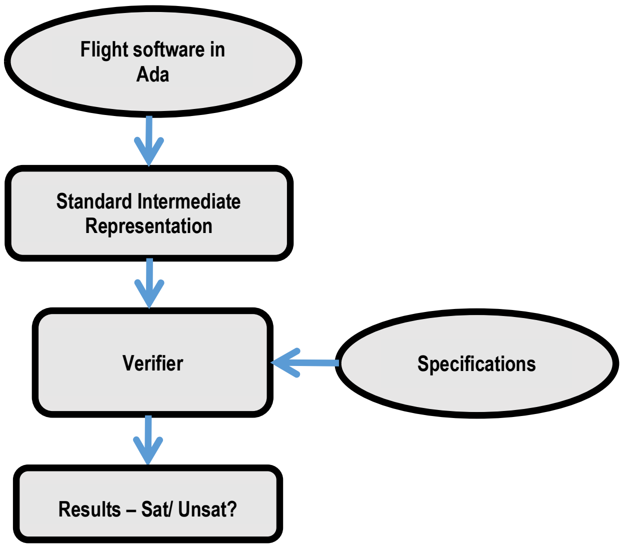


Figure 4: Verification approach with Bounded Model Checking

Syntax Tree (AST). Dragonegg [45], a gcc plug-in which can generate LLVM IR, was chosen for this purpose. LLVM is a compiler and tool chain designed around a language independent intermediate representation (IR). Dragonegg is a gcc plug-in which uses the optimizers and code generators from the LLVM project in place of GCC components. It has support for different target platforms, including ARM, x86-32/64 and others. It can act as a frontend for a wide variety of languages, including Fortran, C/C++ and Ada, besides partial handling of other languages like Java, Obj-C/C++ and Go. It accepts Ada input programs and generates LLVM IR. Since dragonegg was not compatible with the latest version of LLVM or GCC tools, several changes were implemented to customize it for our application.

The next step is to feed the LLVM IR to the static analysis tool. For this, we developed a bounded model checker named LLVMBMC. It uses Microsoft Z3 as the underlying verifier and can accept the LLVM IR generated for our software in Ada through dragonegg. The different components in the software are verified separately using our custom tool. Each component in the launch vehicle OBC software consists of multiple files. Each of these is pre-processed individually using dragonegg. Since there are function calls across files, llvm-link was used to merge all the .s files together manually before passing it as input to LLVMBMC. The overall approach for our verification with bounded model checking, starting with this merged LLVM IR file is shown in Fig 5.

We defined a specification file format for the system, containing details of each thread, including its name, entry function, priority, periodicity and the global variables relevant to the properties. The pre-conditions for the system are part of the file. All the properties of each software component, derived from the corresponding functional requirements document, are also specified in this file in terms of the input and output global variables as post conditions. This specification file is written in the standard SMT2 format and fed as an input to the model checker along

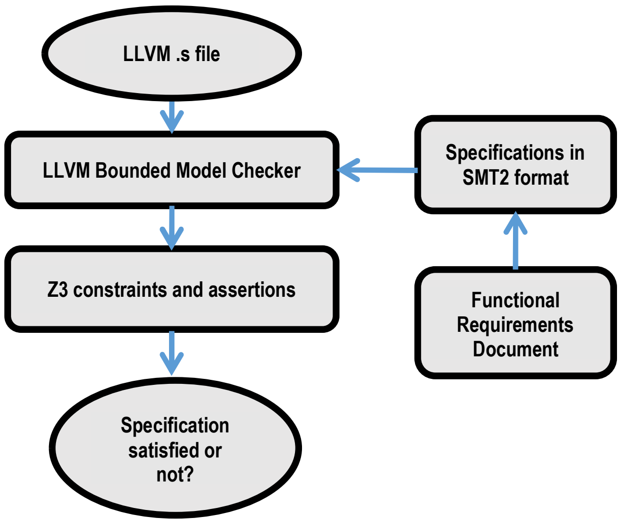

LLVM .s file
LLVM Bounded Model Checker
Specifications in SMT2 format
Z3 constraints and assertions
Functional Requirements Document
Specification satisfied or not?

Figure 5: Steps in LLVMBMC

with the IR file. A sample specification file for a system with two threads is shown below in Fig. 6. Thread1 and 2 are the two threads, with periodicity of 1ms and 2ms and priority 1 and 2, respectively. A post condition for the concurrent system is also shown.

The details of the various steps of processing carried out within the tool are now explained. The sequence of steps implemented for processing an input file in LLVM IR format is represented in Fig.7. We developed the parser for the specification file to populate different data structures. For each global variable, the name and type are stored. All parameters related to each thread, including its name, entry function, periodicity and priority are populated. The Least Common Multiple (LCM) of the periods of all threads is then computed to determine the number of iterations of each thread in a cycle. The pre- and post conditions are recognized by means of assert statements, parsed and converted to Z3 expressions. This parsing of the specification file constitutes the first step in LLVMBMC.

```
(declare-var var i16)

(declare-thread one dekker__thread0)

(invoke-param one repeated 1 priority 1)

(end-thread one)

(declare-thread two dekker__thread1)

(invoke-param two repeated 2 priority 2)

(end-thread two)

(post-condition all (assert (= var #x0002)))
```

Figure 6: A sample specification file

The translation from the functional requirement of a software module to the SMT2 format for inclusion in the specification file is illustrated with two examples below:

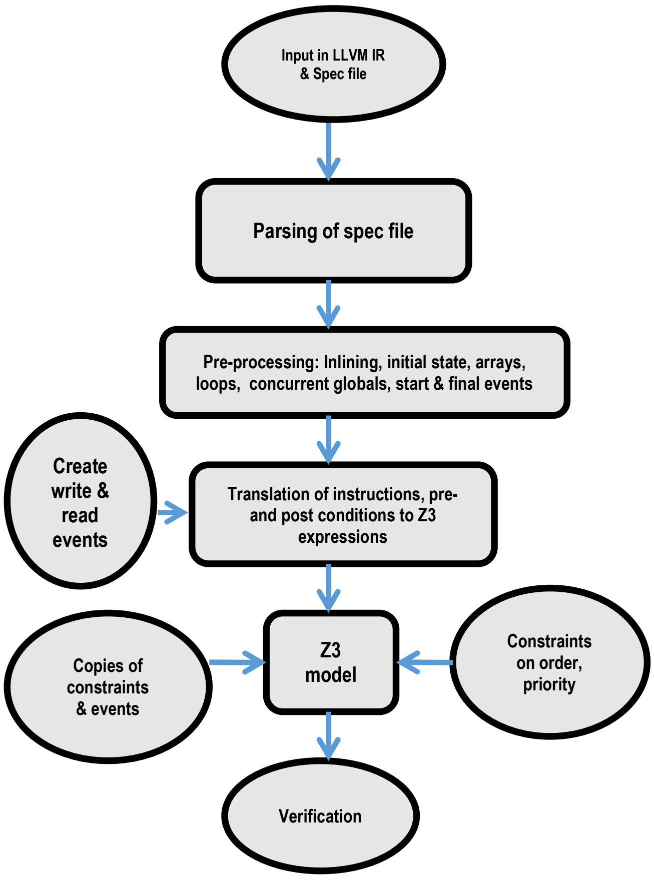

Input in LLVM IR & Spec file
Parsing of spec file
Pre-processing: Inlining, initial state, arrays, loops, concurrent globals, start & final events
Create write & read events
Translation of instructions, pre- and post conditions to Z3 expressions
Copies of constraints & events
Z3 model
Constraints on order, priority
Verification

Figure 7: Processing in LLVMBMC

Requirement1 (in Requirements document): Fuel tank pressure should always be maintained within the lower and upper limits

Property (in terms of input and output variables): *Always (@FuelVent__Pr >= @FuelVent__LowerLimit) and (@FuelVent__Pr <= @FuelVent__UpperLimit)*

Specification (in SMT2 format): `(post-condition all (assert(and (bvuge @fuelvent__pr @fuelvent__lowerlimit) (bvule @fuelvent__pr @fuelvent__upperlimit))))`

Requirement2 (in document): Clock synchronization is disabled if window out error occurs for 5 cycles

Property (in terms of input and output variables): *if (@synchronization__sync_out_wndw_err_cnt >= 5), then @synchronization__sync_disable_flg is set*

Specification (in SMT2 format): `(post-condition all (assert(=> (= @synchronization__sync_disable_flg #xAAAA)(bvsge @synchronization__sync_out_wndw_err_cnt #x0005))))`

In the next step, the combined IR file is prepared for verification by invoking the inlining pass in LLVM. Starting with the entry function, the module is completely inlined. The initial memory state is then defined, containing the parameters such as name and data type of all global variables in the module. The arrays in the module are also processed. Then, the details of loops in the inlined function are gathered. Some of the software components in our launch vehicle case study have only a single thread, while some have multiple threads. For a concurrent system with more than one thread, the truly global variables are collected as follows: for each global variable that is read or written by a particular thread, if the same variable is loaded or stored by any other thread, it is considered a concurrent global variable. The start and final events for the system are also declared here.

The processing then proceeds with the translation of the threads in the system. The pre-conditions are translated by substituting the global variable names with their updated names in the initial memory state. In the post conditions, the variable names are replaced by their modified names in the last basic block in the module being processed. These are then converted to Z3 expressions and added to the data structure containing the list of specifications. Subsequently, processing of every instruction in each basic block of the module is commenced.

The various instructions - arithmetic, logical, compare, load, store, call, branch, select and so on - are then translated to Z3 constraints. For each store and load to a truly global variable, write and read events are created and inserted into the event list of the thread. The final event for every thread is also defined. Depending on the number of iterations of a thread, copies of the constraints are created with new names for the variables in each copy. Copies of the event list of the threads are also generated according to their respective periodicities with the events renamed in the copies. For analyzing the different interleavings of events possible in the concurrent system, the principle of symbolic predictive analysis [46] is applied. The program execution is encoded in Static Single Assignment (SSA) form where every new write and read to a variable is distinguished using a distinct name for the variable. The writes could happen in different threads according to the program execution trace. A selection function chooses the value from the most recent write to a particular global variable for each read.

The constraints on order of events are then included. This consists of specifying that the start event for each thread occurs before all other events and the end event is the last event executed in the thread. Also, for the rate monotonic scheduling scheme followed in the embedded software in our launch vehicle case study, additional constraints are required. For a particular thread, every event can occur either before the start event or after the final event in a specific iteration of all threads of higher priority. For instance, if we consider a typical system with minor and major threads and higher priority for the minor threads, all events in the minor thread would be completed before any event in the major thread can occur. That is, the major thread can execute only between two iterations of the minor thread and never in the middle of an iteration of the minor thread. The lower priority thread cannot interrupt the execution of the higher priority threads.

Verification involves checking properties like confirming whether a function is invoked, in-order execution of actions, timing of events, conditional execution of events etc. Assuming the pre-conditions are true, the verifier checks whether the post conditions are satisfied or not. In case any property is not satisfied, the execution trace can be dumped and checked to get the sequence of events leading to the violation. The tool can be used for checking standard errors such as overflow, underflow and array index out of bounds. Custom specifications for the embedded software in safety critical aerospace or nuclear systems can also be verified by preparing the corresponding specification file.

The definitions, algorithm and mathematical background of LLVMBMC as explained in [52] are summarized here.

Definitions

- Concurrent, loop-free programs P consisting of a set of global variables G, a set of local variables L, pre-condition *pre*, post condition *post* and a fixed number of threads T_i with priorities r_i and periodicities p_i are considered.
- A thread Ti comprises a set of program locations Θ_i including a start and a final location and a set of edges Δ_i.
- Each edge connects two locations $\epsilon\ \Theta_i$ and an assignment of the form $Assgn$:= `assume(c)` | $l := \alpha$ | $l := g$ | $g := l$, where `assume(c)` denotes a conditional execution, l and g denote local and global variables, α is the updated value for l. The last two statements denote a *read* and a *write* respectively to a global variable.
- A branch is modeled using a location θ with one incoming edge and two outgoing edges with assume statements that are complements to each other.
- An event e is of the form $(t, \delta, R/W, g, v)$, with t denoting a thread, δ is the source edge, R/W denote *Read/Write* events, g is a global variable accessed by edge δ with value v.
- An event graph is a directed graph consisting of a set of events E and a set of edges which satisfy *well formed condition* (every *read* reads from exactly one *write*), *write serialization* (*writes* to the same variable are totally ordered) and *from-read condition* (if a *read* reads from a *write*, the subsequent *writes* should happen after the *read*).
- A program execution is represented by $(\rho, \gamma, \gamma', EG)$, where ρ is a vector of states of the threads, γ and γ' are the initial and final valuation of global variables and EG is the event graph.

Algorithm

SSA encoding

- For each global and local variable in the program, substitute a fresh SSA variable:

  $For\ l \ \epsilon\ G \ U\ L,\ \sigma_s(l) := fresh();$

- Assign initialization events of global variables to E and SSA encoding of pre-conditions to the SSA encoding that is returned φ_{ssa}:

  $E := \{(0,0,W,g,\ \sigma'(g)) \mid g\ \epsilon\ G\};$

  $\varphi_{ssa} := \sigma_s(pre);\ \varphi_{pre} := \sigma_s(pre);$

- For each thread t,

  Initialize substitution and condition map for initial thread location θ_s :

  $\sigma(\theta_s) := \sigma_s;\ C_s(\theta_s) := \varphi_{pre};$

- Process each *stmt* as follows:
  - *assume(c)* : Create a dummy event

    $E := E\ U\ (t,i,D,cd);$

  - $l := \alpha$: Append φ_{ssa} with SSA encoding of *stmt* and assign a fresh variable z for the value produced by *stmt* to $\sigma'(l)$

    $\varphi_{ssa} := \varphi_{ssa} \wedge \sigma'(z := \alpha);\ \ \sigma'(l) := z;\ \ E := E\ U\ (t,i,D,cd);$

  - $l := g$: Assign z to $\sigma'(l)$ and add a new *'read'* event.

    $\sigma'(l) := z;\ \ E := E\ U\ (t,i,R,g,z,cd);$

  - $g := l$: Append φ_{ssa} to record the write to g and create a new *'write'* event.

    $\varphi_{ssa} := \varphi_{ssa} \wedge \sigma'(z := \alpha);\ \ E := E\ U\ (t,i,W,g,z,cd);$

Constraint generation

- For encoding the event orderings, integer clock variable c_e is used as follows:

  $HB\ (e, e')$ is defined as $e.cd \wedge e'.cd \Rightarrow c_e < c_{e'}$ (if e and e' occur, e happens before e')

- For each global variable g, for each read r,
  - well-formed condition φ_{wf}: r reads from exactly one write
  - from read condition φ_{fr}: if r reads from a *write* w, the subsequent *writes* should happen after the *read*
  - write serialization condition φ_{ws}: all *writes* to g must be ordered
  - program order condition φ_{ppo}: the events in a thread must be ordered. The pre-condition occurs before and the post-condition after all the events in a thread. Also, for the rate monotonic scheduling, the events in a thread can occur either before the start event or after the final event in every iteration of all threads of higher priority. These are encoded as follows:

    $for\ e,e'\ \epsilon\ E,\ if\ e < e',\ \varphi_{ppo} := \varphi_{ppo} \wedge HB(e,e')$

    $for\ e\ \epsilon\ E|_W,\ e'\ \epsilon\ E,\ if\ e.t = 0,\ \varphi_{ppo} := \varphi_{ppo} \wedge HB(e,e')$

    $for\ e\ \epsilon\ E,\ e'\ \epsilon\ E|_R,\ if\ e'.t = 0,\ \varphi_{ppo} := \varphi_{ppo} \wedge HB(e,e')$

    $for\ e\ \epsilon\ E,\ e'\ \epsilon\ E' such\ that\ pr(e'.t) > pr(e.t)$

    $if\ e' = start,\ \varphi_{ppo} := \varphi_{ppo} \wedge HB(e,e')$

    $if\ e' = finish,\ \varphi_{ppo} := \varphi_{ppo} \wedge HB(e',e)$

6 Experiments

The case study was the flight software in the onboard computer for a typical Indian launch vehicle consisting of the real time OS in assembly language and other components like sequencing, fuel tank vent algorithm, guidance, navigation, digital autopilot, clock synchronization and rate data processing module in Ada. Together, this corresponds to ~40000 lines of code. Out of this, ~30000 lines were processed using LLVMBMC and ~15000 lines using SPIN. Since the case study is classified legacy software, a subset of the results is presented here, with comparison of the results for verification of some of the individual software components with both approaches. The tests were run on a PC with 16 GB RAM and 1 TB hard disk with Ubuntu 18.04 operating system.

6.1 Model Checking with SPIN

Fuel tank vent algorithm

The fuel tank vent software (~700 LOC in Ada source code) was modelled in Promela (~800 LOC) and simulated and verified using SPIN. The software is responsible for maintaining the fuel tank pressures within specified limits by operating various valves, as shown in Fig. 8. The model was first ensured to be executing properly as all outputs matched the outputs from the actual software. Then, relevant errors in input data were simulated for each specification: error in messages from the 3 sources of pressure data, minor ID integrity check error, sequencing command integrity error and pressure value outside limits. These were simulated intermittently, for a few cycles and throughout the run in various cases. 6 safety and liveness properties were added in the pressure vent software model.

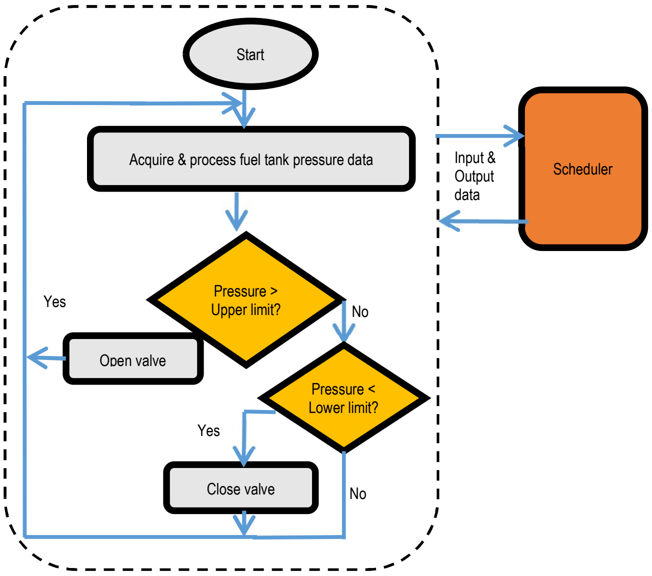


Figure 8: Fuel tank vent algorithm

The number of states, time and memory for verification of the model for 6 properties are shown in Table 3. As observed from the table, properties which involve a larger state space require more memory and time for model checking. This memory includes the stack, the hash table and other data structures inside the verifier. Two examples are shown below for the properties verified in the fuel tank vent algorithm:

Property1: Fuel tank pressure should always eventually be maintained within the lower and upper limits

```
ltl STS {always eventually (p1 && q1)}
```

`p1:` Pressure >= Lower limit, `q1:` Pressure < Upper limit

Property2: Invoke the algorithm processing if there is no sequencing command integrity error

```
ltl STS {always (!p2 implies q2)}
```

`p2:` Sequencing command integrity error, `q2:` Algorithm is invoked

The remaining properties which were verified are the following:

Property3: When communication status is not OK for 10 cycles for the pressure sensors, the algorithm is disabled.

Property4: If there is sequencing command integrity error, the pressure error counter and bound check failure counter are reset.

Property5: The median value of tank pressures is computed if the communication status is OK for the pressure sensors; else the previous cycle data is retained.

Property6: If the algorithm is disabled, the valve in the tank should be closed and execution of the algorithm is stopped.

Table 3: State space, time and memory for verification

Property	Number of reachable states	Elapsed time (in seconds)	Memory (MB)
Property1	8956344	11.1	94.235
Property2	35643699	54.9	216.305
Property3	1637	1.1	85.348
Property4	41782929	162	216.305
Property5	8962540	10.8	94.235
Property6	3618	2.1	85.739

Clock synchronization

The algorithm for clock synchronization (~200 LOC in Ada source code) was modelled in Promela (~400 LOC) and simulated and verified using SPIN. The synchronization software maintains time synchronization between the clocks of the processors functioning as onboard computers, as shown in Fig. 9. With nominal inputs, we executed the model and confirmed that the output values match those from the actual software. 4 properties were included for verification in the model. The properties that were verified for the clock synchronization algorithm are listed in Table 4. The number of states, memory and time for verification of the model are shown in Table 5.

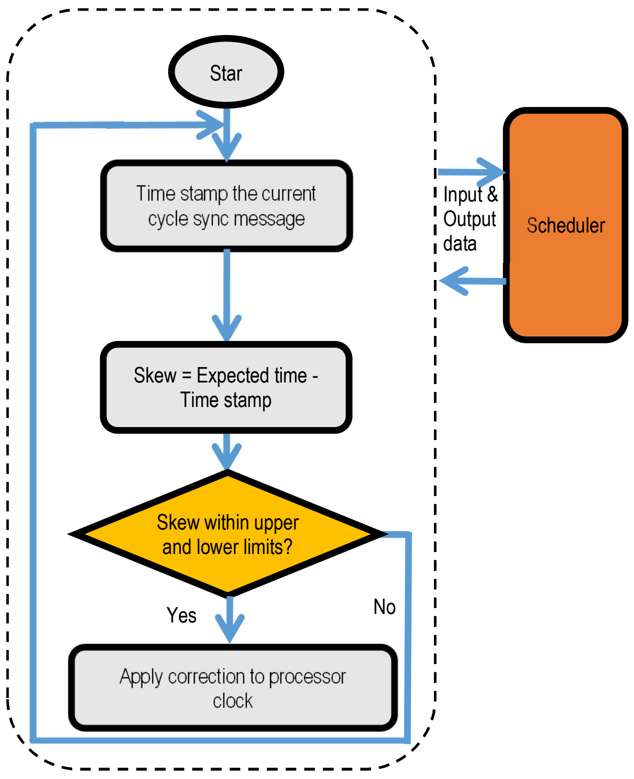

Star
Time stamp the current cycle sync message
Skew = Expected time - Time stamp
Skew within upper and lower limits?
Yes
No
Apply correction to processor clock
Input & Output data
Scheduler

Figure 9: Clock synchronization scheme

Table 4: Properties verified in clock synchronization

Property	**Details of property**
Property1	If the absolute value of clock skew is outside the lower and upper bounds, no clock correction is applied.
Property2	Clock synchronization is disabled if window out error occurs for 5 cycles.
Property3	If time stamp value is not received from the master for 5 consecutive cycles, error bit in sync flag is set.
Property4	If the absolute value of clock skew is greater than the upper bound for 5 consecutive cycles, synchronization is disabled permanently.

Table 5: State space, time and memory for verification – Clock synchronization

Property	**Number of reachable states**	**Elapsed time (in seconds)**	**Memory (MB)**
Property1	193633	0.12	763.645
Property2	169937	0.13	610.352
Property3	363570	0.13	853.48
Property4	281692	0.13	816.305

Though these software modules are part of legacy code, the verification attempt instilled confidence in the developers on the utility of formal methods in complementing manual testing. The properties that were satisfied re-affirmed that the software requirements are met whereas the detection of known bugs in older versions of the code confirmed the utility of the tool. Compared to the traditional V&V methods which are carried out on the code, model checking with SPIN is done on the Promela model of the system. This model can be generated from the software requirements itself, thus enabling the detection of errors much earlier in the software development lifecycle. This would be greatly beneficial for new, complicated designs, like human-rated missions. Also, the injection of random faults in the model helped in establishing the correctness and adequacy of error handling implemented in the system. Since redundancy and fault handling are essential aspects of safety critical systems, this error injection testing forms a significant part of the verification workflow.

6.2 Bounded Model Checking with LLVMBMC

To ensure the correctness of the implementation of the tool chain based on bounded model checking, simple standard benchmarks such as concurrency litmus tests and mutual exclusion protocols were first selected as the test programs. The litmus tests include standard programs like Store buffering (SB), Load buffering (LB) and other small, carefully designed, parallel programs that exercise the memory model of a shared memory computer. In concurrent systems, various mutual exclusion protocols such as Dekker's protocol, Peterson's algorithm and others are used to protect shared resources from concurrent non-atomic accesses. The litmus tests and mutual exclusion algorithms were coded in Ada and fed into the tool for initial testing. The verification was successful only when the priority was also considered properly in the case of the mutual exclusion protocols, thus increasing the confidence on the validity of the tool.

Next, each software component in the aerospace case study was first fed to dragonegg to generate LLVM IR. All the files in a component were combined using llvm-link and this combined IR .s file was given as input to our bounded model checker LLVMBMC. The commands for running the tool are as follows:

```
gcc-8  -S  -fplugin=./dragonegg.so  -fplugin-arg-dragonegg-emit-ir ./Examples/test.adb -o test.s
```

```
./llvmbmc -b examples/AssemblyFiles/test.s
```

At first, software components with only a single thread, like fuel tank vent algorithm, rate data processing module and clock synchronization scheme were verified. The specification file for each was also written in SMT2 format, with properties derived from the corresponding functional requirements document. The results of the verification, including the lines of code in the source file, the number of specifications verified and the pass/ fail status are listed in Table 6. It was seen that one property is violated in the fuel tank vent algorithm. On further discussions with the designers, it was understood that though the violation detected by the tool is correct in a standalone manner, it is handled properly at the system level in the real scenario. Similarly, in the clock synchronization module, the tool threw up a violation of one specification. This was because one variable was not initialized in the individual module, but it is done elsewhere in the actual integrated software. On including this initialization in the driver file, the property was satisfied.

Table 6: Launch vehicle software – Results

Software component	Size (LOC)	No. of specifications	Violations
Fuel tank vent algorithm	700	6	1
Rate data processing	900	30	0
Clock Synchroniza-tion	200	4	1
Guidance	1600	15	0

In the next step, the guidance software with two threads and several complex computations was verified with LLVMBMC. The priority among the threads was considered correctly by the tool. All the specifications were found to be satisfied. Some of the specifications involved checking the proper sequence of function calls in the module during execution. For such call sequence properties, a monitor variable is inserted which is initialized to 0. When the first function in the sequence is invoked, it is made equal to 1. When the second function is called, it is checked whether the variable is 1. If yes, it is left unchanged; otherwise it is set to 2, indicating an improper order of function calls. At the end of the module, compare and assert instructions are added to check the value of the monitor variable. A value of 1 for the variable indicates that the call sequence property is satisfied and any other value denotes a violation.

The navigation and guidance software, part of the case study, consist of hundreds of single and double precision floating point variables and complex computations, including trigonometric operations. During our experiments, the verification of some of the properties for these components was either not completing or taking 4-5 hours for completion. So, some trials were carried out using strategies such as reducing the number of bits in the significand and specifying finite ranges for some inputs through pre-conditions. At first, a few small Ada programs were written with arithmetic (addition, subtraction, multiplication, division) and trigonometric (sine, cosine, sin inverse, cos inverse, arc tan) and other (square root, exponent, logarithm) operations on floating point variables. It was seen that, for a single property, as number of bits in the significand for float values decreases, the time and memory for verification also reduce. In addition, properties with more intensive computations like division and arc tan take more time. Similar techniques resulted in expediting the verification of the specifications for the navigation software also.

We also tested the tool with the onboard software components in different missions, with bugs found during various stages in development - designer level testing, code inspection by QA team, simulations and launch. The corresponding property violations were detected by LLVMBMC, proving its utility in detecting actual bugs in the case study. The results are summarized in Table 7. To illustrate the nature of bugs, two of these are explained below. In the clock synchronization module, since the timer uses a down counter initialized to 5000, the time stamped value is subtracted from 5000 to get the actual value. During a simulation test, the time stamp location contained an erroneous large value (>5000) and the subtraction operation caused an overflow. This overflow was correctly captured by LLVMBMC with the old code. Subsequently, the software was corrected by limiting the time stamp value to 5000 if it is greater than 5000. The original code snippet without the limiting is as shown:

```
TimeStamp := 5000 - TimeStamp;

Skew := ExpectedTime - TimeStamp;
```

Table 7: Results of flight software verification

Module	Error	Violation detected?
Clock synchroniza-tion	Overflow in computation of skew	Yes
Navigation	Saturation of result in a computation causing arithmetic error	Yes
Aided navigation	Missing initialization of global variables leading to setting of error flag in case of communication error	Yes
Sequencing interface	Wrong packing of complement of a byte in a 16 bit word leading to complement check failure	Yes
Guidance	Data passing from lower priority to higher priority task was non-atomic causing failure of complement integrity check	Yes

In the code for sequencing interface, the event information (range 00H-FFH) and its logical negation are packed in the LS byte of a flag and its complement flag, respectively. This is extracted and assigned to the corresponding variables in sequencing software, where the variables are checked for integrity using complement check. After extracting the negation of the event information from the flag, it should be ORed with FF00H to form the complement correctly. This was missed initially in the code and the complement check failure was raised in a subsystem level simulation run where this path was exercised. Later, the code was modified to form the negation properly. The initial code snippet with the bug, where LLVMBMC detected the error as a violation of a property (the error flag is always zero) is as shown:

```
Event  :=  Bit_And(EvntIp, 16#00FF#);

EventC :=  Bit_And(EvntIpC, 16#00FF#);

if  (Event  +  EventC  /=  16#FFFF#)  then
      Err_Flag := 16#0001#;

end if;
```

7 Discussion

In this section, a comparison of the two verification approaches is presented - using SPIN model checker and LLVMBMC. A comparison of the two approaches for some of the modules in the launch vehicle flight software is shown in Table 8. It lists the time and memory for verification of the same property with both tools when applied to the same software components.

Table 8: Comparison – Verification with SPIN and LLVMBMC

Software component	Time (seconds)		Memory (MB)	
	SPIN	LLVM BMC	SPIN	LLVM BMC
Fuel tank vent algorithm	54.9	2.6	216.305	140.968
Clock Synchronization	0.13	0.08	853.48	104.496
Input data validation	1.1	0.12	85.934	128.488
Output data filling	2.7	0.13	189.524	126.788

Table 9 compares SPIN and LLVMBMC with respect to their main features, merits and limitations. The advantages of SPIN are that it has a simple user interface, a C-like modelling language and is highly suitable for formal verification of communication protocols and other software systems. The properties to be verified can be included in the model as assertions or LTL formulae. Sufficient data types are supported and different switches can be used to speed up the verification. We were also able to define a systematic procedure for translation of Ada code to Promela. It is possible to insert C code directly into the model too using the c_code primitive. The code must be syntactically valid C and can be used to include a larger piece of code that is stored in a separate file into a model.

On the other hand, SPIN model checker has certain drawbacks also. It does not scale for large systems. Significant abstraction of the model is necessary to complete the verification within a reasonable time. It is extremely difficult to analyse an integrated flight software with several components using a single model. The development of the system model in Promela is a manual process and is hence prone to errors. Promela does not support float data types and so SPIN cannot be used for verifying computation-intensive applications with floating point operations. Thus, launch vehicle software components such as navigation and guidance, involving numerous floating point computations could not be verified with SPIN.

Table 9: Comparison of SPIN and LLVMBMC features

SPIN	LLVMBMC
Simple user interface, a C-like modelling language named Promela	Tool is customised for Ada and does not require any translation or abstractions
Properties to be verified are included in the model as assertions or LTL formulae	Accepts and processes custom properties listed in a specification file in the standard SMT2 format
Does not support float data types	Supports floating point data type also
Does not scale for large systems	Can be used for analysis of large software systems
Possible to insert C code directly into the model	Can be used for verification of C/C++ programs also using a Clang compiler frontend

Coming to the bounded model checking based static analysis tool, it is customized for Ada and is fully automated. It accepts the Ada source code directly and does not require any translation or abstractions. It models the rate monotonic priority among tasks and thus represents the execution environment of the launch vehicle software accurately. It supports floating point data type also and can be used for analysis of large software systems. It checks for errors like overflows, underflows, array indices out of bounds etc. It can accept and process custom properties listed in a specification file in the standard SMT2 format and generate the execution trace in case of violations. It uses LLVM IR as the intermediate form and can be used for verification of C/C++ programs also using a Clang compiler frontend. It can be used by other teams working on software for safety critical applications such as the nuclear sector, satellite systems and automotive software. The tool chain developed by us has some limitations also. The specification file should be written in a particular format by the user. The tool cannot be used for analysis of assembly programs at present.

In summary, SPIN model checker is suitable for a beginner in formal verification, to verify small systems. For verification of large systems with varied data types and diverse components, our custom tool would be more advantageous and convenient. A GUI based interface support for more general properties would make it more user-friendly.

8 Conclusion

In this work, the formal verification of safety critical embedded software in Ada language is attempted through two different approaches. The first method, using the SPIN model checker, is suitable for small to medium systems, but requires manual development of the model with intelligent abstractions. The customized, automated tool chain LLVMBMC for verification of Ada programs has a suitable front end and carries out analysis based on bounded model checking. Both techniques were applied to a real-world case study, the embedded software in the onboard computer of a launch vehicle. It was seen that the custom tool is more advantageous and can be used for verification of Ada code in other, similar systems.

Specifically, LLVMBMC is scalable for larger software implemented in Ada language and can be used without any manual translation or development of models with abstractions. It supports varied data types, including floating point and finishes the verification typically ten times faster than tools like the SPIN model checker. It can model any concurrent system with tasks running at different periodicities and priorities using minimal information provided by the user in a specification file. Other tools do not consider this priority automatically for a concurrent system and demand considerable manual effort for accurate modeling.

In future, it is proposed to develop a GUI interface so that users can verify general and custom specifications for their software more conveniently. It is also planned to augment the tool by implementing optimizations for particular, repetitive computations in the launch vehicle software.

References

[1] J.L. Lions, *Ariane 5—Flight 501 Failure*, European Space Agency, Paris, France, Tech. Rep, 1996.

[2] N. G. Leveson and C. S. Turner, *An investigation of the Therac-25 accidents*, in *Computer*, vol. 26, no. 7, pp. 18-41, 1993

[3] E. Marshall, *Fatal error: how Patriot overlooked a Scud.* Science 255, no. 5050: 1347-1347, 1992.

[4] A. G. Stephenson, D. R. Mulville, F. H. Bauer, G. A. Dukeman, P. Norvig, L. S. LaPiana, P. J. Rutledge, D. Folta, and R. Sackheim 1999, *Mars Climate Orbiter Mishap Investigation Board—Phase I Report*, NASA, Tech. Rep,1999.

[5] European Cooperation for Space Standardization Std. ECSS-ST-40C, *Space Engineering—Software*, 2009.

[6] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate, *Testing or formal verification: DO-178C alternatives and industrial experience*, IEEE Softw., vol. 30, no. 3, pp. 50–57, 2013.

[7] C. A. R. Hoare, *An axiomatic basis for computer programming*, Commun. *ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[8] P. Cousot and R. Cousot, *Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in *Proc. 4th ACM SIGACT-SIGPLAN Symp. Principles of Programming Languages (ser. POPL'77)*, New York, NY, USA, pp. 238–252, 1997.

[9] Comprehensive Static Analysis Using Polyspace Products-White paper by Mathworks.

[10] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, *Bounded model checking*, Handbook of satisfiability 185, no. 99, pp. 457-481, 2009.

[11] G.J.Holtzman, *The SPIN Model Checker, Primer and Reference Manual*, Addison-Wesley, Boston, 2003.

[12] R.A.B e Silva, N. N. Arai, L. A. Burgareli, J. M. P. de Oliveira, and J. S. Pinto, *Formal verification with Frama-C: A case study in the space software domain*, IEEE Transactions on Reliability 65, no. 3, pp. 1163-1179, 2015.

[13] W. A. Hunt Jr, M. Kaufmann, J. Strother Moore, and A. Slobodova, *Industrial hardware and software verification with ACL2*, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 375, no. 2104 : 20150399, 2017.

[14] M. D..Harrison, P. Masci, J. C. Campos, and P. Curzon, *Verification of user interface software: the example of use-related safety requirements and programmable medical devices*, IEEE Transactions on Human-Machine Systems 47, no. 6, pp. 834-846, 2017

[15] G. Hird, *Formal specification and verification of Ada software*, In 8th Computing in Aerospace Conference, p. 3713, 1991.

[16] S. Li, L. Qiao, and M. Yang, *Memory state verification based on inductive and deductive reasoning*, IEEE Transactions on Reliability 70, no. 3, pp. 1026-1039, 1991.

[17] M. Farrell, N. Mavrakis, C. Dixon and Y. Gao, *Formal verification of an autonomous grasping algorithm*, In International Symposium on Artificial Intelligence, Robotics and Automation in Space, ESA, 2020.

[18] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival, *The ASTRÉE analyzer*, In Programming Languages and Systems: 14th European Symposium on Programming, ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005. Proceedings 14, pp. 21-30. Springer Berlin Heidelberg, 2005.

[19] O. Bouissou, E. Conquet, P. Cousot, R. Cousot, J. Feret, K. Ghorbal, E. Goubault et al., *Space software validation using abstract interpretation*, In The International Space System Engineering Conference: Data Systems in Aerospace-DASIA 2009, vol. 1, pp. 1-7, European Space Agency, 2009.

[20] V. Todorov, F. Boulanger, and S. Taha, *Formal verification of automotive embedded software,* In Proceedings of the 6th Conference on Formal Methods in Software Engineering, pp. 84-87, 2018.

[21] Ph Lacan, J. N. Monfort, L. V. Q. Ribal, A. Deutsch, and G. Gonthier, *Ariane 5-the software reliability verification process*, In DASIA 98-Data Systems in Aerospace, vol. 422, p. 201, 1998.

[22] C. T. Spiliopoulou, E. B. Viñuela, and B. F. Adiego, *Experience With Static PLC Code Analysis at CERN*, In 16th Int. Conf. on Accelerator and Large Experimental Control Systems (ICALEPCS'17), Barcelona, Spain, pp. 1787-1791, 2017.

[23] E. Clarke, D. Kroening, and F. Lerda, *A tool for checking ANSI-C programs*, In Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference, TACAS 2004, Proceedings 10, pp. 168-176, Springer Berlin Heidelberg, 2004.

[24] N. Ge, E. Jenn, N. Breton, and Y. Fonteneau, *Formal verification of a rover anti-collision system,* In Critical Systems: Formal Methods and Automated Verification: Joint 21st International Workshop on Formal Methods for Industrial Critical Systems and 16th International Workshop on Automated Verification of Critical Systems, FMICS-AVoCS 2016, Pisa, Italy, Proceedings 21, pp. 171-188, Springer International Publishing, 2016.

[25] K. Yang, C. Tian, N. Zhang, Z. Duan and H. Du, *A CEGAR-Based Static–Dynamic Approach to Verifying Full Regular Properties of C Programs*, IEEE Transactions on Reliability 70, no. 4, pp. 1455-1467, 2021.

[26] O.M. Alhawi, H.Rocha, M. R. Gadelha, L. C. Cordeiro, and E. Batista, *Verification and refutation of C programs based on k-induction and invariant inference*, International journal on software tools for technology transfer 23, pp. 115-135, 2021.

[27] L. Asplund and K. Lundqvist, *Safety critical systems based on formal models*, ACM SIGAda Ada Letters 20, no. 4, pp. 32-39, 2000.

[28] B. Dobbing and A. Burns, *The Ravenscar Tasking Profile for High Integrity Real-Time Programs*, In SIGAda'98, 1998.

[29] S. Miller, E. Anderson, L. Wagner, M. Whalen, and M. Heimdahl, *Formal verification of flight critical software*, In AIAA Guidance, Navigation, and Control Conference and Exhibit, p. 6431, 2005.

[30] J. Yoo, E. Jee and S. Cha, *Formal modeling and verification of safety-critical software*, IEEE software 26, no. 3, pp. 42-49.

[31] O. Sokolsky, M. Younis, I. Lee, H. Kwak, and J. Zhou, *Verification of the redundancy management system for space launch vehicle: a case study*, In Proceedings of Fourth IEEE Real-Time Technology and Applications Symposium, pp. 220-229, 1998.

[32] J. Lahtinen, J. Valkonen, K. Björkman, J. Frits, I. Niemelä and K. Heljanko, *Model checking of safety-critical software in the nuclear engineering domain*, Reliability Engineering & System Safety, pp. 104-113, 2012.

[33] R. C. Bhushan and D. K. Yadav, *Modeling and Formally Verifying a Safety-Critical System Through MCRL2*, In 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence), pp. 775-779, IEEE, 2018.

[34] X. Gan, J. Dubrovin, and K. Heljanko, *A symbolic model checking approach to verifying satellite onboard software*, Science of Computer Programming 82, pp. 44-55, 2014.

[35] H. Mkaouar, B. Zalila, J. Hugues, and M. Jmaiel, *A formal approach to AADL model-based software engineering*, International Journal on Software Tools for Technology Transfer, pp. 219-247, 2020.

[36] F. Yan and T. Tang, *Formal modeling and verification of real-time concurrent systems,* In 2007 IEEE International Conference on Vehicular Electronics and Safety, pp. 1-6, 2007.

[37] L. Huang, EY.Kang, *Formal Verification of Safety & Security Related Timing Constraints for a Cooperative Automotive System*, In: Hähnle, R., van der Aalst, W. (eds) Fundamental Approaches to Software Engineering. FASE 2019, 2019.

[38] S. Chouali, A. Boukerche, A. Mostefaoui and M. A. Merzoug, *Formal Verification and Performance Analysis of a New Data Exchange Protocol for Connected Vehicles*, in *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15385-15397, 2020.

[39] P. R. Gluck and G. J. Holzmann, *Using SPIN model checking for flight software verification*, Proceedings of Institute of Electrical and Electronics Engineers (IEEE) Aerospace Conference, Big Sky, MT, USA, pp. 1-1, 2002.

[40] F. Schneider, S.M.Easterbrook, J.R.Callahan, and G.J.Holzmann, *Validating Requirements for Fault Tolerant Systems using Model Checking*, Proceedings of the Third Institute of Electrical and Electronics Engineers (IEEE) International Symposium on Requirements Engineering, Colorado Springs, CO, USA, pp. 4-13, 1998.

[41] K. Havelund, M. Lowry, S.J. Park, C. Pecheur, J. Penix, J. Visser and J.L. White, *Formal Analysis of the Remote Agent Before and After Flight*, Proceedings of Fifth NASA Langley Formal Methods Workshop, Williamsburg, Virginia, 2000.

[42] G. Horvath, G. Jones, and R. Joshi, *A Model-based Approach to Verification of Spacecraft Software using*

the SPIN Model Checker, AIAA SPACE 2009 Conference & Exposition, Pasadena, California, 2009.

[43] K. Havelund, M. Lowry and J. Penix, *Formal analysis of a space-craft controller using SPIN*, IEEE Transactions on Software Engineering, vol. 27, no. 8, pp. 749-765, 2001.

[44] The Adacore website.

[45] LLVM Team, *Dragonegg-Using LLVM as a GCC Backend,* 2013.

[46] C. Wang, S. Kundu, R. Limaye, M. Ganai, and A. Gupta, *Symbolic predictive analysis for concurrent programs,* Formal aspects of computing 23, pp. 781-805, 2011.

[47] R. Krishnan and V.R Lalithambika, *Modeling and Validating Launch Vehicle Onboard Software Using the SPIN Model Checker*, Journal of Aerospace Information Systems, 17(12), pp. 695-699, 2020.

[48] M. Carter, S. He, J. Whitaker, Z. Rakamaric and M. Emmi, *SMACK Software Verification Toolchain*, 2016 Institute of Electrical and Electronics Engineers (IEEE) / American Computing Machinery (ACM) 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, USA, pp. 589-592, 2016.

[49] A. Gurfinkel, T. Kahsai, and J. A. Navas, *SeaHorn: A framework for verifying C programs*, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 447–450, 2015.

[50] F. Merz, S. Falke, and C. Sinz, *LLBMC: Bounded model checking of C and C++ programs using a compiler IR*, *Verified Software: Theories, Tools, Experiments (VSTTE)*, pp. 146–161, 2012.

[51] L. Correnson, P. Cuoq, F. Kirchner, A. Maroneze, V. Prevosto, A. Puccetti, J. Signoles and B. Yakobowski, *Frama-C User Manual For Frama-C 28.0 (Nickel)*, CEA List, Saclay, France, 2023.

[52] R Krishnan and A Gupta, *Modelling Task Priority in Symbolic Predictive Analysis for Embedded Software in Ada*, Ada-Europe 2024 (accepted for presentation in journal track).

[53] J.M. Faria, J. Martins, J.S. Pinto, *An Approach to Model Checking Ada Programs,* M. Brorsson, L.M.Pinho (eds) *Reliable Software Technologies – Ada-Europe 2012,* LNCS 7308, Springer-Verlag, 2012.

[54] C.B. Lourenço, M.J. Frade, J.S. Pinto, *A Bounded Model Checker for SPARK Programs,* F. Cassez, J.F.Raskin, (eds), *Automated Technology for Verification and Analysis,* ATVA 2014, LNCS 8837, Springer, 2014.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard

Ada-Deutschland

Dr. Hubert B. Keller CEO
ci-tec GmbH
Beuthener Str. 16
76139 Karlsruhe
Germany
+491712075269
Email: h.keller@ci-tec.de
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch

Ada-Europe Sponsors

Ada Edge

27 Rue Rasson
B-1030 Brussels
Belgium
Contact: Ludovic Brenta
ludovic@ludovic-brenta.org

AdaCore

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

AdaLabs
innovate.all

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

ADA LOG

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

Deep Blue Capital

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

Ellidiss Technologies

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

EUROCITY

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

KONAD
Software for Control and Administration

In der Reiss 5
D-79232 March-Buchheim
Germany
Contact: Frank Piron
info@konad.de
www.konad.de

PTC®
Developer Tools

3271 Valley Centre Drive,Suite 300
San Diego, CA 92069
USA
Contact: Shawn Fanning
sfanning@ptc.com
www.ptc.com/developer-tool

SYSADA

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

systerel
Safe real-time solutions

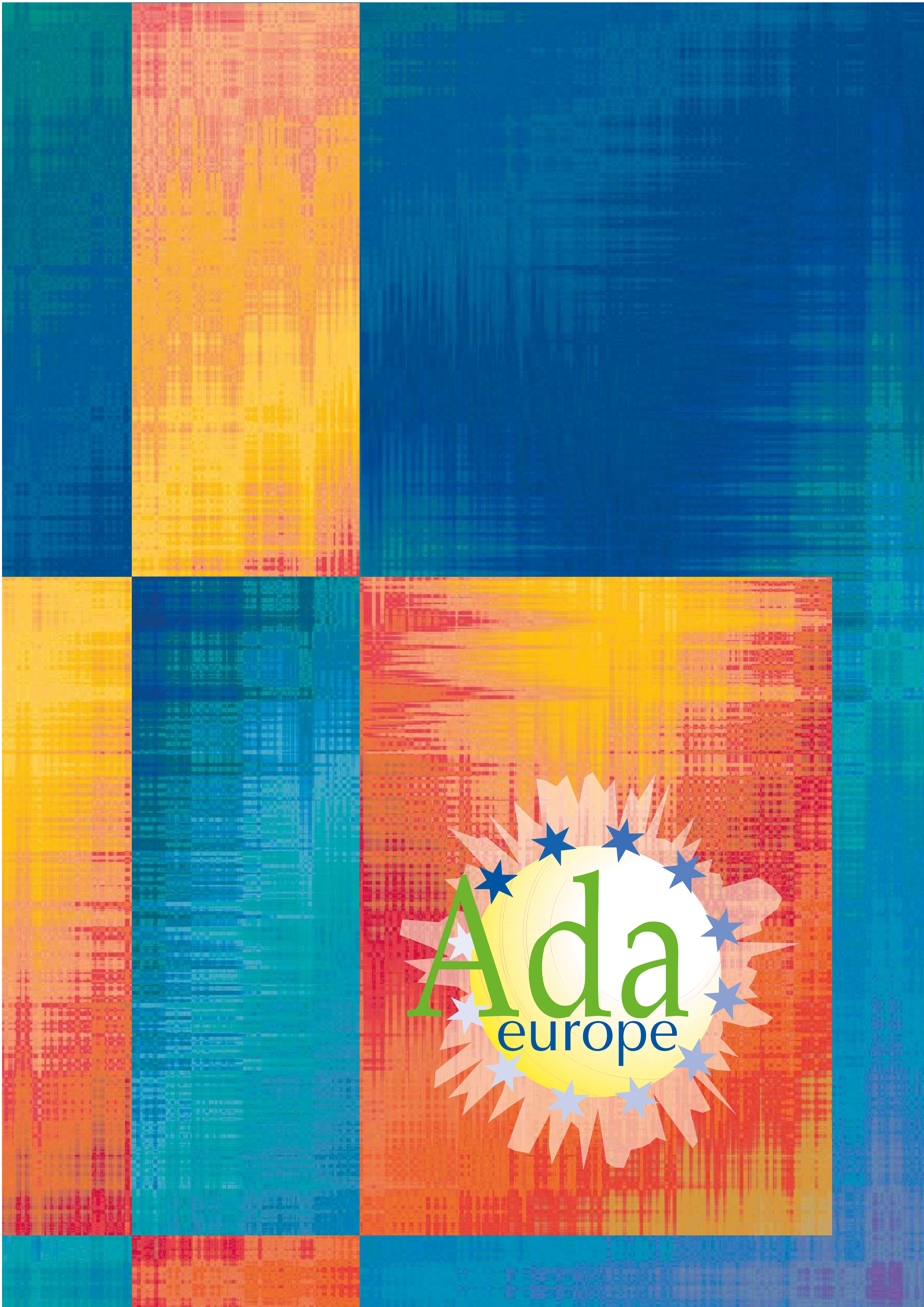
1115 Rue René Descartes
13100 Aix en Provence
France
Contact: Patricia Langle
patricia.langle@systerel.fr
www.systerel.fr/en/

Tid rum

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

WhiteElephant

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


Ada
europe