ADA USER JOURNAL

Volume 39
Number 4
December 2018

Contents

Editorial

This Editorial starts with a very sad note, as the issue remembers two friends that left us in 2018. Jacob, which I was fortunate to work with in the past six years in his role as Ada User Journal News Editor, and José, a colleague for 20 years, since we were both doing our PhDs, and one of the organizers of the RUME workshop at Ada-Europe 2018. Although passing away in a period of 6 months, coincidence made both be remembered in this issue. I leave to the *in memoriams* the words of remembrance.

As for the technical contents of the issue, the reader will find a set of contributions related to the Ada-Europe conference week, that took place last June, in Lisbon, Portugal.

Frist, a paper derived from an industrial presentation at the conference, from a group of authors of Intecs, Italy, on the use of agile software development approaches in the railway domain. The second part of the issue publishes the proceedings of the workshop on Challenges and New Approaches for Dependable and Cyber-Physical System Engineering (DeCPS 2018), which as usual co-located with the Ada-Europe conference. The published papers provide overviews of several of running European projects addressing this difficult challenge of building dependable cyber-physical systems and systems of systems.

Finally, the issue also publishes the proceedings of the Runtime Verification and Monitoring Technologies for Embedded Systems Workshop (RUME 2018), which was also co-located with Ada-Europe, in Lisbon. The workshop papers address different approaches to provide runtime monitoring and verification capabilities, more and more a significant challenge for more adaptative, whilst reliable, systems.

The Ada-Europe conference week is an important meeting point for researchers and practitioners in all aspects of reliable technologies and systems. Therefore, I hope to see you all, next June, in Warsaw, for Ada-Europe 2019.

Luís Miguel Pinho
Porto
December 2018
Email: AUJ_Editor@Ada-Europe.org

In Memoriam: Jacob Sparre Andersen

Jacob Sparre Andersen, Ada User Journal News Editor, passed away on Sunday 16th December 2018, after a very short period fighting an aggressive cancer. The news of Jacob's illness was very sudden, and quickly escalated. We were still trying to accept that he was seriously ill when we received news that he was no longer with us.

Jacob was a long-time member of the Ada community, and a very active one at that. He was always keen on helping newbies (and others) about how to program high-quality software, and an enthusiastic interlocutor to everyone approaching Ada technology. He regularly participated in groups, discussions and events, promoting Ada, software quality and open source software, showing how these three could be combined harmoniously.

He was also an active volunteer within Ada-Europe. Not only in the role of Ada User Journal News Editor, a position that he fulfilled in the past six years, but also in his continuous support to the Ada-Europe conferences, in multiple roles, from participation in the industrial committee (which he chaired in 2017) to the multiple presentations and tutorials he offered. He made himself known also through many comments and engaging discussions that he sparked during and after conference sessions.

We will always remember Jacob as a very friendly, gentle and helpful person. Our community will miss him very much. And next June in Warsaw, we will find it very awkward to look around and not see him stand out in the conference crowd.

Farewell, our friend and colleague Jacob, Rest in Peace.

Ada-Europe Board
December 2018

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.

Quarterly News Digest

Kristoffer Nyborg Gregertsen

SINTEF, Email: kristoffer.gregertsen@sintef.no

Contents

Ada-related Events

[To give an idea about the many Ada-related events organised by local groups, some information is included here. If you are organising such an event feel free to inform us as soon as possible. If you attended one please consider writing a small report for the Ada User Journal.]

FOSDEM 2019

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Fri, 16 Nov 2018 19:58:34 -0000
Subject: CfP - Ada Developer Room at FOSDEM 2019, Brussels, Belgium
Newsgroups: comp.lang.ada, fr.comp.lang.ada

Call for Presentations

9th Ada Developer Room at FOSDEM 2019

Saturday 2 February 2019, Brussels, Belgium

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/19/190202-fosdem.html

Organized in cooperation with Ada-Europe

Ada-Belgium [1] is pleased to announce that there will be a one-day Ada Developer Room on Saturday 2 February 2019 at FOSDEM 2019 in Brussels, Belgium. This Ada DevRoom is once more organized in cooperation with Ada-Europe [2].

General Information

FOSDEM [3], the Free and Open source Software Developers' European Meeting, is a free and non-commercial two-day weekend event organized early each year in Brussels, Belgium. It is highly developer-oriented and brings together 8000+ participants from all over the world. No registration is necessary.

The goal is to provide open source developers and communities a place to meet with other developers and projects, to be informed about the latest developments in the open source world, to attend interesting talks and presentations on various topics by open source project leaders and committers, and to promote the development and the benefits of open source solutions.

Ada Programming Language and Technology

Awareness of safety and security issues in software systems is increasing. Multi-core platforms are now abundant. These are some of the reasons that the Ada programming language and technology attracts more and more attention, among others due to Ada's support for programming by contract and for multi-core targets.

The latest Ada language definition was updated early 2016. Work on new features is ongoing, such as improved support for fine-grained parallelism, and will result in a new Ada standard scheduled for 2020.

Ada-related technology such as SPARK provides a solution for the safety and security aspects stated above.

More and more tools are available, many are open source, including for small and recent platforms. Interest in Ada keeps increasing, also in the open source community, and many exciting projects have been started.

Ada Developer Room

FOSDEM is an ideal fit for an Ada Developer Room. On the one hand, it gives the general open source community an opportunity to see what is happening in the Ada community and how Ada technology can help to produce reliable and efficient open source software. On the other hand, it gives open source Ada projects an opportunity to present themselves, get feedback and ideas, and attract participants to their project and collaboration between projects.

At previous FOSDEM events, the Ada-Belgium non-profit organization organized very well attended Ada Developer Rooms, offering a full day program in 2006 [4], a two-day program in 2009 [5], and full day programs in 2012 [6], 2013 [7], 2014 [8], 2015 [9], 2016 [10], and 2018 [11]. An important goal is to present exciting Ada technology and projects also to people outside the traditional Ada community.

Our proposal for another dedicated Ada DevRoom was accepted, and now work continues to prepare the detailed program. We most probably will have a total of 8 schedulable hours between 11:00 and 19:00 in one of the rooms which accommodate from 59 to 85 participants.

More information will be posted on the dedicated web-page on the Ada-Belgium site [12], and final announcements will of course also be sent to various forums, lists and newsgroups.

Call for Presentations

We would like to schedule technical presentations, tutorials, demos, live performances, project status reports, discussions, etc., in the Ada Developer Room.

Ada-Belgium calls on you to:

- inform us at ada-belg...@cs.kuleuven.be about specific presentations you would like to hear in this Ada DevRoom;
- for bonus points, subscribe to the Ada-FOSDEM mailing list [13] to discuss and help organize the details;
- for more bonus points, be a speaker: the Ada-FOSDEM mailing list is the place to be!

Do you have a talk you want to give?

Do you have a project you would like to present?

Would you like to get more people involved with your project?

We're inviting proposals that are related to Ada software development, and include a technical oriented discussion. You're not limited to slide presentations, of course. Be creative. Propose something fun to share with people so they might feel some of your enthusiasm for Ada!

Speaking slots are either 15 or 45 minutes, plus 5 minutes for Q&A. Depending on interest, we might also have a session with lightning presentations (e.g. 5 minutes each).

Note that all talks will be streamed live (audio+video) and recorded, for remote as well as later viewing of talks, and so that people can watch streams in the hallways when rooms are full. By submitting a proposal, you agree to being recorded and streamed, and agree the content of your talk will be published under the same license as all FOSDEM content, a Creative Commons (CC-BY) license.

Submission Guidelines

Subscribe to the Ada-FOSDEM mailing list [13], and submit your proposal there. If needed, feel free to contact us at ada-belgium@cs.kuleuven.be.

Please include:

- your name, affiliation, contact info;
- the title of your talk (be descriptive and creative);
- a short descriptive abstract;
- potentially pointers to more information;
- a short bio and photo.

See programs of previous Ada DevRooms (URLs below) for presentation examples, as well as for the kind of info we need.

We'd like to put together a draft schedule by early December. So, please act ASAP, and definitely by Saturday December 1, 2018 at the latest.

We look forward to lots of feedback and proposals!

Ada-Europe 2019

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: 2 Dec 2018 11:22 -0000
Subject: CfP Ada-Europe 24th Conf. on Reliable Software Technologies
Newsgroups: comp.lang.ada

Call for Papers

Ada-Europe 24th International Conference on Reliable Software Technologies (Ada-Europe 2019)

10-14 June 2019, Warsaw, Poland

http://www.ada-europe.org/conference2019

Organized by EDC and Ada-Europe, in cooperation with ACM SIGAda, SIGBED (pending), SIGPLAN (pending) and the Ada Resource Association (ARA)

General Information

Ada-Europe is pleased to announce that its 24th International Conference on Reliable Software Technologies (Ada-Europe 2019) will take place in Warsaw, Poland, in the week of 10-14 June.

The conference schedule at its fullest includes a three-day technical program and vendor exhibition from Tuesday to Thursday, and parallel tutorials and workshops on Monday and Friday.

This edition of the conference inaugurates a major revamp in the registration fees, redesigned to extend participation from industry and academia, and to reward contributors, especially but not solely students and post-doc researchers.

[More information in the forthcoming events section of the Journal.]

Ada-related Resources

New SPARK/Ada Blog

From: apemant...@gmail.com
Date: Thu, 27 Sep 2018 02:51:41 -0700
Subject: New SPARK/Ada Blog
Newsgroups: comp.lang.ada

A new SPARK/Ada blog (by me)

Only just started, so not much there, yet: its focus will be developing commercial applications (i.e. competing with C# etc)

Feel free to pop over and take a look

http://www.apemantus.co.uk

From: joak...@kth.se
Date: Thu, 27 Sep 2018 04:17:50 -0700
Subject: Re: New SPARK/Ada Blog
Newsgroups: comp.lang.ada

Great! Now many SPARK/Ada blogs around. Looking forward to more articles!

From: Henrik Härkönen <heha...@gmail.com>
Date: Thu, 27 Sep 2018 07:52:59 -0700
Subject: Re: New SPARK/Ada Blog
Newsgroups: comp.lang.ada

Nice!

I'm very, very new to Ada, but I've just recently picked up an interest for it. I'm always on the lookout for "new" (to me) programming languages, but it's quite much just window shopping. But this time, I think I've come across with something that I'd actually like to learn. Seems that your blog will be something that I'll keep my eyes on! :)

From: Pascal Obry <pas...@obry.net>
Date: Thu, 27 Sep 2018 17:36:53 +0200
Subject: Re: New SPARK/Ada Blog
Newsgroups: comp.lang.ada

If you are new to Ada and want to learn it, be sure to also have a look here:

https://learn.adacore.com

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Thu, 27 Sep 2018 17:30:14 -0500
Subject: Re: New SPARK/Ada Blog
Newsgroups: comp.lang.ada

> [...]

And of course here:

http://www.adaic.org/learn

Ada-related Tools

GNAT for AVR

From: ada.ne...@gmail.com
Date: Sun, 16 Sep 2018 19:22:45 -0700
Subject: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

I'm programming an ATmega328P with the GNAT compiler for AVR.

I need to use the Sqrt, Arctan and Atan2 functions. But maybe in the future I will need to use some more.

I don't have access to the regular Ada.Numerics package.

By now, I wrote the Sqrt using the Newton's method and Arctan using Taylor series.

I would like to know if there is a better way to use/implement mathematical functions (maybe import them from C?).

I really appreciate any help.

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Mon, 17 Sep 2018 09:18:06 +0200
Subject: Re: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

> By now, I wrote the Sqrt using the Newton's method and Arctan using Taylor series.

Chebyshev's polynomials for arctan?

From: Simon Wright <si...@pushface.org>
Date: Mon, 17 Sep 2018 13:22:05 +0100
Subject: Re: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

For what it's worth, the FSF GCC arm-eabi compiler/runtime that I built imports the basic maths functions from the C library. This may be because I built the C library (newlib) first?

The GNAT CE arm-eabi compiler goes to basics with very deep-looking code; possibly because it's "the Ada Cert Math specific version" (from s-libsin.ads), i.e. one with which AdaCore will support customers with certification requirements.

From: Bill Findlay <findl...@blueyonder.co.uk>
Date: Mon, 17 Sep 2018 18:28:17 +0100
Subject: Re: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

Since you have sqrt, see: http://www.findlayw.plus.com/KDF9/#ATN

From: Aurele Vitali <aurele...@gmail.com>
Date: Mon, 17 Sep 2018 15:41:33 -0700
Subject: Re: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

I don't know anything about the ATmega328P, but if it uses a builtin floating point processor, you can use Ada inline floating point assembler. It`s not hard to do. Here is a simple example of the square root function:

```
with System.Machine_Code;
use System.Machine_Code;
EOL : constant String :=
        ASCII.LF & ASCII.HT;
```

```
function Sqrt( x : in Long_Float ) return
Long_Float is
  Result : Long_Float := 0.0;
begin
  Asm(( "fldl  %1" & EOL & -- Load x in St(0)
     "fsqrt" & EOL & -- Take the Sqrt of St(0)
     "fstpl %0" ), -- Store result and pop St(0)
      Outputs => ( Long_Float'Asm_Output(
                   "=m", Result ) ),
      Inputs  => ( Long_Float'Asm_Input (
                   "m",      x ) ) );
  return Result;
end Sqrt;
```

You can do the same thing for trig functions... Just cut and paste and see if this example works.

From: rakusu...@fastmail.jp
Date: Mon, 17 Sep 2018 18:16:59 -0700
Subject: Re: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

It is always be bottleneck - doing math on slow integer 8-bit CPU with tiny amount of memory and instructions only for addition and subtraction,- because it produce a huge pieces of slow machine code. So all math there are doing in integers by a table calculations, adds and shifts. Therefore in future it's better to go away from AVR for you with math, I think.

Btw, there is a quite old C-runtime library for AVR at http://savannah.nongnu.org/projects/avr-libc/ - it might be helpful for you, especially its handwritten libm. It can also be useful to look at CORDIC algorithm, frex http://www.dcs.gla.ac.uk/~jhw/cordic/inzex.html

From: R R <rrr.e...@gmail.com>
Date: Tue, 25 Sep 2018 00:14:45 -0700
Subject: Re: GNAT for AVR - Mathematical Functions
Newsgroups: comp.lang.ada

The old AVR-ADA project never supported floating point math. The AVR 8bit processors are not made for that, even though you can use float and double in Arduino. The AVR compiler by Adacore from around 2011 did support floating point variables as far as I remember, I am not sure about the math functions.

XNAdaLib

From: Pascal Pignard <p....@orange.fr>
Date: Fri, 21 Sep 2018 07:57:18 +0200
Subject: [ANN] XNAdaLib 2018 binaries for High Sierra including GTKAda and more.
Newsgroups: comp.lang.ada

This is XNAdaLib 2018 built on macOS 10.13 High Sierra for Native Quartz with GNAT Community 2018 including:

- GTKAda 18.0w mid-2018 (www.adacore.com/gtkada) with GTK+ 3.22.29 (www.gtk.org) complete,
- Glade 3.22.1 (glade.gnome.org),
- GnatColl mid-2018 (github.com/AdaCore/gnatcoll),
- Florist mid-2018a (www.cs.fsu.edu/~baker/florist.html),
- AdaCurses 20110404 (invisible-island.net/ncurses/ncurses-Ada95.html),
- Gate 3.05-b (sourceforge.net/projects/lorenz),
- Components 4.30 (www.dmitry-kazakov.de/ada/components.htm),
- AICWL 3.19 (www.dmitry-kazakov.de/ada/aicwl.htm),
- Zanyblue 1.4.0 (zanyblue.sourceforge.net),
- PragmARC mid-2018 (pragmada.x10hosting.com/pragmarc.htm),
- GNOGA 1.4-beta (www.gnoga.com),
- AdaControl 1.19r10 (adalog.fr/fr/adacontrol.html),
- AdaDep 1.4r1 (adalog.fr/fr/composants.html),
- AdaSubst 1.5r1 (adalog.fr/fr/composants.html),
- SparForte 2.2-180916 (sparforte.com),

and as side libraries:

- Template Parser 19.0,
- gtksourceview 3.24.4,
- GNUTLS 3.5.9,
- ASIS GPL 2018,
- SDL 1.2.15 et SDL_Image 1.2.12.

XNAdaLib binaries have been post on Source Forge:

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2018-high-sierra/

Feel free to send comments.

Report preferably all comments to MacAda.org mailing list:

http://macada.org/macada/Contacts.html

See list archive:

https://hermes.gwu.edu/archives/gnat-osx.html

Third-party library management

From: Henrik Härkönen <heha...@gmail.com>
Date: Tue, 9 Oct 2018 00:03:48 -0700
Subject: Per-project third party library management
Newsgroups: comp.lang.ada

What would be the most convenient way to manage (mostly install & upgrade) a third party library for one's project?

With python projects I'd probably fire up an virtualenv per project and install stuff there with pip etc. With scala I'd use SBT to handle the libraries. In my C development days, we had a proprietary RTOS with all of its dependencies managed by someone else, so I didn't have to (or get to learn) worry about those personally... :(

As I've understood, Ada doesn't have such a package distribution system, so one would typically download sources, compile and install the library through its make system. Or install readily packaged version of the library, for example with apt-get etc.

So far, I've installed AWS from source and Ahven from a DEB package, and as such they are just fine methods. But both were using root access and system wide install.

What I'm aiming for with my question, is to learn a way to install a library so that it would not require root access and it would be more tied to the project.

Different projects might need to use different versions etc.

Should I use GNU Stow, or configure --prefix to point somewhere under my project tree and have gprbuild include it from there, or...?

My apologies if this has been asked a lot, but at least I didn't find that many directly related discussions about this.

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Tue, 9 Oct 2018 09:56:14 +0200
Subject: Re: Per-project third party library management
Newsgroups: comp.lang.ada

> What I'm aiming for with my question, is to learn a way to install a library so that it would not require root access and it would be more tied to the project.

If you don't want to follow the rules imposed by the OS, there is no such thing as "install" anymore. Simply copy the library file where you want it to be.

Specifically for GNAT Ada there is a ready-to-use tool gprinstall which does installation of Ada projects. A third-party library can be described as a separate (externally built) library project. I didn't try it, but I suppose it must work as expected.

But again, for anything beyond simplest stuff you have no choice but to use the corresponding packaging tool of the corresponding OS, however painful, like in the case of DEB and RPM, it might be.

From: Henrik Härkönen <heha...@gmail.com>
Date: Tue, 9 Oct 2018 04:57:06 -0700
Subject: Re: Per-project third party library management
Newsgroups: comp.lang.ada

Ok, I have to tinker with these options and see what seems like the best option.

The gprinstall at least seems nice in that way that it would reduce some manual and error prone steps on the way, perhaps.

Gnoga Gallery

From: Pascal Pignard <p....@orange.fr>
Date: Sat, 13 Oct 2018 10:04:06 +0200
Subject: [ANN] Gnoga gallery.
Newsgroups: comp.lang.ada

Please find on Gnoga Wiki (https://sourceforge.net/p/gnoga/wiki/), the Gnoga Gallery of demonstration, tutorial programs and more...

It consists for each Gnoga app, demo, test, etc to show a screen capture with the list of the main Gnoga components used by the program.

The corresponding source code is available with the link source code.

Many of them are online with the link "Try it online" to the corresponding app.

Maybe you'll find some down, I'll look after them from time to time, I apologize if they are not all OK.

Maybe you'll find some slow, they are hosted on a VM from AWS and some work has to do to reduce latency.

https://sourceforge.net/p/gnoga/wiki/Gnoga-Gallery

Feel free to send your feedback on Gnoga list:

https://sourceforge.net/p/gnoga/mailman/gnoga-list/

Feel free to send your own Gnoga program screen capture, the Gnoga components used by the program and the program web site link.

I will add it to the gallery.

From: Henrik Härkönen <heha...@gmail.com>
Date: Sun, 14 Oct 2018 12:17:10 -0700
Subject: Re: [ANN] Gnoga gallery.
Newsgroups: comp.lang.ada

> [...]

Cool, thanks for the link and gallery! Just what I need, I'm slowly getting into Gnoga as well as Ada itself. :)

Gnoga

From: Pascal Pignard <p....@orange.fr>
Date: Sat, 20 Oct 2018 18:49:00 +0200
Subject: [ANN] Gnoga version 1.4a and 1.5-alpha.
Newsgroups: comp.lang.ada

Gnoga version 1.4a has been released on SF GIT:

https://sourceforge.net/p/gnoga/code/ci/dev_1.4/tree/

and on SF files as zipped source code:

https://sourceforge.net/projects/gnoga/files/

See HISTORY for details:

https://sourceforge.net/p/gnoga/code/ci/dev_1.4/tree/HISTORY

Then new branch dev_1.5 has been created to collect new Gnoga 1.5-alpha developments, see TODO:

https://sourceforge.net/p/gnoga/code/ci/dev_1.5/tree/TODO

Contributors are welcome.

Feel free to report detailed issues on Gnoga list or create tickets on SF:

https://sourceforge.net/p/gnoga/mailman/

https://sourceforge.net/p/gnoga/tickets/

Strings Edit v3.4

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Tue, 6 Nov 2018 22:21:41 +0100
Subject: ANN: Strings Edit v3.4
Newsgroups: comp.lang.ada

The package Strings_Edit provides I/O facilities:

- Generic axis scales support;
- Integer numbers (generic, package Integer_Edit);
- Integer sub- and superscript numbers;
- Floating-point numbers (generic, package Float_Edit);
- Roman numbers (the type Roman);
- Strings;
- Ada-style quoted strings;
- Base64 encoding;
- RFC 8439 (ChaCha20 cipher, Poly1305 digest, AEAD);
- UTF-8 encoded strings and conversions to older encoding standards;
- Unicode maps and sets;
- Wildcard pattern matching.

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

Changes to the previous version:

- The package Strings_Edit.UTF8.Windows_1250 provides Windows-1250 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1251 provides Windows-1251 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1252 provides Windows-1252 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1253 provides Windows-1253 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1254 provides Windows-1254 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1255 provides Windows-1255 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1256 provides Windows-1256 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1257 provides Windows-1257 encoding conversions;
- The package Strings_Edit.UTF8.Windows_1258 provides Windows-1258 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_2 provides ISO/IEC 8859-2 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_3 provides ISO/IEC 8859-3 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_4 provides ISO/IEC 8859-4 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_5 provides ISO/IEC 8859-5 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_6 provides ISO/IEC 8859-6 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_7 provides ISO/IEC 8859-7 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_8 provides ISO/IEC 8859-8 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_9 provides ISO/IEC 8859-9 encoding conversions;
- The package Strings_Edit.UTF8.ISO_8859_10 provides ISO/IEC 8859-10 encoding conversions;
- The package Strings_Edit.UTF8.KOI8 provides KOI8 encoding conversions;
- The package Strings_Edit.UTF8.MacOS_Roman provides Mac OS Roman encoding conversions;
- The package Strings_Edit.UTF8.RADIX50 provides DEC RADIX-50 encoding conversions;
- The package Strings_Edit.UTF8.Recoding_Streams provides streams recoding into/from UTF-8.

Simple Components

From: Dmitry A. Kazakov <mai...@dmitry-kazakov.de>
Date: Wed, 7 Nov 2018 18:08:20 +0100
Subject: ANN: Simple components v4.31
Newsgroups: comp.lang.ada

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

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

Changes the previous version:

- Bug fix in GNAT.Sockets.MQTT. Server.Push, a check for the case when no client is connected.

Industrial Control Widget Library

From: Dmitry A. Kazakov <mai...@dmitry-kazakov.de>
Date: Thu, 8 Nov 2018 18:06:11 +0100
Subject: ANN: Ada industrial control widget library v3.20
Newsgroups: comp.lang.ada

Intended for design high-quality industrial control widgets for Ada applications. The software is based on GtkAda, Ada bindings to GTK+ and cairo. The key features of the library:

- Widgets composed of transparent layers drawn by cairo;
- Fully scalable graphics;
- Support of time controlled refresh policy for real-time and heavy-duty applications;
- Caching graphical operations;
- Stream I/O support for serialization and deserialization;
- Ready-to-use gauge, meter, oscilloscope widgets;
- Editor widget for WYSIWYG design of complex dashboards.

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

Changes to the previous version:

- Bug fix in the package Gtk.Layered.Waveform that causes wrong time conversion caused daylight saving shift.

Ada-related Products

AdaControl ASIS Exception

From: Markus Schöpflin <no.spam@spam.spam>
Date: Fri, 12 Oct 2018 10:23:24 +0200
Subject: ASIS exception with AdaCtl v1.19r10
Newsgroups: comp.lang.ada

Given the following simple test program:

```
procedure TEST
is
   package A is
     type T is new FLOAT;
   end A;

   function F return A.T'BASE
   is
   begin
     return 0.0;
   end F;

begin
   null;
end TEST;
```

Checking this program with AdaCtl gives the following exception:

```
 > adactl -l "check style(no_closing_name)"
test
=========Phase: Processing ==========
AdaCtl version: 1.19r10 with ASIS 2.0.R for
GNAT Pro 7.4.2 (20160527)
ASIS error: ASIS.EXCEPTIONS.
ASIS_INAPPROPRIATE_ELEMENT
   In rule: STYLE
  For unit: TEST
Status   : VALUE_ERROR
Diagnosis: Inappropriate Element Kind in
Asis.Expressions.Corresponding_Name_Dec
laration (A_SELECTED_COMPONENT)
called in Actual procedure for Pre_Operation
with the argument :
A_FUNCTION_BODY_DECLARATION
located in TEST (body, Unit_Id = 2,
Context_Id = 1)
text position : test.adb:7:4
   Nodes:
     Node           : 2315 -
           N_SUBPROGRAM_BODY
     R_Node         : 2315 -
           N_SUBPROGRAM_BODY
     Node_Field_1    : 0 - N_EMPTY
     Node_Field_2    : 0 - N_EMPTY
   Rel_Sloc          : 65
   obtained from the tree test.adt
(Tree_Id = 1)
called in Asis.Iterator.Traverse_Element
with the argument :
A_FUNCTION_BODY_DECLARATION
located in TEST (body, Unit_Id = 2,
Context_Id = 1)
text position : test.adb:7:4
   Nodes:
     Node           : 2315 -
           N_SUBPROGRAM_BODY
     R_Node         : 2315 -
           N_SUBPROGRAM_BODY
     Node_Field_1    : 0 - N_EMPTY
     Node_Field_2    : 0 - N_EMPTY
  Rel_Sloc          : 65
obtained from the tree test.adt
(Tree_Id = 1)
called in Actual procedure for Pre_Operation
with the argument :
A_PROCEDURE_BODY_DECLARATION
located in TEST (body, Unit_Id = 2,
Context_Id = 1)
text position : test.adb:1:1
   Nodes:
     Node           : 2279 -
           N_SUBPROGRAM_BODY
     R_Node         : 2279 -
           N_SUBPROGRAM_BODY
     Node_Field_1    : 0 - N_EMPTY
     Node_Field_2    : 0 - N_EMPTY
   Rel_Sloc          :-10
   obtained from the tree test.adt
           (Tree_Id = 1)
called in Asis.Iterator.Traverse_Element
with the argument :
A_PROCEDURE_BODY_DECLARATION
located in TEST (body, Unit_Id = 2,
Context_Id = 1)
text position : test.adb:1:1
   Nodes:
     Node           : 2279 -
           N_SUBPROGRAM_BODY
     R_Node         : 2279 -
           N_SUBPROGRAM_BODY
     Node_Field_1    : 0 - N_EMPTY
     Node_Field_2    : 0 - N_EMPTY
   Rel_Sloc          :-10
   obtained from the tree test.adt
   (Tree_Id = 1)
```

Is there anything wrong with my test, or am I facing an error in either the ASIS implementation or AdaCtl here?

From: "J-P. Rosen" <ro...@adalog.fr>
Date: Fri, 12 Oct 2018 14:25:08 +0200
Subject: Re: ASIS exception with AdaCtl v1.19r10
Newsgroups: comp.lang.ada

> […]

Excellent occasion to remind everybody that there is a bug tracking system for AdaControl at https://sourceforge.net/p/adacontrol/tickets/

By all means, please report any issue you may have!

As for this particular case, I'll have a look at it. Please write to me directly if you want a personalized answer, since you anonymized your address.

(Hmmm... BTW GnatPRO 7.4.2 is quite old, did you try with a more recent one?)

From: Markus Schöpflin <no.spam@spam.spam>
Date: Fri, 12 Oct 2018 14:37:30 +0200
Subject: Re: ASIS exception with AdaCtl v1.19r10
Newsgroups: comp.lang.ada

> [...]

> By all means, please report any issue you may have!

I will, once I'm convinced that AdaControl is at fault here. As I'm not sure whether the ASIS library or AdaControl is the culprit, I decided to ask here first.

> As for this particular case, I'll have a look at it. Please write to me directly if you want a personalized answer, since you anonymized your address.

Thanks. Will do.

> (Hmmm... BTW GnatPRO 7.4.2 is quite old, did you try with a more recent one?)

Not yet, will do, thanks for the hint. (BTW, GNAT Pro 7.4.2 has been released somewhen in summer 2016, that's only two years ago. Not what I would consider old, at least not for an Ada compiler.)

From: "J-P. Rosen" <ro...@adalog.fr>
Date: Fri, 12 Oct 2018 14:50:17 +0200
Subject: Re: ASIS exception with AdaCtl v1.19r10
Newsgroups: comp.lang.ada

This is fixed now in the wavefront version of AdaControl.

Simple reproducer, easy to fix... Thanks.

Strange, it seems that nobody ever used a selected name with a 'BASE attribute before...

From: Markus Schöpflin <no.spam@spam.spam>
Date: Fri, 12 Oct 2018 15:25:10 +0200
Subject: Re: ASIS exception with AdaCtl v1.19r10
Newsgroups: comp.lang.ada

> This is fixed now in the wavefront version of AdaControl.

That was fast, thanks. So my other post has become obsolete. Will the GIT repository over at Sourceforge contain the fix?

> Simple reproducer, easy to fix... Thanks.

Just for the sake of completeness: I tried with GNAT Pro 18.2 and got the same error as with GNAT Pro 7.4.2.

[...]

From: "J-P. Rosen" <ro...@adalog.fr>
Date: Fri, 12 Oct 2018 15:35:22 +0200
Subject: Re: ASIS exception with AdaCtl v1.19r10
Newsgroups: comp.lang.ada

> That was fast, thanks. So my other post has become obsolete. Will the GIT repository over at Sourceforge contain the fix?

I don't update the Sourceforge very frequently, especially because I like to be able to cancel or rework my local GIT tree, and you can't do that anymore once you have pushed something.

Of course, supported users have beta versions as soon as they ask for...

GNAT Modification_Time Limitation

From: Lionel Draghi <lionel...@gmail.com>
Date: Mon, 19 Nov 2018 14:56:29 -0800
Subject: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

I am coding a kind of make application, that depends on file's time tag (thanks to Ada.Directories.Modification_Time), and on Ada.Calendar.Clock, both returning Ada.Calendar.Time.

Unfortunately, I came across a GNAT limitation in the Modification_Time implementation on Linux : sub-second are ignored, and Modification_Time returns

> Time_Of (Year, Month, Day, Hour, Minute, Second, 0.0);

So, at the same time Clock returns 2018-10-29 20:36:01.47 while Modification_ Time returns 2018-10-29 20:36:01.00.

This prevents me from knowing if a file is modified before or after certain time, and thus undermine my efforts.

My workaround was to impair also Clock precision, with an ugly rounding:

```
> Time := Ada.Calendar.Clock;
> New_Time := Time_Of
>   (Year    => Year (Time),
>    Month   => Month (Time),
>    Day     => Day (Time),
>    Seconds => Day_Duration
   (Float'Floor (Float (Seconds (Time)))));
```

But that's not a correct solution either : I have to order lots of file creation, and having all files created during the same second returning the same time tag also prevent my algorithm from properly working.

Any workaround to get a precise file time tag?

Or to compare file's time tag with Clock?

From: Shark8 <onewing...@gmail.com>
Date: Mon, 19 Nov 2018 16:47:46 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

The problem with using the filesystem timestamp is that its resolution is too coarse compared to the processing-speed of your CPU.

I would recommend either implementing some sort of controlled cache, version-control, or 'hacking' the timestamp so that it's a really a build-number (eg Build 1 -> 01 Jan 1900, build 2 -> 02 Jan 1900, build 35 -> 04 Feb 1900, etc).

From: Keith Thompson <ks...@mib.org>
Date: Mon, 19 Nov 2018 17:33:04 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> [...]

It's odd that GNAT's Modification_Time truncates the time to one-second precision. A quick experiment on my system (Ubuntu 18.04) also indicates that it does so, even though the system stores the timestamp in nanosecond precision.

On Linux 2.6 and later, the underlying stat() system call gives you a "struct timespec" value for the modification time, as specified by the current POSIX standard. (struct timespec represents times with nanosecond precision.) A file system isn't required to store times with that precision, but many do.

If you're on a POSIX system, you should be able to call the stat() system call and *probably* get a more precise timestamp.

If you're on a non-POSIX system, there might still be a system-specific way to get a more precise timestamp. (NTFS also seems to store timestamps with high precision.)

(And remember that nanosecond precision doesn't necessarily imply nanosecond accuracy.)

From: Keith Thompson <ks...@mib.org>
Date: Mon, 19 Nov 2018 17:33:40 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> The problem with using the filesystem timestamp is that its resolution is too coarse compared to the processing-speed of your CPU.

That depends on the filesystem. See my other followup in this thread.

From: briot.e...@gmail.com
Date: Tue, 20 Nov 2018 00:08:52 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> I am coding a kind of make application, that depends on file's time tag (thanks to Ada.Directories.Modification_Time), and on Ada.Calendar.Clock, both returning Ada.Calendar.Time.

Interesting. I am in the middle of a discussion with AdaCore about gprbuild, which fails to recompile when using an alternative body that happens to have the same time stamp (to the second). gprbuild sees that the modification time appears to be the same, and thus doesn't recompile.

Two points:

- AdaCore mentioned they made progress recently on timestamp precision and it would likely fix the scenario. I think this is similar to what you reported, so it is likely your issue has been fixed now.

- I am arguing with AdaCore that checking timestamps is not enough (might not even be useful at all), as Shark8 mentioned. The scenario I have is the following:

> Create a project with one scenario variable. Depending on that variable, chose src1 or src2 for source dirs. In each of these directories, have a file utils.adb with a different content. "touch" these two files so that they have the same timestamp. If you build your application once with one value of the variable, then rebuild with another value, gprbuild does nothing the second time.

I had a similar real case because git created two files with the same timestamp. And then it took me days to understand why some of my tests appeared to be linked with both versions of utils.adb, since I could see in the log file traces from both src1/utils.adb and src2/utils.adb.

Very very confusing.

So I would indeed recommend that you don't bother with timestamps, and only look at file contents (or use timestamp+file path at the very least, or perhaps inodes).

I am interested in hearing more why you want to code a new 'make-like'?

Now trying to persuade AdaCore that gprbuild's behavior is incorrect...

From: Lionel Draghi <lionel...@gmail.com>
Date: Tue, 20 Nov 2018 03:57:42 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

Thank you guys for your answers:

@Shark: see the description of my app hereafter, I will try the simple way first :-)

@Keith and Emmanuel: the Time_Of call I put in my message comes from the body of Ada.Directories (/opt/GNAT/2018/lib/gcc/x86_64-pc-linux-gnu/7.3.1/adainclude/a-direct.adb)

…

```
Date := File_Time_Stamp (Name);
GM_Split (Date, Year, Month, Day, Hour,
Minute, Second);
return Time_Of (Year, Month, Day, Hour,
Minute, Second, 0.0);
```

…

and GM_Split (in System.OS_Lib package) is calling

```
procedure To_GM_Time
  (P_Time_T : Address; P_Year   : Address;
   P_Month  : Address;  P_Day   : Address;
   P_Hours  : Address;  P_Mins  : Address;
   P_Secs   : Address);
    pragma Import (C, To_GM_Time,
"__gnat_to_gm_time");
```

P_Secs is pointing an Integer.

So the limitation seems to come from GNAT C interface to OS lib.

@Keith: my App is (in this first version) using strace, so thanks for the stat idea, I should directly get the OS time stamp from strace output.

@Emmanuel: my make is a POC to do a make without makefile! :-) it runs command and observes files accesses (thanks to linux kernel ptrace interface), and automatically understand what files it depends on, and what files are output.

My first test case is to replace this Makefile:

```
all: hello
hello.o: hello.c
   gcc -o hello.o -c hello.c
main.o: main.c hello.h
   gcc -o main.o -c main.c
hello: hello.o main.o
   gcc -o hello hello.o main.o
```

with just :

```
gcc -o hello.o -c hello.c
gcc -o main.o -c main.c
gcc -o hello hello.o main.o
```

and to get the same optimized behavior when removing a .o file or touching one of the source files.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Tue, 20 Nov 2018 17:32:21 -0600
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

NTFS has three timestamps (modification, creation, and last access). Only the modification has high precision; the others are only good to full seconds (or something like that).

FAT file systems (as you might encounter on a camera or USB stick) only have precision to 2 seconds. (Which is why we had to deal with this in the Janus/Ada build tools fairly early on.)

Also note that the system clock on Windows systems typically only changes every 0.01 sec (Dmitry says this can be changed, although I've never seen that done). That extends to the file systems and other OS timers as well.

Most Ada vendors use a Ada.Calendar.Clock that blends the system clock with the high performance timer to get useful accuracy of Ada.Calendar.Time.

(A customer/collaborator, Tom Moran, originally wrote that code the Janus/Ada implementation of Calendar to fix some timing problem that he had. He eventually submitted similar code to AdaCore who added it to their Calendar as well.)

Moral: Doing "Make" on a modern machine, especially if you want it to be portable, is a tricky job.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Tue, 20 Nov 2018 17:53:58 -0600
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> [...]

I wouldn't claim that the situation is that dire; it seems to be related to the particular implementation of a particular GNAT feature (project scenario variables). If you're not implementing something where the source code location can be changed for a particular build, then timestamps will work (but you have to remember that they are quite granular).

It also seems to be related in part of source-based compilation (which necessarily keeps less information between builds). In a Janus/Ada project (which is very different than a GNAT project -- it's a binary DB-like file of compilation information), changing the location of a source file would invalidate the entire entry and essentially delete any existing compilations. More likely, however, is that a scenario would be set up using separate project files (most likely using Windows batch files/Unix shell-scripts to automate), so each would have their own set of compilation states. And it's completely impossible to bind multiple versions of a unit into a single executable; only one or the other could be selected - and if somehow some files were compiled against the wrong one, some or all of the compilation timestamps wouldn't match (which would cause binding failure).

The moral here is how to implement a Make-like tool depends a lot on what capabilities it will have.

From: briot.e...@gmail.com
Date: Tue, 20 Nov 2018 23:31:11 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

The trick of course is to define what a "build" is in your sentence.

If it is one execution of the builder (gprbuild, make,...) then I think it is indeed a reasonable assertion.

If however a build is defined to something that amount to "in debug mode, in production mode,..." then of course it might happen that the sources are changed and the timestamp have a timestamp delta of less than 1s (when we generate code for instance).

Furthermore, the actual scenario was the following: in the automatic tests, I need to simulate the connection to the database, so that means I need to have support for alternate bodies (but I still compile in debug mode, or production mode,...). Is that still the same "build" ?

I would guess it is, but in the end we would end up with literally dozens of "build" types, each with its own set of object files, and each taking 20 or 30 minutes to build from scratch. Not realistic for continuous testing.

I spent some time looking around at general builder tools around. Most of them seem to advertise nowadays that they look at file contents, not timestamps. I started from the list at https://en.wikipedia.org/wiki/List_of_build_automation_software, and looked at a few of them.

> It also seems to be related in part of source-based compilation (which necessarily keeps less information between builds). In a Janus/Ada project (which is very different than a GNAT project -- it's a binary DB-like file of compilation information), changing the location of a source file would invalidate the entire entry and essentially delete any existing compilations. More likely, however, is that a scenario would be set up using separate project files (most likely using Windows batch files/Unix shell-scripts to automate), so each would have their own set of compilation states.

That's more or less what gprbuild does in practice. It uses a "distributed database" via the .ALI files, which are found in the object directories, so for best use each "build" should have a different object directories. And we are again hitting the notion of "build".

> And it's completely impossible to bind multiple versions of a unit into a single executable; only one or the other could be selected

That's indeed one of the ways gprbuild could detect the error. To me it is a bug in gprbuild that it allows linking different files for the same unit into the same executable.

> somehow some files were compiled against the wrong one, some or all of the compilation timestamps wouldn't match (which would cause binding failure).

timestamps are not reliable enough, especially on modern fast machines. I am pretty sure you will hit a similar issue I had, one day.

From: briot.e...@gmail.com
Date: Tue, 20 Nov 2018 23:40:06 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> @Emmanuel : my make is a POC to do a make without makefile! :-)

> it runs command and observes files accesses (thanks to linux kernel ptrace interface), and automatically understand what files it depends on, and what files are output.

There was an article earlier this week on reddit about `redo`, which seems to have a similar idea of top-down compilation: you have a linker script that tells redo it needs a.o, b.o and c.o (then redo recursively processes those), and finally does the link.

In turn, for a.o you would tell redo it needs a.ads, a.adb and b.ads, and then compile,...

With your idea of using ptrace, that would be an automatic way maybe to tell redo about the dependency graph.

I am not sure redo would be really usable on actual projects though. You have to list the dependencies for the linker for instance (I much prefer the gprbuild approach of finding those automatically).

A similar limitation seems to exist in your POC: how do I, as a novice user, know what to compile in the first place? It seems you would need a combination of what gprbuild does, with ptrace:

- compile (with ptrace) the main unit.

- gprbuild then uses the ALI file to find the dependencies, and check those recursively.

- in your case, you would instead look at the ptrace output to find those dependencies.

The ptrace approach would be much more reliable (though linux-specific), since you would know for instance:

- that the compiler searched and did not find foo,ads in /first/dir

- found and opened /other/dir/foo.ads

so next time there is a build you can check first whether 'foo.ads' now exists in /first/dir. If that file now exists, you need to rebuild.

gprbuild doesn't handle such changes on the system, it only store what it found.

(this is all an interesting concept I learned this week from `redo`)

Let us know the result of the experiment!

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Wed, 21 Nov 2018 09:23:05 +0100
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> Also note that the system clock on Windows systems typically only changes every 0.01 sec (Dmitry says this can be changed, although I've never seen that done).

The API call is timeBeginPeriod

https://docs.microsoft.com/en-us/windows/desktop/api/timeapi/nf-timeapi-timebeginperiod

The time resolution could be set down to 1ms (and never call timeEndPeriod as the page suggests (:-))

> Moral: Doing "Make" on a modern machine, especially if you want it to be portable, is a tricky job.

Yes, especially because the OS on the modern machine tends to deploy worst possible time source available. I guess that some MS-DOS code still does that job on your i9 ...

From: briot.e...@gmail.com
Date: Wed, 21 Nov 2018 03:16:10 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

Slightly out of topic (sorry): I found tup (http://gittup.org/tup/index.html) which appears to be doing exactly what you want to achieve. It monitors file accesses but it uses a fuse filesystem for this, rather than ptrace.

I had implemented a fuse filesystem in Ada at some point, though I do not have that code anymore. AdaCore was using that to access a database that contains all build+tests results on all possible combinations, if I remember right.

From: Shark8 <onewing...@gmail.com>
Date: Wed, 21 Nov 2018 06:38:37 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

I read that as https://en.wikipedia.org/wiki/List_of_build_abomination_software and had to do a double take.

From: Simon Wright <si...@pushface.org>
Date: Wed, 21 Nov 2018 17:32:48 +0000
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> That's more or less what gprbuild does in practice. It uses a "distributed database" via the .ALI files, which are found in the object directories, so for best use each "build" should have a different object directories. And we are again hitting the notion of "build".

Ideally, each distinct set of scenario variable values should have its own object directory. Will take a lot of time for the initial compilations, of course.

From: briot.e...@gmail.com
Date: Wed, 21 Nov 2018 09:43:56 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> Ideally, each distinct set of scenario variable values should have its own object directory. Will take a lot of time for the initial compilations, of course.

That's actually more than that. We already use the above (and indeed we have like 5 or 6 major scenarios, thankfully we do not compile quite all the possible combinations).

But in the context of tests, we use extending projects to override some of the sources (for instance so that we do not have to actually have a database running). The test project itself is an extending-all.

So if you have the simple case:

a.gpr imports b.gpr imports c.gpr imports d.gpr

and need to substitute a body for a file c.adb in C. you then extend that project, and make a2.gpr an extending-all project, thus we now have:

a.gpr imports b.gpr imports c.gpr imports d.gpr

|

a2.gpr imports b.gpr imports c2.gpr imports d.gpr

The scenario variables have not changed, so b's objects will go in the 'obj-production' directory as before, for instance. But in fact, some of object files now depend on that alternate body of c.adb. If you had some inlined subprograms in c.adb (using -gnatn), then part of their code is in b.o.

In the common (and optimistic) case where c.adb has a different timestamp from before, b.o will be recompiled and all is fine.

If c.adb has the same timestamp as the original file (because, hey, git does what it wants), gprbuild doesn't notice the change in c.adb, so doesn't recompile b.o, and when we link the executable we go some case from the old c.adb (the inlined code).

This is why just checking the timestamp is not (cannot) be good enough.

Ideally, we should try and use a different object directory here (though the scenario is the same), but I don't know how to do that (b.gpr hasn't changed, thanks to the extend-all project).

And if you add to the original 5 scenario variables another case where you can potentially mock any number of project, you end up with way too many combinations of object directories, my disk would not be big enough I think.

From: Lionel Draghi <lionel...@gmail.com>
Date: Wed, 21 Nov 2018 11:02:35 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> [...]

> With your idea of using ptrace, that would be an automatic way maybe to tell redo about the dependency graph.

Exactly, the idea of the POC is see how far we can go without any explicit description of the dependency graph, or whatever build recipes.

...

> A similar limitation seems to exist in your POC: how do I, as a novice user, > know what to compile in the first place?

It's not in my scope: I don't target making easier compilations (I don't pretend doing a better job than gprbuild or so), just running smartly a list of command.

I used a C compilation example as it's a classical make example, but it could be whatever suite of command:

latex <file>.tex

dvips <file>.dvi

ps2pdf <file>.ps

pdf2eps <pagenumber> <file>

And gprbuild, or even a complex make could be one those command.

> The ptrace approach would be much more reliable (though linux-specific), since you would know

> for instance:

> - that the compiler searched and did not find foo,ads in /first/dir

> - found and opened /other/dir/foo.ads

> so next time there is a build you can check first whether 'foo.ads' now exists in /first/dir. If that file now exists, you need to rebuild.

Exactly my intent.

And to build the dependency graph, I need to identify which file is an input file, and which one is an output (a target).

To do so, I can either:

1. make a complex analysis of a detailed strace log file on each file operation;
2. just ask strace the list of the involved files, and classify those file thanks to modification time : if file modification time > execution time, then it's an output.

The second option seems to be far less complex, but I need enough precision in time stamps to discriminate if a file is older than the command run time or not.

Note also that I could store a hashtag for each used file to check if the file is the same without getting in all those time tag problems (I am pretty sure most OSes propose such services).

It would certainly be useful and reliable to decide re-executing a command, but wouldn't help to classify if the used file was only read, or an output.

So, I didn't investigate in that direction.

From: Lionel Draghi <lionel...@gmail.com>
Date: Wed, 21 Nov 2018 11:13:06 -0800
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> Slightly out of topic (sorry): I found tup (http://gittup.org/tup/index.html) which appears to be doing exactly what you want to achieve. It monitors file accesses but it uses a fuse filesystem for this, rather than ptrace.

Very interesting information for me at least :-), thank you.Not sure the goal is the same.

I see on http://gittup.org/tup/ex_a_first_tupfile.html a small example of tupfile, and it give's both the input and the target with the command:

: hello.c |> gcc hello.c -o hello |> hello

This is what I try to avoid! (not to mention one more specific format)

From: Simon Wright <si...@pushface.org>
Date: Wed, 21 Nov 2018 19:48:39 +0000
Subject: Re: GNAT Modification_Time limitation
Newsgroups: comp.lang.ada

> [...]

Can't you tell from strace which files were opened for read and which for write?

I suppose there are some files that are opened read/write; either, perhaps most usually, in separate parts of the build, or by being updated in one.

I have one project (tcladashell) which runs a tcl script to generate a C source, which is compiled, built, and run to generate an Ada package spec. Which is then used in the rest of the build.

Ada and Operating Systems

PicoRV32

From: fabien....@gmail.com
Date: Tue, 11 Sep 2018 06:07:26 -0700
Subject: Ada on FPGAs with PicoRV32
Newsgroups: comp.lang.ada

New blog post on my experience with the TinyFPGA BX board and the PicoRV32 RISC-V Verilog CPU: https://blog.adacore.com/ada-on-fpgas-with-picorv32

Msys2

From: Björn Lundin <b.f.l...@gmail.com>
Date: Wed, 19 Sep 2018 10:47:30 +0200
Subject: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

Is anyone using (recently) the win-64 bit compiler found in msys2 for windows?

I found a post here some time ago with instructions, but:

pacman -S mingw-w64-x86_64-gcc-ada

runs fine - installs the compiler

pacman -S mingw-w64-x86_64-gprbuild-gpl

pacman -S mingw-w64-x86_64-aws

pacman -S mingw-w64-x86_64-asis

does not. They do not exist anymore.

I did find another gprbuild

mingw-w64-x86_64-gprbuild-bootstrap-git

which works ok to install.

Looking for xml/ada - I came up short.

so downloading from web I get 4.6.1

Compiling it goes well, but linking - not so much

```
$ mingw32-make all install
gprbuild -j0 -m –p XLIBRARY_TYPE=
static -XBUILD=Production
-XPROCESSORS=0 xmlada.gpr
gprbuild -j0 -m -p -XLIBRARY_TYPE=
relocatable -XBUILD=Production
-XPROCESSORS=0 xmlada.gpr
Build Libraries
   [gprlib] xmlada_unicode.lexch
   [link library] libxmlada_unicode.dll
d:/apps/tools/mingw2/mingw64/bin/../lib/
gcc/x86_64-w64-mingw32/8.2.0/../../../
../x86_64-w64-mingw32/bin/ld.exe:
D:\apps\tools\MinGW2\xmlada_tmp\tags\
xmlada-4.6.1\unicode\obj\relocatable\
unicode-ccs-iso_8859_1.o:unicode-ccs-
iso_8859_1.adb:(.text+0x54):
undefined reference to
`system__img_uns__set_image_unsigned'
d:/apps/tools/mingw2/mingw64/bin/../lib/
gcc/x86_64-w64-mingw32/8.2.0/../../../
../x86_64-w64-mingw32/bin/ld.exe:
D:\apps\tools\MinGW2\xmlada_tmp\tags\
xmlada-4.6.1\unicode\obj\relocatable\
unicode-ccs-iso_8859_1.o:unicode-ccs-
iso_8859_1.adb:(.text+0xd8):
undefined reference to
`__gnat_raise_exception'
ca 200 more lines of unresolved symbols
d:/apps/tools/mingw2/mingw64/bin/../lib/
gcc/x86_64-w64-mingw32/8.2.0/../../../../
x86_64-w64-mingw32/bin/ld.exe:
D:\apps\tools\MinGW2\xmlada_tmp\tags\
xmlada-4.6.1\unicode\obj\relocatable\
unicode.o:unicode.adb:(.text+0x1c5):
undefined reference to
`__gnat_rcheck_CE_Range_Check'
collect2.exe: error: ld returned 1 exit
status
gprlib: d:\apps\tools\mingw2\mingw64\
bin\gcc execution error
gprbuild: could not build library for
project xmlada_unicode
mingw32-make: *** [Makefile:55:
relocatable] Error 4
```

I even tried to add

```
  for Switches ("Ada") use ("-L" &
"mingw64/bin", "-lgnat");
```

in share.gpr - but same result - lots of unresolved symbols.

It looks like it cannot find the gnat runtime libs.

Anyone seen this before? And have an idea of how to proceed?

From: alby....@gmail.com
Date: Wed, 19 Sep 2018 02:56:53 -0700
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

The most recent version of xmlada can be found at https://github.com/AdaCore/xmlada (same for gprbuild)

I've found these to build/work on msys2, and as you have already mentioned/noticed, you need to bootstrap gprbuild first, and have the xmlada sources available.

The build instructions are self - explanatory, and from memory you need to specify the xmlada source directory when building/installing gprbuild (bootstrap)

After which you need to go back and build xmlada with gprbuild

From: "Alejandro R. Mosteo" <alej...@mosteo.com>
Date: Wed, 19 Sep 2018 12:39:44 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

I very recently (like a month ago) tried it for the first time. It compiled a hello world without issue.

I also got a bunch of unresolved symbols at first but it was my fault because I had profiling ("-p") enabled by mistake.

I didn't try anything serious though.

From: Björn Lundin <b.f.l...@gmail.com>
Date: Wed, 19 Sep 2018 15:06:52 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> The most recent version of xmlada can be found at https://github.com/AdaCore/xmlada (same for gprbuild)

Yes, I got the whole tree.

I get the same result for 4.6.1 as for 18.2 - lots of unresolved symbols.

> I've found these to build/work on msys2, and as you have already mentioned/noticed, you need to bootstrap gprbuild first, and have the xmlada sources available

Hmm, yes - do you have anything more detailed?

I don't quite follow this boostrap thing.

Does gprbuild depend on xmlada?

looking in the branches/gprbuild-1.6 directory it is still xmlada

```
$ ./configure --help
`configure' configures XML/Ada 4.2w to
adapt to many kinds of systems.
```

> The build instructions are self explanatory,

hmm, I have to disagree...

> and from memory you need to specify the xmlada source directory when building/installing gprbuild (bootstrap)

are you talking about the gprbuild in https://github.com/AdaCore/xmlada?

> After which you need to go back and build xmlada with gprbuild

It seems that I'm on the wrong rack. No matter what I do I get unresolved symbols.

The gprbuild from pacman is 18.0w.

The one from github is within the 18.2 tree but the gprbuild --version shows 18.1

Neither works now to build xmlada 18.2

```
sattmate@caleb MINGW64
/xmlada_tmp/branches/18.2
$ ./configure --prefix=/ada/xml/18.2
configure: loading site script
/mingw64/etc/config.site
checking build system type... x86_64-
w64-mingw32
checking host system type... x86_64-w64-
mingw32
checking target system type... x86_64-
w64-mingw32
checking whether gnat can build shared
libs... no
checking for a BSD-compatible install...
/usr/bin/install -c
checking whether ln -s works... no, using
cp -pR
configure: creating ./config.status
config.status: creating shared.gpr
config.status: creating Makefile
config.status: creating
tests/dom/default.gpr
```

ok good

```
sattmate@caleb MINGW64
/xmlada_tmp/branches/18.2
$ mingw32-make all install
gprbuild -j0 -m  -p -
XLIBRARY_TYPE=static -
XBUILD=Production
-XPROCESSORS=0 xmlada.gpr
No valid configuration found
Generation of configuration files failed
GNAT-TEMP-000001.TMP:1:01:
"project" expected
gprbuild: processing of configuration
project
"C:\tmp\GNAT-TEMP-000001.TMP"
failed
mingw32-make: *** [Makefile:61: static]
Error 4
sattmate@caleb MINGW64
/xmlada_tmp/branches/18.2
$ gprbuild --version
GPRBUILD Pro 18.1 (19940713)
(x86_64-w64-mingw32)
```

From: Björn Lundin <b.f.l...@gmail.com>
Date: Wed, 19 Sep 2018 15:08:14 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

Did you try compiling xmlada or AWS?

From: Simon Wright <si...@pushface.org>
Date: Wed, 19 Sep 2018 15:54:39 +0100
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

< [...]

etc - I've no idea how to go about fixing this Windows/mingw problem, other than to try to sidestep it by not trying to build the shared (relocatable) library (try configuring with --disable-shared).

You shouldn't need to rebuild gprbuild, not sure why anyone's suggesting it.

From: Jere <jhb....@gmail.com>
Date: Wed, 19 Sep 2018 19:44:12 -0700
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> You shouldn't need to rebuild gprbuild, not sure why anyone's suggesting it.

In my past experience, there were no versions of gprbuild already built for 64bit msys2. So I too tried to build it myself with no luck. I ended up installing a 32bit version which did not work at all until I called gprconfig using the --target=xxxxx option and had it create an auto.cgpr file for me. I then renamed it to default.cgpr and put it in the default share location for gprbuild (got it from the adacore manual) and then gprbuild would work.

Nowadays I suppose you could install gnat community to get the 64bit version of gprbuild, install the FSF gnat as well, then make sure both are in your path but the FSF path is earlier. This picks up your FSF gnat (since its path is before gnat community's) but also gprbuild.

From: briot.e...@gmail.com
Date: Wed, 19 Sep 2018 23:49:49 -0700
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> GNAT-TEMP-000001.TMP:1:01: "project" expected

> gprbuild: processing of configuration project

> "C:\tmp\GNAT-TEMP-000001.TMP" failed

> mingw32-make: *** [Makefile:61: static] Error 4

Does C:\tmp exist on your system? I think on Windows you need to setup some environment variable to point to a proper tmp directory.

I haven't used Windows in years so I can't be more helpful, sorry...

From: Björn Lundin <b.f.l...@gmail.com>
Date: Thu, 20 Sep 2018 09:35:13 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> Does C:\tmp exist on your system? I think on Windows you need to setup some environment variable to point to a proper tmp directory.

> I haven't used Windows in years so I can't be more helpful, sorry...

Yes it does

```
> dir c:\tmp
 Volume in drive C has no label.
 Volume Serial Number is 1602-8731
 Directory of c:\tmp
2018-09-19 15:01  <DIR>        .
2018-09-19  15:01    <DIR>        ..
```

env vars:

```
sattmate@caleb MINGW64 /xmlada_tmp/branches/18.2
$ env | grep TMP
TMP=/tmp
TMPDIR=c:/tmp
ORIGINAL_TMP=/c/Users/sattmate/AppData/Local/Temp/2
```

also /tmp exists

Hmm - I'll try Simon's suggestion now

From: Björn Lundin <b.f.l...@gmail.com>
Date: Thu, 20 Sep 2018 09:51:10 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> [...]

Ok - I got a little bit further - I disabled shared as you suggested.

However that got me the same result.

But - I did the ./configure from a bash/mingw prompt.

It failed when make was executed in bash/mingw.

But when I started an ordinary cmd.exe prompt I get d:\apps\tools\MinGW2\xmlada_tmp\branches\18.2

```
>  mingw32-make all install
gprbuild -j0 -m  -p -XLIBRARY_TYPE=static -XBUILD=Production
-XPROCESSORS=0 xmlada.gpr
gprinstall --uninstall -XBUILD=Production -XPROCESSORS=0
--prefix=/ada/xml/18.2 \
--project-subdir=lib/gnat xmlada
Uninstall project xmlada
gprinstall -f -p -XLIBRARY_TYPE=static -XBUILD=Production -XPROCESSORS=0 \
        --prefix=/ada/xml/18.2 --project-subdir=lib/gnat \
        --build-var=XMLADA_BUILD --build-name=static \
        --install-name=xmlada xmlada.gpr
Install project XmlAda_Schema - static
Install project XmlAda_Dom - static
Install project XmlAda_Sax - static
Install project XmlAda_Unicode - static
Install project XmlAda_Input - static
warning: path does not exist
'd:\apps\tools\mingw2\xmlada_tmp\branches\18.2\input_sources\../docs/_build/html/'
warning: path does not exist
'd:\apps\tools\mingw2\xmlada_tmp\branches\18.2\input_sources\../docs/_build/latex/'
d:\apps\tools\MinGW2\xmlada_tmp\branches\18.2
```

> You shouldn't need to rebuild gprbuild, not sure why anyone's suggesting it.

This is with gprbuild from pacman - the 18.0 one setting path to the 18.2 one makes it worse.

So - it looks better - but still no libraries installed.

I'll keep digging for a while.

From: Simon Wright <si...@pushface.org>
Date: Thu, 20 Sep 2018 10:17:58 +0100
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

I get that too: it "just" means that the documentation isn't installed, probably because I don't have the necessary tool chain for building html from $documentation_source_files.

You should have the static libraries and the project GPR installed properly.

From: Björn Lundin <b.f.l...@gmail.com>
Date: Thu, 20 Sep 2018 11:27:11 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

Yes, I should.

But they are to be found nowhere.

I would expect them in /ada/xml/18.2 as the prefix says - but no.

Nothing. So I created the missing _build/html and _build/latex and put XMLAda.pdf there. But no go. I got rid of the warnings - but nothing created. And I see that it compiles and creates the static libs. But it does not install them.

So - I gave up (temporarily) I took all the sources and put them into a directory and pointed _my_ .gpr file to that directory. It works - but is a crude cludge.

I am now on to install AWS too. I think that will be easier - if I can convince AWS that I do have XMLAda 'installed'...

From: Simon Wright <si...@pushface.org>
Date: Thu, 20 Sep 2018 11:06:43 +0100
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> [...]

> Yes, I should.

> But they are to be found nowhere.

I would expect the static libraries to be in

$prefix/lib/xmlada/xmlada_*.static/ and the GPRs in $prefix/lib/gnat/

(my GPRs are in $prefix/share/gpr/, don't know why the two possibilities exist or why some GPRs end up in one or the other!)

From: Björn Lundin <b.f.l...@gmail.com>
Date: Thu, 20 Sep 2018 12:51:20 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> [...]

>> Yes, I should.

>> But they are to be found nowhere.

Or I should say they get build in the source.

> I would expect the static libraries to be in $prefix/lib/xmlada/xmlada_*.static/ and the GPRs in $prefix/lib/gnat/ (my GPRs are in $prefix/share/gpr/, don't know why the two possibilities exist or why some GPRs end up in one or the other!)

Yes. Strange things are happening here.

I now found them in

```
gprinstall -f -p -XLIBRARY_TYPE=static -XBUILD=Production -XPROCESSORS=0 \
--prefix=/ada/xml/18.2 --project-subdir=lib/gnat \
--build-var=XMLADA_BUILD --build-name=static \
--install-name=xmlada xmlada.gpr
```

xmlada/18.2/static

But under a totally different root.

and it is the *.ali and lib*.a but no *.ads and no *.gpr

hmm I need to revisit this later.

From: Simon Wright <si...@pushface.org>
Date: Thu, 20 Sep 2018 17:01:20 +0100
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> and it is the *.ali and lib*.a but no *.ads and no *.gpr

I'd expect the source files to be somewhere under $prefix/include

From: Björn Lundin <b.f.l...@gmail.com>
Date: Fri, 21 Sep 2018 14:04:47 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

Yes - but there are no include directories...

This - and the other strange things - made me try it on a win10 box.

I did all the above on a win2016 box.

clean install of msys2.

grabbed xmlada (gpl-2018) and it compiled ok (with --disable-shared)

grabbed aws (gpl-2018) and it did NOT compile ok

got lots of warnings and some errors which seems bad. See below.

But I _think_ they are in the templates-parser, which we do not use.

commented out quite a bit and then

win32\build.cmd path/to/installation

worked.

copied the libs from win10 -> win2016.

Generated the system from scratch went good started went well.

Now I'm in for some heavy testing.

```
aws-os_lib-tmplt.c:323:11: warning: asm operand 1 probably doesn't match constraints
 /*NOGEN*/ CND(SIN_FAMILY_OFFSET, "sin_family offset in record");
           ^~~
aws-os_lib-tmplt.c:342:11: warning: asm operand 1 probably doesn't match constraints
 /*NOGEN*/ CND(AI_FLAGS_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:343:11: warning: asm operand 1 probably doesn't match constraints
 /*NOGEN*/ CND(AI_FAMILY_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:344:11: warning: asm operand 1 probably doesn't match constraints
 /*NOGEN*/ CND(AI_CANONNAME_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:345:11: warning: asm operand 1 probably doesn't match constraints /*NOGEN*/ CND(AI_ADDR_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:323:11: error: impossible constraint in 'asm' /*NOGEN*/ CND(SIN_FAMILY_OFFSET, "sin_family offset in record");
           ^~~
aws-os_lib-tmplt.c:342:11: error: impossible constraint in 'asm'
 /*NOGEN*/ CND(AI_FLAGS_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:343:11: error: impossible constraint in 'asm'
 /*NOGEN*/ CND(AI_FAMILY_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:344:11: error: impossible constraint in 'asm'
 /*NOGEN*/ CND(AI_CANONNAME_OFFSET, "???");
           ^~~
aws-os_lib-tmplt.c:345:11: error: impossible constraint in 'asm'
 /*NOGEN*/ CND(AI_ADDR_OFFSET, "???");
           ^~~
/usr/bin/sh: line 4: ../xoscons: No such file or directory
mingw32-make[1]: *** [Makefile:90:
../build/x86_64-w64-mingw32/release/../setup/src/aws-os_lib.ads] Error 127
mingw32-make: *** [Makefile:162: config_setup] Error 2
```

From: Maxim Reznik <rezn...@gmail.com>
Date: Mon, 24 Sep 2018 03:46:21 -0700
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

I've used msys2 to build Matreshka project on Windows 64 before GNAT Community 2018 was released. At that time msys2 GNAT was most capable compiler to build it.

The only solution I found is to move gnat*.dll, gnarl*.dll into adalib/ removing corresponding gna*.dll.a libraries.

I used commands like that:

```
 adalib=$(dirname `gcc -print-libgcc-file-name`)/adalib
 bin_dir=$(dirname `which gcc`)
 rm -f ${adalib}/libgna{t,rl}-6.dll.a
 cp ${bin_dir}/libgna{t,rl}-6.dll ${adalib}/
```

After that I was able to link shared libraries.

From: Björn Lundin <b.f.l...@gmail.com>
Date: Mon, 24 Sep 2018 12:51:16 +0200
Subject: Re: gnat via msys2 pacman + xml/ada
Newsgroups: comp.lang.ada

> […]

> The only solution I found is to move gnat*.dll, gnarl*.dll into adalib/ removing corresponding gna*.dll.a libraries.

Ok, thanks.

I got it to works with static linking, and that is all I need for the moment.

But if I want dynamic linking, I'll keep this in mind

Undefined fentry Reference

From: Petter Fryklund <petter....@atero.se>
Date: Thu, 18 Oct 2018 23:57:58 -0700
Subject: Undefined reference to __fentry__ on Windows 7.
Newsgroups: comp.lang.ada

I'm trying to make system call or spawn using GNAT.OS_lib. Both results in undefined reference to __fentry__. What am I missing?

From: Shark8 <onewing...@gmail.com>
Date: Fri, 19 Oct 2018 10:53:26 -0700
Subject: Re: Undefined reference to __fentry__ on Windows 7.
Newsgroups: comp.lang.ada

Check the linker options; it looks like the entry isn't being picked up. (I think it's something like -L[lib-name].)

From: Simon Wright <si...@pushface.org>
Date: Fri, 19 Oct 2018 19:34:53 +0100
Subject: Re: Undefined reference to __fentry__ on Windows 7.
Newsgroups: comp.lang.ada

Googling __fentry__ finds

https://unix.stackexchange.com/questions/259591/linux-kernel-missing-fentry-symbol, don't know if that's any use

From: alby....@gmail.com
Date: Sat, 20 Oct 2018 18:44:42 -0700
Subject: Re: Undefined reference to __fentry__ on Windows 7.
Newsgroups: comp.lang.ada

I am using msys2/mingw64 FSF 8.1 on windows 10 and the below sample works fine (i.e. I get the "cmd /?" output in my debug window, note I am using my Visual studio ada addin, called "Visual Ada") Note However that the Success output parameter comes back as false! Not sure why. But it does seem to work.

```
-----------------------------------------------------------
with GNAT.os_lib;
-----------------------------------------------------------
procedure ConsoleApp1 is
   Args    : GNAT.os_lib.argument_list(1..1)
:= ( others => new string(1..2) );
   Success : Boolean;
begin
   Args(1).all := "/?";
               GNAT.os_lib.spawn("cmd",
             Args, Success);
end;
```

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Sun, 21 Oct 2018 09:50:34 +0200
Subject: Re: Undefined reference to __fentry__ on Windows 7.
Newsgroups: comp.lang.ada

> I am using msys2/mingw64 FSF 8.1 on windows 10 and the below sample works fine (i.e. I get the "cmd /?" output in my debug window, note I am using my Visual studio ada addin, called "Visual Ada") Note However that the Success output parameter comes back as false! Not sure why.

Because cmd /? sets ERRORLEVEL to 1, which is then reflected in Success.

From: Petter Fryklund <petter....@atero.se>
Date: Mon, 22 Oct 2018 02:58:09 -0700
Subject: Re: Undefined reference to __fentry__ on Windows 7.
Newsgroups: comp.lang.ada

Profiling is not supported on 64 bit windows, which was the problem.

Win32 and WinRT bindings

From: alby....@gmail.com
Date: Fri, 2 Nov 2018 20:17:02 -0700
Subject: Ann: Win32 and WinRT bindings update
Newsgroups: comp.lang.ada

Dear Ada Community

The Win32 and WinRT bindings have both been updated to the latest Microsoft SDK version (10.0.17763). This version corresponds to the 1809 release of Windows 10.

Packages/Source can be found at

- https://github.com/Alex-Gamper/Ada-Win32
- https://github.com/Alex-Gamper/Ada-WinRT

Microsoft release notes:

https://docs.microsoft.com/en-au/windows/uwp/whats-new/windows-10-build-17763

From: Jere <jhb....@gmail.com>
Date: Sat, 3 Nov 2018 07:43:39 -0700
Subject: Re: Ann: Win32 and WinRT bindings update
Newsgroups: comp.lang.ada

> [...]

Thanks for this!

Question: In your readme, you specify that you need a 64bit windows build environment, but then later on say if you don't there are linux scripts in the mingw64 directory. Does this mean that you don't consider mingw64 to be a valid 64bit windows build environment? I run GCC/GNAT 8.2 on mingw64 in x86_64 and thought that that should be sufficient?

From: alby....@gmail.com
Date: Sat, 3 Nov 2018 17:22:24 -0700
Subject: Re: Ann: Win32 and WinRT bindings update
Newsgroups: comp.lang.ada

> [...]

> Question: In your readme, you specify that you need a 64bit windows build environment, but then later on say if you don't there are linux scripts in the mingw64 directory. Does this mean that you don't consider mingw64 to be a valid 64bit windows build environment? I run GCC/GNAT 8.2 on mingw64 in x86_64 and thought that that should be sufficient?

I definitely consider Mingw64 a 64bit build environment, and I have recently started using it rather than building a cross compiler on Linux (which is what the build scripts do)

So yes mingw64 is sufficient and very useable. Adacore community edition, now that its 64Bit should also work (although I must admit I have not as yet tested this.

Ada in Context

Concurrent Modification Exception in Ada's Cursors

From: rakusu...@fastmail.jp
Date: Wed, 19 Sep 2018 06:12:31 -0700
Subject: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

Why Ada's Cursors does not provide the ConcurrentModificationException, as Java Collections do, or some variant of it? Consider the following:

```
with
Ada.Containers.Indefinite_Ordered_Maps;
...
  The_Map : Map;
...
declare
  I : Cursor := First (The_Map);
  J : Cursor := First (The_Map);
begin
  -- Now Cursors are synchronized with
 -- each other and with a container.
  Delete (The_Map, I);
  -- It's O'k. But now J lost a sync and points
  -- to a dead Node.
  Next (J);
  -- What should I expected here,
  -- any well defined exception or
  -- raising a zombie?
end;
```

From: Jacob Sparre Andersen <ja...@jacob-sparre.dk>
Date: Wed, 19 Sep 2018 17:22:51 +0200
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> Why Ada's Cursors does not provide the ConcurrentModificationException, as Java Collections do,

Because that is something from Java. ;-)

> or some variant of it?

The Ada containers define the concept of tampering. I can't remember which exception you get in case you tamper with a standard container, but you can be pretty sure you will get one.

Did you try it?

Both with GCC 6.3.0 and with GNAT CE 2018 I get System.Assertions. Assert_Failure, but that is definitely not defined as a part of the tampering checks, so I suspect GNAT is wrong (but still safe) here.

I've posted an executable example here:

https://bitbucket.org/sparre/ada-2012-examples/src/default/src/container_tampering.adb

(Randy, feel free to adapt it for ACATS, if it shows something relevant)

From: Simon Wright <si...@pushface.org>
Date: Wed, 19 Sep 2018 16:53:28 +0100
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

What actually happens in this case (GNAT CE 2018) is that the program enters an endless loop looking at (what it thinks is) a node with both left and right pointers designating itself.

The ARM goes to a lot of trouble to prevent "tampering with cursors", but that's mainly to do with detecting altering the structure of a container while iterating over it, and the code you show isn't really covered by that. So I'm not sure if it isn't 'just' erroneous [1].

It would be a good thing if the error was detected. Perhaps submit a bug report to AdaCore?

[1] http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-9.html

From: Simon Wright <si...@pushface.org>
Date: Wed, 19 Sep 2018 17:05:02 +0100
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> I've posted an executable example here:

>

> https://bitbucket.org/sparre/ada-2012-examples/src/default/src/container_tampering.adb

On macOS this hangs. Also on Debian stretch. No assertion failures.

From: Jacob Sparre Andersen <ja...@jacob-sparre.dk>
Date: Wed, 19 Sep 2018 18:08:52 +0200
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> On macOS this hangs. Also on debian stretch. No assertion failures.

Even built with the project file? Strange. I'm running Debian/Stretch (9.5) here.

From: Anh Vo <anhvo...@gmail.com>
Date: Wed, 19 Sep 2018 09:31:41 -0700
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> On macOS this hangs. Also on debian stretch. No assertion failures.

It also occurred on GNAT Community 2018 running on Windows 7 and Red Hat Enterprise Linux Workstation release 7.5 (Maipo)

From: Simon Wright <si...@pushface.org>
Date: Wed, 19 Sep 2018 17:47:08 +0100
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

It hadn't even occurred to me that there might be a project file! I don't use bitbucket (in spite of having relatives working at Atlassian) and confused it with pastebin ...

OK, building with -gnata to enable assertions does indeed produce assertion failures:

```
$ ./container_tampering
ABC
SYSTEM.ASSERTIONS.ASSERT_FAILURE: Position cursor of Next is bad
SYSTEM.ASSERTIONS.ASSERT_FAILURE: Position cursor of Key is bad
```

But, if I'd been writing the Containers, this would have been a Program_Error; it's a disaster (and quite legal to reward erroneous code with PE, too).

From: Anh Vo <anhvo...@gmail.com>
Date: Wed, 19 Sep 2018 10:23:17 -0700
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

Adding -gnata switch to compilation, the SYSTEM.ASSERTIONS.ASSERT_FAILURE was raised on both Windows and Red Hat.

This is compiler dependency. Should pragma Assertion_Policy(check) be used for consistency.

From: rakusu...@fastmail.jp
Date: Wed, 19 Sep 2018 10:37:17 -0700
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

On WinXP with GNAT GPL 2017 (20170515) (i686-pc-mingw32) I get the indefinite loop without “-gnata” and with this option — the same assertions as others have, which produces by the Vet procedure, which checks if a dead Node referring to itself. I don't think that necromancy is a good idea — the memory for a dead Node is actually free and may use for another Node object, so what happens in that case?

From: Simon Wright <si...@pushface.org>
Date: Wed, 19 Sep 2018 19:05:56 +0100
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> [...]

Nothing good.

And it might be in use for something completely different, anyway!

From: rakusu...@fastmail.jp
Date: Wed, 19 Sep 2018 11:24:32 -0700
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> It would be a good thing if the error was detected. Perhaps submit a bug report to AdaCore?

I am new in Ada, so don't know what is it — a bug or a feature.

Honestly, I just try to implement a standard containers in Ada by myself for educational purposes. In process I looking in GNU Classpath sources for an advice, and notice, that they are used an int counters (an int in Java have wrap-around semantic) in both containers and iterators for preventing working with invalid iterators. They call this a “fail-fast semantic”. So I decided to implement my own containers in that way, for example:

```
private
  type State_Type is mod 2 ** Integer'Size;

  type Map is
    record
      State : State_Type  := State_Type'First;
      Size  : Natural     := Natural'First;
      Root  : Node_Access := new
         Node_Object (Variant => Empty);
    end record;

  type Map_Access is access constant
  Map;

  type Cursor is
    record
      Container : Map_Access  := null;
      State     : State_Type  :=
         State_Type'First;
      Node      : Node_Access := null;
    end record;
```

In operations with container I just compare States in Container and Cursor and throw a Concurent_Modification exception if they are not equal. If any operation deletes a Node, I just increment States in Container and Cursor, if it used, — it makes invalid any other Cursors. It may be stupid, but works well for me.

After all I looked at the Ada.Containers and notice, that its Cursors does not have any kind of modification counters. So I decided to ask here about it.

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Wed, 19 Sep 2018 22:16:15 +0200
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

>[...]

The ARM covers this case in ARM A.18.4(76-80) [I am unable to access the current ARM right now, so I'm quoting from ISO/IEC 8652:2007, which should be similar]:

"A Cursor value is invalid if ... The node it designates has been deleted from the map. The result of "=" or Has_Element is unspecified if these functions are called with an invalid cursor parameter. Execution is erroneous if any other subprogram declared in

Containers.Hashed_Maps or Containers.Ordered_Maps is called with an invalid cursor parameter.

So J is invalid and Next (J) is erroneous. ARM 1.1.5(10) defines erroneous execution: "there is no language-specified bound on the possible effect of erroneous execution; the effect is in general not predictable." In other words, this call can do anything.

From: rakusu...@fastmail.jp
Date: Wed, 19 Sep 2018 13:56:58 -0700
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> [...]

That isn't right, even if it defined in reference manual. Depending on heap usage, that https://pastebin.com/H73KZ8Ti mess will work for years and in one fine day may fail for unknown reason.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Fri, 21 Sep 2018 18:21:26 -0500
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> [...]

Of course it's right. The intent is that a cursor is directly implemented by a pair of access objects, and that is the behavior of a dangling pointer.

(That is, all real programs in Ada (and C!!) have exactly this possibility - we've lived with it for 40+ years. Doing something different would have been too radical in 1980, and it's way too late to do that now - other than optionally).

We wanted it to be possible for the Ada containers to be performance-competitive with C++ containers, and expensive cursor checks would make that impossible. In particular, the only known way to implement perfect dangling cursor detection would be to make all cursors controlled and keep links from them back to their originating container. That would be much slower (5-10 times) on every cursor operation than the current definition -- and such an implementation would have been required had we not made the operations erroneous.

(Note that the performance of the checks we did require are considered too expensive such that some of those will be eliminated from Ada 2020's definition.)

That said, another goal was to allow (but not require) dangling cursor detection -- an implementation *can* detect dangling cursors if it wants.

There are definitely schemes available that can detect 99% of such problems (noting that some corner cases can't be detected) without much extra overhead. It's also possible that an implementation could have a "checking" implementation of the containers to make such checks as well as a "fast" implementation that does not.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Fri, 21 Sep 2018 18:27:57 -0500
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

>In operations with container I just compare States in Container and Cursor and throw a Concurrent_Modification exception if they are not equal. If any operation deletes a Node, I just increment States in Container and Cursor, if it used, - it makes invalid any other Cursors. It may be stupid, but works well for me.

And it would be wrong: deleting a node from the Map only invalidates cursors that point at that node, not cursors that point at other nodes in the Map.

Those can continue to be used (for instance, if stored in another container) until their nodes or the map as a whole are deleted.

You would have to use such a counter in each *node* for this to work. An implementation on this line is the 99% percent solution that I was suggesting, but it could fail in various circumstances, most likely when a container is destroyed and a new one created in the same location (as could happen with a commonly called subprogram). It also could fail if the counter wrapped around (as it could if many nodes are created and destroyed repeatedly).

From: rakusu...@fastmail.jp
Date: Fri, 21 Sep 2018 18:09:41 -0700
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

You missed the point, perhaps because I choose obscure names for properties. Sorry.

State_Type is a modification's counter with a wrap-around semantic. It counts modifications of an internal container's structure and exists in all container's instances (call them Actual_Modifications) and in every iterator's instance (call them Known_Modifications). When an iterator is created for an existing container, its counter is initialized by value of that container, so their values are equal. When container is changed by any public method, its modification counter is incremented by one. If modification process by public method involves an iterator, the modification counter inside the iterator also incremented by one. Before any modification will doing, values of counters for container and iterator will be compared for equality, and if The_Container.Actual_Modifications /= The_Cursor.Known_Modifications then raise Concurent_Modification.

Look at http://developer.classpath.org/doc/java/util/TreeMap-source.html "knownMod" and "modCount" private ints.

> when a container is destroyed and a new one created in the same location

It will be created with zero number of modifications. If iterator will have zero number of modifications too, nothing wrong happens (just because container is empty), otherwise exception will be raised.

> It also could fail if the counter wrapped around

Why? We just need to check if the value of Known_Modifications in iterator is equal to the value of Actual_Modifications in container. It is the reason why I called them both a State — an unique number that reflects a number of container's modifications. And on a 32-bit machine we have a 4_294_967_296 modifications before a wrap.

Of course, there is a possibility of check failure exists, but it has very low probability.

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Sat, 22 Sep 2018 10:05:22 +0200
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

This schema (sequence numbers) would invalidate all cursors. Not a good idea at all, in presence of many cursors. If there is only one cursor then there is also no problem. So the case looks quite marginal to me.

If I wanted to cover it, provided I ever used cursors, I would rather have the task ID to identify the owner of the change. This would be sort of re-entrant mutex with the difference that it would raise exception instead of blocking the offender.

In short:

1. The whole idea of fine interlocking/detection of concurrent access at the granularity level of individual container elements is wrong. It will never work, IMO.
2. The idea of raising exceptions concurrently at run-time to indicate tasking design errors is even worse.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Mon, 24 Sep 2018 16:47:47 -0500
Subject: Re: Ada.Containers and concurrent modification exception.
Newsgroups: comp.lang.ada

> [...]

No, I understood the point quite well. It doesn't work. Consider the following Ada pseudo-code [...]:

```
M : Map;
R1, R2 : Cursor;
M.Insert (..., R1); -- Insert a record saving
--  the cursor. Mod counter = 1.
M.Insert (..., R2); -- Mod counter = 2.
M.Delete (R1); -- Oops, cursor is invalid,
--  container was modifed since it was
--  created but this must work.
... Element (R2) ...; -- Oops, cursor is
-- invalid, container was modified
-- since it was created, but this must work.
... Element (R1) ...; -- Cursor is invalid, OK
-- to have raise an error here.
```

The important point is that a cursor remains valid until either the container as a whole or the element it designated is deleted. Thus, if you were to use some sort of counter implementation, it has to be per-node (that is, per-element).

In addition, some container operations (especially with lists) allow moving nodes from one container to another, so the counter has to be global to all of the containers of a particular type. This brings tasking issues into it.

Unlikely Dmitry, I think this check can be done usefully, *but* it can't be done in a way which is 100% accurate. False positives (that is, errors in correct cases) cannot be tolerated, so it necessarily has to be conservative.

>> when a container is destroyed and a new one created in the same location It will be created with zero number of modifications. If iterator will have zero number of modifications too, nothing wrong happens (just because container is empty), otherwise exception will be raised.

As noted above, the counter has to be per-element and global to all containers of a given type (at least for some types of containers). So resetting for each container doesn't work.

>> It also could fail if the counter wrapped around

>Why? We just need to check if the value of Known_Modifications in iterator

>...

We're not talking about "iterators", we're talking about cursors. Iterators have the tampering check (an iterator being an active structure that iterates, a for loop being the basic example), which does indeed work like this. (And that is mandated.) Cursors are references, rather similar to access values in Ada. They live individually.

>...is equal to the value of Actual_Modifications in container. It is the reason why I called them both a State - an unique number that reflects a number of container's modifications.

>And on a 32-bit machine we have a 4_294_967_296 modifications before a wrap.

Remember, this check has to be per-element, as detailed above. Then consider a long-lived program (like a web server, which may run weeks or months) and a data structure that might be modified thousands of times per second. I agree that this is not very likely, but a check based on such a counter cannot detect all possible errors -- 99.9999% perhaps, but that is not an appropriate answer to a requirement. (And it will be much less effective if the memory is returned to the system when nodes are deleted.)

>Of course, there is a possibility of check failure exists, but it has very low probability.

Too high in my view to consider it a solution for cursor checks. Especially as they cannot be effective once the memory of the designated node is reclaimed. (For the Janus/Ada implementation, we will delay reclamation among other tricks to maximize detection, but it's far from perfect.)

Very Large Arrays

From: Shark8 <onewing...@gmail.com>
Date: Thu, 4 Oct 2018 14:38:17 -0700
Subject: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

I'm trying to implement a FITS library for work -- see https://fits.gsfc.nasa.gov/standard40/fits_standard40aa.pdf -- and have come across some rather interesting problems implementing it.

The main-problem right now is the "Primary Data Array" which can have a dimensionality in 1..999, each itself with some non-zero range. (In the files these are specified by keywords in the file like NAXIS = n, NAXIS1 = n_1, NAXIS2 = n_2, and so on until the NAXISn = n_n keyword/value pair is encountered.)

Relatively straightforward, no? Well, I'd thought I could handle everything with a dimensionality-array and generic like:

```
Type Axis_Count is range 0..999 with Size
      => 10;
Type Axis_Dimensions is Array
    (Axis_Count range <>) of Positive
    with Default_Component_Value => 1;
...

Generic
   Type Element is (<>);
   Dim : Axis_Dimensions:= (1..999 => 1);
Package FITS.Data with Pure is
   Type Data_Array is not null access
   Array(1..Dim( 1),1..Dim( 2),
      1..Dim( 3),1..Dim( 4), 1..Dim( 5),
      1..Dim( 6),1..Dim( 7),1..Dim( 8),
       --...
      1..Dim( 997),1..Dim( 998),1..Dim( 999))
   of Element
   with Convention => Fortran;
End FITS.Data;
```

But no dice.

GNAT won't even compile an array like this [999 indexes]. What's the proper way to go about doing this?

(As another interesting constraint, the file-format mandates a sort of block-structure of 2880 bytes [23040 bits], and while I don't anticipate this being an issue, something that might be relevant.)

From: Jacob Sparre Andersen <ja...@jacob-sparre.dk>
Date: Fri, 05 Oct 2018 08:17:46 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

Ouch. :-(

Something like this will work, but it doesn't look nice:

```
package Variable_Dimensionality is

  type Raw is array (Positive range <>,
              Positive range <>,
              Positive range <>) of Boolean;

  type Nice (Dim_1, Dim_2, Dim_3 :
    Positive) is
    record
      Data : Raw (1 .. Dim_1,
              1 .. Dim_2,
              1 .. Dim_3);
    end record;

end Variable_Dimensionality;
```

> (As another interesting constraint, the file-format mandates a sort of block-structure of 2880 bytes [23040 bits], and while I don't anticipate this being an issue, something that might be relevant.)

Ada.Sequential_IO and Ada.Direct_IO can both be instantiated with types of any size, so you could simply use a 2880 character String, or a 2880 element Storage_Element_Array (remember to assert that Storage_Element'Size = 8).

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 5 Oct 2018 09:20:35 +0300
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> [...]

Give it some thought. Even if each dimension would have the smallest sensible length, which is two index values, the total number of elements in that array would be 2**999, somewhat larger than the memories of current computers.

> What's the proper way to go about doing this?

If you really want to support up to 999 dimensions (though I doubt that any real FITS file will be close to that number), your program has to manage the data in blocks of some practical size.

> (As another interesting constraint, the file-format mandates a sort of block-structure of 2880 bytes [23040 bits], and while I don't anticipate this being an issue, something that might be relevant.)

Perhaps that is the solution, not a new problem.

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Fri, 5 Oct 2018 08:36:37 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> [...]

A wrong way dealing with protocols is attempting to define an Ada type having the exact representation of the data as defined by the protocol.

It is both useless and difficult to impossible, especially if bits are involved.

As a starting point consider representation clauses non-existent and simply provide operations to construct reasonably defined Ada objects from raw protocol data and conversely. Nobody would ever program anything using 999-D arrays. Nobody would ever instantiate n**1000 instances.

You could use a flat array internally and provide operations for image serialization/deserialization in whatever format, e.g. by Get_Pixel/Set_Pixel.

The hardest problem would be controlling bit representations. If they really mean that. Modern hardware usually handles octets atomically and simply does not allow accessing individual bits. There is basically no way to tell the bit order programmatically or even define "order".

From: Shark8 <onewing...@gmail.com>
Date: Fri, 5 Oct 2018 09:47:56 -0700
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> [...]

> Give it some thought. Even if each dimension would have the smallest sensible length, which is two index values, the total number of elements in that array would be 2**999, somewhat larger than the memories of current computers.

No, the smallest sensible number of indices is 1, for everything except maybe the first two or three dimensions: eg Image data from a camera, or perhaps topological data from a map (longitude, latitude, elevation).

FITS was developed for handling "image" transport by the astronomy world, back when there were 9-bit bytes and such.

From: Shark8 <onewing...@gmail.com>
Date: Fri, 5 Oct 2018 09:56:43 -0700
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> [...]

> A wrong way dealing with protocols is attempting to define an Ada type having the exact representation of the data as defined by the protocol. It is both useless and difficult to impossible, especially if bits are involved.

Protocol?

FITS is a file-format. The only reason bits are involved at all in the spec is because it was developed back when some machines had 9-bit bytes. It's all defined based on 2880 byte blocks at the very lowest level; atop that there are headers (key-value pairs) and data-arrays/-structure (indicated by data within the header).

> As a starting point consider representation clauses non-existent and simply provide operations to construct reasonably defined Ada objects from raw protocol data and conversely. Nobody would ever program anything using 999-D arrays. Nobody would ever instantiate n**1000 instances.

I still need a way to conform to the standard, which means if the standard says that it's possible to have a 999-dimension array, I need to have some way to represent this... even if it is never in actuality used.

>

> You could use a flat array internally and provide operations for image serialization/deserialization in whatever format, e.g. by Get_Pixel/Set_Pixel.

I tried this, it doesn't quite work though. (Stack overflow, oddly enough.)

```
   Function Flatten( Item : Axis_Dimensions
) return Natural is
      (case Item'Length is
               when 0 => 1,
               when 1 => Item( Item'First ),
               when 2 => Item( Item'First ) *
                         Item( Item'Last ),
               when others =>
      Flatten( Item(Item'First..Item'Last/2) )*
         Flatten( Item(Axis_Count'Succ(
               Item'Last/2)..Item'Last) ) );
```

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 5 Oct 2018 20:39:18 +0300
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> [...]

> No, the smallest sensible number of indices is 1, for everything except maybe the first two or three dimensions: e.g. Image data from a camera, or perhaps topological data from a map (longitude, latitude, elevation).

FITS images can have more dimensions than that. Further dimensions might be the frequency of the light (spectral imaging); polarisation; time when image was taken; and perhaps a couple more that don't come to mind immediately.

I understand what you tried to do, including having length-one dimensions, but I don't think that it is a sensible approach to handling up to 999 dimensions. I agree with the flattening approach that Dimitry suggested.

If your FITS files are not much larger than your RAM, the fastest approach is probably to "mmap" the file into your virtual address space and then compute the address of any given image pixel with the flattening method. If your FITS files are larger than your RAM, your program should process the file as a stream, which may or may not be practical, depending on what the program should output.

> FITS was developed for handling "image" transport by the astronomy world, back when there were 9-bit bytes and such.

I know, I used to work in astronomy. What's your point about 9-bit bytes? FITS standard version 4.0 defines "byte" as 8 bits, and allows only 8, 16, 32 and 64-bit pixels. No 9-bit pixels.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Fri, 5 Oct 2018 21:07:47 +0300
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

FITS version 4.0 defines everything with 8-bit bytes, as far as I could see with a glance at the standard. Do you need to process some older FITS files with a different byte-size?

Yes, the FITS block size (2880 octets) was chosen to be divisible by 9, and other ancient word-sizes and byte-sizes, but so what?

>> You could use a flat array internally and provide operations for image serialization/deserialization in whatever format, e.g. by Get_Pixel/Set_Pixel.

> I tried this, it doesn't quite work though. (Stack overflow, oddly enough.)

> […]

That Item'Last/2 does not seem right. If you want the middle index, it should be (Item'First + Item'Last) / 2. Perhaps this error leads to an unending recursion, explaining the stack overflow.

> Flatten(Item(Axis_Count'Succ(Item'Last/2)..Item'Last)));

But what is this function supposed to do? Is it meant to compute the length (number of elements) in the flattened array? That is just the product of the Axis_Dimension values, isn't it?

```
function Product (Item : Axis_Dimensions)
return Natural
is
  Result : Natural := 1;
begin
  for I in Item'Range loop
    Result := Result * Item(I);
  end loop;
  return Result;
end Product;
```

For computing the position (flattened index) of an element in a flattened multi-dimensional array, you need a function that takes two arguments:

- a vector giving the length of each axis
- a vector giving the index (of the element) for each axis.

Coding that function as a double recursion gives no benefit IMO. A simple loop is better, as in the function above.

Also remember that the FITS array is in Fortran order, so the index of the first axis varies most rapidly in the flattened sequence of array elements. This can be done by a "loop .. in reverse ...".

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Fri, 5 Oct 2018 21:06:14 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> I still need a way to conform to the standard, that means if the standard says that it's possible to have a 999-dimension array, I need to have some way to represent this... even if it is never in actuality used.

No. You only need to support applications reading/writing 999-D arrays in the defined format. Nothing in the standard orders any application to actually have 999-D arrays or any arrays at all.

This is why it is so important to distinguish objects and their representations as defined by the protocol from the objects and their representations in the application. The problems you face arise from an attempt to equate them.

From: Shark8 <onewing...@gmail.com>
Date: Fri, 5 Oct 2018 12:49:07 -0700
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> FITS images can have more dimensions than that. Further dimensions might be the frequency of the light (spectral imaging); polarisation; time when image was taken; and perhaps a couple more that don't come to mind immediately.

Sure; but even those are a fairly small dimensionality than what the standard allows.

> I understand what you tried to do, including having length-one dimensions, but I don't think that it is a sensible approach to handling up to 999 dimensions. I agree with the flattening approach that Dimitry suggested.

That would be rather unfortunate, to be honest. I'd much rather rely on the compiler translating the indexes than have to do so manually. I trust the compiler more than myself; plus letting it take care of keeping track of the mapping (i.e. FORTRAN convention) is nice.

My ultimate goal was to have some FITS_OBJECT type that had the appropriate data-members be able to simply "write itself to a stream" to output the proper FITS format file.

>

> If your FITS files are not much larger than your RAM, the fastest approach is probably to "mmap" the file into your virtual address space and then compute the address of any given image pixel with the flattening method. If your FITS files are larger than your RAM, your program should process the file as a stream, which may or may not be practical, depending on what the program should output.

Most of the anticipated usage for where I am right now would be producing FITS files, likely in something that would boil down to a coupling like this:

```
Count : Positive := 1;
Today : Ada.Calendar.Time renames
    Ada.Calendar.Clock;
New_Image: Camera_Image renames
    Normalize( Get_Camera_Image );
New_Object : FITS.Object :=
  FITS.Create_w_Defaults(
    New_Image );
--..
-- Writes data out to "Observation
-- (YYYY-MM-DD)_00X.FITS".
New_Object.Write( Base => "Observation",
   Date => Today, Count => X );
```

I'd rather not tie things to a memory - mapped file at a high level, but it may be that my ideal abstraction is non-tenable.

> > FITS was developed for handling "image" transport by the astronomy world, back when there were 9-bit bytes and such.

>

> I know, I used to work in astronomy. What's your point about 9-bit bytes? FITS standard version 4.0 defines "byte" as 8 bits, and allows only 8, 16, 32 and 64-bit pixels. No 9-bit pixels.

Sorry, that was more about Dmitry's suggestion to pretend representation-clauses don't exist; I haven't done anything at a bit-level at all. (And I don't think I need to, except perhaps to mark the Primary-Data array elements as Big-endian [IIRC].)

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Fri, 5 Oct 2018 22:31:25 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> [...]

> I'd rather not tie things to a memory-mapped file at a high level, but it may be that my ideal abstraction is non-tenable.

You still can do this. The object can have any representation, the stream attribute will encode/decode it as required by FITS:

```
  Object : FITS.Image :=
    Create
    ( Base  => "Observation",
      Date  => Clock,
      Image => Get_Camera_Image
    );
begin
  FITS.Image'Write (Stream, Object);
```

Or without any intermediate objects:

```
FITS.Store
(  File  => Stream,
   Base  => "Observation",
   Date  => Clock,
   Image => Get_Camera_Image
);
```

The problem with intermediate objects is copying bulky data like images unless you deploy some complex reference-counting schema. Good bindings support provide in-place I/O operations.

From: Jacob Sparre Andersen <ja...@jacob-sparre.dk>
Date: Sat, 06 Oct 2018 08:40:22 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> […]

Why not leave the transport between disk and RAM to the operating system, and use memory mapping even if the file is larger than the RAM of the system?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Sat, 6 Oct 2018 12:35:21 +0300
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> Why not leave the transport between disk and RAM to the operating system, and use memory mapping even if the file is larger than the RAM of the system?

One could use mmap even for very large files, I guess, but on a 32-bit system the virtual address space could run out. On a 64-bit system, probably not.

This was advice based on my feeling of what would work best. If the file is processed as a stream, the OS is likely to use read-ahead to speed things up. If the file is mmap'ed to use the virtual-memory paging system, I'm not sure if the OS will do read-ahead, but perhaps modern OS's have some such adaptive optimisations even for mmap'ed files.

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Sat, 6 Oct 2018 18:04:55 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

>

> Most of the anticipated usage for where I am right now would be producing FITS files, likely in something that would boil down to a coupling like this:

For that you can probably get by with something that translates your image into a sequence of FITS "blocks" and writes them to a file:

```
FITS.Write (Image => Image, File_Name =>
"George");
```

There doesn't seem to be any reason to store a FITS object.

From: Shark8 <onewing...@gmail.com>
Date: Sat, 6 Oct 2018 11:49:59 -0700
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> […]

> There doesn't seem to be any reason to store a FITS object.

For our specific usage *RIGHT NOW*, sure.

All that's *REALLY* required, for the Telescope's production-side is writing out those blocks, this is true... but doing it this way would be kneecapping myself in the sense of maintenance & usability. (Like global-variables/states.)

[WRT software:] The Astronomy field is pretty fragmented and ripe for solid, reliable, libraries. Getting a good FITS library is only one of several things I'd like to produce:

(1) A good ISO 8601 library, to include periods and intervals;

--(a) This would include a secondary scheduling library.

(2) A stellar-coordinate library;

(3) A good abstraction for telescope-control.

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Sat, 6 Oct 2018 23:40:52 +0200
Subject: Re: A little trouble with very large arrays.
Newsgroups: comp.lang.ada

> All that's *REALLY* required, for the Telescope's production-side is writing out those blocks, this is true... but doing it this way would be kneecapping myself in the sense of maintenance & usability. (Like global-variables/states.)

Of course you'd also implement

```
Image : FITS.Image_Handle := FITS.Read
("George");
```

But there still doesn't seem to be a reason to store a FITS object.

Documentation usefulness

From: pat...@spellingbeewinnars.org
Date: Sun, 14 Oct 2018 17:42:03 -0700
Subject: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

I was just reading through the list of libraries included in C++ boost. We don't have matches for all of this but the libraries that ship with Ada have quite a bit of coverage too and I was surprised that we more or less match up.

The thing is, that the Ada libraries come with almost no documentation/example code at all.

Do you find that just reading through the spec files is enough for you to understand how to use the library in most cases? I was thinking I would try more of them out but I also wonder if I am about to go off on a suicide mission.

From: Henrik Härkönen <heha...@gmail.com>
Date: Sun, 14 Oct 2018 22:44:56 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> [...]

My experience of Ada is still very limited, but so far I'd like to say that I like the separate spec files, and for small "interfaces" the might be enough. But I'm also a learn-by-example kind of person, so an example always gives me much more confidence when starting to use some new library. I don't think that seeing just the types and function signatures is always enough to convey the "intent" of that particular interface.

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Mon, 15 Oct 2018 09:22:13 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> The thing is, that the Ada libraries come with almost no documentation/example code at all.

Some specific libraries in mind?

> Do you find that just reading through the spec files is enough for you to understand how to use the library in most cases? I was thinking I would try more of them out but I also wonder if I am about to go off on a suicide mission.

Usually specifications is all you need.

Tricky stuff must be explained of course, especially things which specifications do not cover: exception contracts, behavior under tasking, numeric complexity etc.

Frameworks is a different kind of thing, they always require getting started, examples etc., regardless the language.

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Mon, 15 Oct 2018 10:42:19 +0300
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> […]

If you mean the language-defined libraries, such as the standard container packages, those are of course documented in the Ada Reference Manual, which can be found at http://www.ada-auth.org/arm.html.

The RM description is rather condensed and the number of examples is small, but I have found them sufficient.

For more tutorial documentation and more examples one must turn to books or other learning materials (http://www.adaic.org/learn/materials/).

From: AdaMagica <christ-u...@t-online.de>
Date: Mon, 15 Oct 2018 03:05:51 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

If you think of nonlanguage defined libraries, I must say that what I've seen so far is mostly very poorly documented.

It's rarely the case that the pure Ada text in the package specs is enough to understand how a library works and how it is supposed to be used. At least a lot of comments should be included.

In an ideal world, any libraries should be documented in the same detail as all language supplied ones in the RM.

This is almost never the case.

The biggest sin: If you want to use some library and have to look into the bodies to find out how it has to be used and what it does.

From: pat...@spellingbeewinnars.org
Date: Mon, 15 Oct 2018 04:25:26 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

Thanks Guys

So I just printed all the headers for the Ada.XXX packages and picked one at random. Yes the reference manual covered it that helps a lot.

I also printed all the Gnat.XXX headers and here are a few just randomly picked:

Gnat.Heap.Sort_G

Gnat.Memory_dump

Gnat.Byte_Swapping

I just picked Gnat.Byte_Swapping and I tried to look for an example. I found the GNAT reference manual that gives a short description but I did not find an example of it in use.

Can I assume that if Adacore included this, it's good software? but can I also assume that I will need to post to this list and/or read through the spec and body to understand how it works?

From: Markus Schöpflin <no.spam@spam.spam>
Date: Mon, 15 Oct 2018 13:57:42 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

[...]

> Can I assume that if Adacore included this, it's good software? but can I also assume that I will need to post to this list and/or read through the spec and body to understand how it works?

AdaCore includes extensive documentation in the spec files. So you just need to look at the spec files, e.g. for Gnat.Byte_Swapping have a look at g-bytswa.ads.

From: pat...@spellingbeewinnars.org
Date: Mon, 15 Oct 2018 06:02:27 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

Okay thanks :)

As long as I am SUPPOSED to know how to use the library by using only the spec file, I will give it a try.

Thanks to everyone again

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Mon, 15 Oct 2018 18:50:19 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> The thing is, that the Ada libraries come with almost no documentation/example code at all.

"The Ada libraries" covers a lot of ground. If you're referring to the standard library, as others have pointed out, it's well documented in the ARM.

For other libraries, if they can't be used by only reading the specs or by documentation similar to that in Annex A, then I question the competence of the developer and the correctness of the implementation.

(Now you can look at the pkgs at github.com/jrcarter and question my competence. If they can be clearer I'd like to improve them.)

From: AdaMagica <christ-u...@t-online.de>
Date: Tue, 16 Oct 2018 02:57:37 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> For other libraries, if they can't be used by only reading the specs or by documentation similar to that in Annex A, then I question the competence of the developer and the correctness of the implementation.

How true!

> (Now you can look at the pkgs at github.com/jrcarter and question my competence.

> If they can be clearer I'd like to improve them.)

OK. In function Password_Generation.Generate, what are Domain and Master for?

What does Hash_Symbol do?

What does "correctness of the implementation" mean for this package?

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Tue, 16 Oct 2018 18:57:46 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> OK. In function Password_Generation.Generate, what are Domain and Master for?

> What does Hash_Symbol do?

Good points. This was extracted from a more monolithic, earlier version of Password_Gen, where perhaps these concepts were better explained, and I didn't think to clarify them when I pulled them out. I'll try to explain these better.

> What does "correctness of the implementation" mean for this package?

That the function returns the same password for the same inputs, and the passwords have all the desirable features for generated passwords: they appear random, contain characters from all the major food groups, and give away nothing about the master password.

From: AdaMagica <christ-u...@t-online.de>
Date: Thu, 18 Oct 2018 02:06:32 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> […]

> Good points. This was extracted from a more monolithic, earlier version of Password_Gen, where perhaps these concepts were better explained, and I didn't think to clarify them when I pulled them out. I'll try to explain these better.

Yes. Even inner specs that are not visible for the user must follow these rules that everything visible in the spec is exactly described. Then nothing bad happens when an internal package is extracted.

> > What does "correctness of the implementation" mean for this package?

> That the function returns the same password for the same inputs, and the passwords have all the desirable features for generated passwords: they appear random, contain characters from all the major food groups, and give away nothing about the master password.

This is what I mean. There is no requirement defined for this operation. So how can I as a user know what I get? So you have to put this in the spec as a description. Then a user can make test to check whether the claims are true.

From: Brad Moore <bmoor...@gmail.com>
Date: Thu, 18 Oct 2018 08:24:37 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

It should also be mentioned that with Ada 2012, the addition of contracts helps also aid to reader of a spec understand what a subprogram does, and how it was intended to be used by the author.

A problem with documentation is that it can become stale if not maintained, whereas the contracts are assertions in the code and thus tend to be more accurate.

A designer of a package should consider, for example, what pre and post conditions should be applied to the subprograms of that package. The addition of contracts tends to simplify the documentation that is needed.

As an example, in the standard package Ada.Locales, there is;

```
type Country_Code is new String (1 .. 2)
  with Dynamic_Predicate =>
    (for all E of Country_Code =>
        E in 'A' .. 'Z');

function Country return Country_Code;
```

If we didn't have the contract for Dynamic_Predicate on the Country_Code type, we would need to document that the function Country returns a 2 character string where all the characters of the string consist of capital letters from A to Z inclusive.

With the contract, this doesn't need to be documented, and the contract is more concise for the reader than having to read a full paragraph of text. Further, anywhere the Country_Code result is used in the user's program, it is clear that the contract holds, since it is a property of the type.

If the implementation changes in a way that breaks the contracts, then this tends to get caught, and either the implementation is adjusted to meet the contracts, or the contracts are adjusted to meet the implementation. Generally one tries to avoid making changes to contracts, particularly if there are existing users of those contracts.

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Thu, 18 Oct 2018 19:29:14 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> […]

I'm not sure I agree. This is package Password_Generation, function Generate, a service to generate passwords. I think that is clear from the code and needs no further explanation. The description above is simply the definition of a good generated password.

I don't think this spec should be a tutorial on password generation. Someone looking at it wants to generate passwords, and should know why one generates passwords and what the qualities of a good generated password are.

From: AdaMagica <christ-u...@t-online.de>
Date: Thu, 18 Oct 2018 10:54:23 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

So just say WHAT it does, not HOW it does.

If you do not say that your code produces a good password, how can the user know he will get a good password? He must trust JC because he possibly knows him personally or because he uses other well defined and well written sw from him.

But I claim: In SW, there must be no such trust. JC may just have been being lazy in this case.

And I further claim there are tons of bad SW around. Just because a unit's name says XXX, there is no guarantee that it indeed does XXX.

If I were looking for password generators, I would not waste my time in trying some that don't claim to produce good ones; instead I'd pick one with such a claim and test it thoroughly.

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Thu, 18 Oct 2018 22:07:16 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> […]

> But I claim: In SW, there must be no such trust. JC may just have been being lazy in this case.

Right. So if I claim the function returns a good password, you won't trust me and won't accept my claim until you've tested it. So there's no point in my making such a claim.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Thu, 18 Oct 2018 16:24:11 -0500
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

I'd also want some documentation as to what the author considers a "good" password, as the advice for that seems to change every few years. I'd guess that this package was based on some advice from a few years ago, so it might not even be "good" anymore. (That's certainly the case with a lot of Ada libraries, which haven't been modified for a long time as no one has seen a need to do so.)

After all, "good" is not a technical term, in any field that I'm aware of.

Certainly, you don't need to put this sort of documentation on individual subprograms; it belongs to the library as a whole. But without it, you really can't judge fitness.

From: "J-P. Rosen" <ro...@adalog.fr>
Date: Fri, 19 Oct 2018 09:39:31 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> After all, "good" is not a technical term, in any field that I'm aware of.

However, quoting A.4.9(12/2):

"The Hash functions should be good hash functions..."

Admittedly, it's only implementation advice ;-)

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Fri, 19 Oct 2018 20:27:14 -0500
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

>> After all, "good" is not a technical term, in any field that I'm aware of.

> However, quoting A.4.9(12/2):

> "The Hash functions should be good hash functions..."

>

> Admittedly, it's only implementation advice ;-)

Exactly. One of the most meaningless statements in the RM. Note that this statement goes on to give a description of what it means: "...returning a wide spread of values for different string values. It should be unlikely for similar strings to return the same value." That's the important part; "good" really doesn't add anything here.

From: AdaMagica <christ-u...@t-online.de>
Date: Sun, 21 Oct 2018 08:20:07 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

> [...]

> Right. So if I claim the function returns a good password, you won't trust me and won't accept my claim until you've tested it. So there's no point in my making such a claim.

So if you used a library written by me, would you take it as is without ever caring whether it does what I claimed that it does? How imprudent!

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Sun, 21 Oct 2018 20:56:39 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

It appears that you have significantly misunderstood what I wrote.

From: AdaMagica <christ-u...@t-online.de>
Date: Mon, 22 Oct 2018 09:04:51 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

OK, let's put it in a different way.

There is a library written by me called XYZ, but I do not claim anything about reliability etc. And there are many others out there also doing XYZ. Which one would you chose? I guess the one with the optimal documentation, but definitely not mine; and you will test it, wouldn't you?

From: "Jeffrey R. Carter" <spam.jrc...@spam.not.acm.org>
Date: Mon, 22 Oct 2018 21:13:31 +0200
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

I'm not talking about me. I'm talking about your statement that you don't trust any S/W, and the logical consequences of that towards claims made by authors about their S/W. If an author makes a claim, you won't trust it until you've tested it, because you don't trust S/W. If an author makes no claim, that shouldn't change anything, because you have the same trust in that S/W that you have in S/W with a claim: none until you've tested it. Yet you seem to say that between a library that makes a claim of goodness and another with no claim, you'd choose the one with the claim, despite having equal lack of trust in both.

From: AdaMagica <christ-u...@t-online.de>
Date: Tue, 23 Oct 2018 03:00:57 -0700
Subject: Re: Is the Documentation In a spec File Usually Enough For You?
Newsgroups: comp.lang.ada

We all are fond of Ada and trust the compilers and like the portability of Ada's code. Why? Because there is the ACATS (the former ACVC) test suite. And according to Randy, many compilers have great difficulties to fulfil the last x% of the ACATS.

So tests of SW are a kind of sine qua non.

Interfacing to C and COBOL

From: pat...@spellingbeewinnars.org
Date: Mon, 15 Oct 2018 06:42:37 -0700
Subject: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

Okay, don't laugh(or laugh too hard)

I need to interface with automatically generated C. function pointers are represented as a pointer to a pointer to a char. This is weird but done for pointer arithmetic purposes.

I have this:

```
type pointer1 is access character ;
type pointer2 is access pointer1 ;
foo : pointer2 ;
```

What do you think is the best way to assign an access to a subprogram to foo?

I realize this is really-really weird and dangerous which is why I am asking for help.

From: pat...@spellingbeewinnars.org
Date: Mon, 15 Oct 2018 07:20:07 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

sorry to reply to my own post.

Here is some C I am trying to redo in Ada:

```
int init_obj(char **test_ptr){
  (*test_ptr) = (char *)&foo ;
}
or in a struct/record, this is obviously just
hacky code to try something out
typedef struct _obj
{
int intty ;
char **procedure_pointer ;
} obj ;
int foo(int nothing){
 printf("this is foo in C \n")  ;
}
int init_obj(obj *obj_test){
  obj_test->intty            = 4 ;
  obj_test->procedure_pointer = (char **)&foo
;
}
```

From: Jacob Sparre Andersen <ja...@jacob-sparre.dk>
Date: Mon, 15 Oct 2018 16:51:28 +0200
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> I need to interface with automatically generated C.

Have you tried to see how a binding generated by `gcc -fdump-ada-spec` (or `g++ -fdump-ada-spec`) looks?

From: Simon Wright <si...@pushface.org>
Date: Mon, 15 Oct 2018 17:05:33 +0100
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> [...]

But it's not really a char**, is it!

```
procedure Pointers is
   type Procedure_P is access procedure
with Convention => C;
   type Procedure_P_P is access
   Procedure_P;
   P : Procedure_P_P;
   procedure Proc is null with Convention
   => C;
begin
   P.all := Proc'Access;
end Pointers;
```

You could try something involving Proc'Address, though it's not obvious that that's mandated to be the address of something that you could call. Maybe convention C does that.

From: pat...@spellingbeewinnars.org
Date: Mon, 15 Oct 2018 12:28:59 -0700
Subject: Re: Least Dangerous Way to Do This ?
Newsgroups: comp.lang.ada

Thanks Jacob and Simon!

This is proving to be quite tricky. I have the pointer stuff worked out but I have memory access errors. I will figure this out. If this works out, I will post to the mailing list, I think you will find it interesting or laughable :)

From: pat...@spellingbeewinnars.org
Date: Mon, 15 Oct 2018 12:36:08 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

Okay, I worked through the last piece. You know what I am doing.....

I am thinking about using the Ada.XXX and Gnat.XXX libraries from GnuCOBOL!

Hee-hee, I know COBOL isn't going to have a lot of fans here but it is actually a lot of fun.

I will post back with some examples just in case someone cares :)

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Mon, 15 Oct 2018 15:36:25 -0500
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> [...]

It's not certain (in the real world, at least) if a function pointer and an object pointer have the same representation. We treat them as completely different things in our code generator, because some weird machines (in particular the U2200) have very large representations for function pointers (one version was 8 36-bit words!). Similarly, on the 16-bit 8086 compilers, function pointers carried a segment (thus a 32-bit address), and object pointers usually didn't (thus a 16-bit address).

So if you want this code to work in different environments, I'd try to avoid mixing the two (as Simon suggested).

From: Shark8 <onewing...@gmail.com>
Date: Tue, 16 Oct 2018 09:07:56 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> [...]

> I am thinking about using the Ada.XXX and Gnat.XXX libraries from GnuCOBOL!

Ada has a whole section in Annex B for interfacing to COBOL -- "B.4 Interfacing with COBOL" -- online here: http://www.ada-auth.org/standards/2xrm/html/RM-B-4.html

Exporting to COBOL will be as easy as saying:

```
Function J return Integer
  with Export, Convention => COBOL;
```

And Importing is similar:

```
Function K return
          Interfaces.COBOL.Floating

  with Import, Convention => COBOL;
```

Actually given Ada's ease with interfacing like this, and the SPARK provers, I'm surprised that the banking industry hasn't leveraged Ada into its infrastructure, using Ada to prove the correctness of the system and COBOL for the fast reporting/record-processing as-needed.

>

> I will post back with some examples just in case someone cares :)

I think that would be pretty cool.

From: pat...@spellingbeewinnars.org
Date: Tue, 16 Oct 2018 15:32:26 -0700
Subject: Re: Least Dangerous Way to Do This ?
Newsgroups: comp.lang.ada

Thanks Randy

GnuCOBOL has 2 types, procedure-pointer and pointer.

I would have to dig into the generated C to tell you for sure how they differ but I think I should be safe.

GnuCOBOL runs on pretty much anything that has an OS from Raspberry PIs to Z Series mainframes so I think that this is a concern that has been addressed.

From: pat...@spellingbeewinnars.org
Date: Tue, 16 Oct 2018 17:09:47 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

I am actually pretty familiar with the interface to COBOL package. I actually don't trust it. I am thinking about trying to improve it but in its current condition, it could be a liability.

So Ada83 was pretty cool already but if we had Ada68 or Ada78 they would have sucked large. COBOL68 and COBOL73 suck and the language is still haunted by criticism from this time.

When I first started with GnuCOBOL(called Open-Cobol at the time) I thought it was awesome and that the critics must all be insane and then I saw code from 68 and 73 revisions. One program (like a procedure) was almost 10000 lines long. All identifiers were limited to 8 characters. There was not a single comment in it. So yes old COBOL is terrible. The thing is that the interfaces cobol package appears to be for this old code and not the current stuff.

In COBOL we have pointers, procedures (called programs) and functions. The spec file in the interfaces cobol package makes all kinds of false statements.

GnuCOBOL generates intermediate C, so I can inspect it and see what is happening. At the moment I am better off with the interfaces C package.

I have wondered the same thing. People can write very reliable COBOL code but there are some topics related to calling other procedures (programs) that could be a concern for the most critical portions, this is where Ada would really shine in banking, yet it doesn't.

Ada has much better facilities for organizing huge amounts of code yet COBOL likely has far larger code bases. I think one bank has something like 130 million lines of COBOL, I could borrow the library book I read this in, to get the exact amount if you are interested but the point is that organizing this much code in COBOL is very hard.

I hope you don't mind a bit of a rant but I think Ada has serious-serious advocacy issues. Adacore has done a horrible job and we could improve more as a community too. Catering to existing Adacore customers does not move the language forward. Adacore did not move to microcontrollers for the longest time and it has staff that go around telling people not to use Ada for webservers or telephony systems when this would actually be good things to do and so on.

People using Ada for fun like on AVRs and on the Raspberry Pi are doing good work. We really need to show people what can be done with it and done with it for fun.

Books are terrible. You can read hundreds and hundreds of pages were all that is being demonstrated is some language feature you are never going to use and text_io displaying something. I have 53lbs of Ada books (and counting) and in all this I think I have 2 or 3 pages that deal with interfacing with real hardware and there is next to no information about interfacing with other languages. How many new application these days are going to be in pure Ada, it's crazy to cover features from Ada2005 and Ada2012 and not providing any examples of interfacing with other languages. What are you actually going to do with interfaces in a new code base that has no contact with other languages? Isn't it more important to understand procedures, function and packages and how to use Ada with other languages?

Today is a special day for me. I realized something super-awesome about Ada that no one told me about and that I did not read about and this really should not have happened. It's so simple in hindsight but I didn't realize that you could use Ada packages with other languages without "with"ing them into an Ada program. I patched byteswap with pragma export and called it from COBOL. I can now use an Ada library directly from COBOL without writing any Ada code. Fortran and C people could do the same, with some simple patching.

So I can't verify that any of this is true but according to MicroFocus, a company with a vested interest in COBOL (and that sued a city here in Canada for having too many copies of their software) the cost to replace COBOL in the USA would be over 1 trillion. They say that there is hundreds of billions of lines of code in use. Others have said that there is more COBOL code than all other languages combined. Again, I am not saying that all of this is true, it could be fanboy propaganda but certainly there is mountains of it.

Please see this site (it is a lot of material on one page and can be slow to load):

https://open-cobol.sourceforge.io/faq/index.html

This will give a good overview of the language. There is a small part that touches on Ada. However look at this code, it's almost all using C libraries. GnuCOBOL has lots of facilities for interfacing with C, especially matching types but we still have to jump though some hoops such as appending null bytes and so on.

Ada and COBOL are much closer then COBOL and C and Ada has tons of facilities for working with other languages, such as its type system and strings.fixed to match COBOL's and so on, they are just poorly documented.

I am planning on patching as many Ada libraries as I can so that they can be used from GnuCOBOL. There is so much functionality that we don't have in our community. Imagine if this ends up being used by existing COBOL code bases... Maybe even banks will see the value if I can do this, I write up lots of documentation and I can actually try to promote these concepts.

If one company is entrusted to promote Ada and all they want to do is service existing customers, the language will die out when there old customers finally switch to C++/Java etc.

There is so much to promote and so much value to be had but Ada instruction books and Adacore are not going to get the word out, we need to!

From: Petter Fryklund <petter....@atero.se>
Date: Tue, 16 Oct 2018 22:16:17 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> [...]

I don't think the U2200 is weird, it is or was actually a very nice machine. Unfortunately most of the Unisys Linköping people where kicked out before we could try Ada, but we were very interested in it then. I for a while maintained the local releases of COBOL before going in to performance analysis and cache-disk simulations.

Is there any U2200 out there? comp.sys.unisys talks a lot about virtual ones.

From: Shark8 <onewing...@gmail.com>
Date: Wed, 17 Oct 2018 13:49:35 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> [...]

I'm rather looking forward to seeing your improved COBOL interface.

(We might even be able to replace Interfaces.COBOL, or perhaps add a new child-package like Interfaces.COBOL.ISO_1989_2014.)

>

> GnuCOBOL generates intermediate C, so I can inspect it and see what is happening. At the moment I am better off with the interfaces C package.

Hm, are you going to use that as a bit of a feedback-loop in your design of a [new] COBOL interfacing package?

> [...]

> Ada has much better facilities for organizing huge amounts of code yet COBOL likely has far larger code bases. I think one bank has something like 130 million lines of COBOL, I could borrow the library book I read this in, to get the exact amount if you are interested but the point is that organizing this much code in COBOL is very hard.

Indeed, and code-organization is one area that Ada really shines [IMO].

> I hope you don't mind a bit of a rant but I think Ada has serious-serious advocacy issues. Adacore has done a horrible job and we could improve more as a community too. Catering to existing Adacore customers does not

move the language forward. Adacore did not move to microcontrollers for the longest time and it has staff that go around telling people not to use Ada for webservers or telephony systems when this would actually be good things to do and so on.

I'm not bothered by your rant -- it actually mirrors a lot of the criticisms the Ada-community as-a-whole has. While I'm glad that AdaCore has interests in Ada, and does some good work, there are some big problems that "only one [viable, opensource] implementation" entails which isn't good for the language as-a-whole.

[...]

> Please see this site(it is a lot of material on one page and can be slow to load):

> https://open-cobol.sourceforge.io/faq/index.html

Interesting, I'll take a look.

> I am planning on patching as many Ada libraries as I can so that they can be used from GnuCOBOL. There is so much functionality that we don't have in our community. Imagine if this ends up being used by existing COBOL code bases... Maybe even banks will see the value if I can do this, I write up lots of documentation and I can actually try to promote these concepts.

>

I, for one, would love to see this.

There's a lot of value to be had here: keeping your main-system organized, proven (where possible), and ensuring that your main reporting/processing isn't being handed garbage is a big win.

> If one company is entrusted to promote Ada and all they want to do is service existing customers, the language will die out when there old customers finally switch to C++/Java etc.

Yep.

This is a big problem, in part because existing customers develop workarounds for flaws and issues that they never voice (that's just how things are), whereas new customers will have different needs and ideas [and expectations] which often at least shed some light on issues.

>

> There is so much to promote and so much value to be had but Ada instruction books and Adacore are not going to get the word out, we need to!

I try.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Wed, 17 Oct 2018 16:50:50 -0500
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

>I don't think the U2200 is weird, it is or was actually a very nice machine.

Well, the actual machine code is very weird; it was weird enough that they provided our project with a guy whose main job was to figure out what was happening at the lowest level and feed it back to the rest of us working in the "normal" world.

>Unfortunately most of the Unisys Linköping people where kicked out before we could try Ada, but we were very interested in it then. I for a while maintained the local releases of COBOL before going in to performance analysis and cache-disk simulations.

So far as I know, no one ever used the Ada 95 compiler we built for Unisys.

(I hope not, 'cause we never got paid for one. :-) It was a lot harder project than we predicted and it didn't get done fast enough for the prospective customers. A very interesting project, though.

>Is there any U2200 out there? comp.sys.unisys talks a lot about virtual ones.

Dunno. I haven't heard from any of that group in many years.

From: Petter Fryklund <petter....@atero.se>
Date: Wed, 17 Oct 2018 22:24:11 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

> Well, the actual machine code is very weird; it was weird enough that they provided our project with a guy whose main job was to figure out what was happening at the lowest level and feed it back to the rest of us working in the "normal" world.

I regard(ed) 1100/2200 as the normal world ;-), it is (was) everything else that is weird. For instance, until much later I didn't know about little-endian, still weird to me. I was almost brought up with 1100, since I got head-hunted together with 27 other people directly from University.

Traditional machine-code like early 1100 is very nice. I love traditional LMJ, Load Modifier and Jump, which loaded the return address in the modifier part of a X (index) register and jumped to the specified location. The called routine could then return to an offset of the X register where 0 was usually the normal return and others different error returns. The X register could also do automatic increments and decrements when used. The Modifier part was added to the other part if a bit in the instruction was set. We needed all 36 bits of the instructions for different purposes including part-word operations. The ability to have for different banks directly visible (maybe later there were more?) and possibility to change any of them with just one or two instructions was very neat. Those were the days.

I agree that the new what-ever-it-was-called, I bet new, was a bit ..., but not weird ;-)

But 1991 I "had" to learn Ada, which I have used most of the time since.

From: pat...@spellingbeewinnars.org
Date: Thu, 18 Oct 2018 06:52:24 -0700
Subject: Re: Least Dangerous Way to Do This?
Newsgroups: comp.lang.ada

Thanks very much. I am going to start another thread tonight about GnuCOBOL and Ada. This thread has drifted quite a bit.

Equality Operator

From: JLotty <jlotsp...@gmail.com>
Date: Sat, 10 Nov 2018 02:36:28 -0800
Subject: "Equality operator appears too late"
Newsgroups: comp.lang.ada

```
with
Ada.Containers.Synchronized_Queue_Interfa
ces;
with
Ada.Containers.Unbounded_Priority_Queues
;
procedure Min_Working_Example is

  generic
    type Data_Type is private;
    type Weight_Type is (<>);
    with function "<" (Left, Right :
    Weight_Type) return Boolean is <>;
  package Min_Data_Structure is
  private
    type Data_Rec is record
      Data   : Data_Type;
      Weight : Weight_Type;
    end record;

    function Get_Priority
      (Element : Data_Rec)
    return Weight_Type;

    function Before
      (Left, Right : Weight_Type)
    return Boolean;
    package Queue_Interface is new
Ada.Containers.Synchronized_Queue_Interfa
ces
      (Data_Rec);

    package Edge_Queue is new
Ada.Containers.Unbounded_Priority_Queues
      (Queue_Interfaces => Queue_Interface,
       Queue_Priority   => Weight_Type,
       Get_Priority     => Get_Priority,
       Before           => Before);
  end Min_Data_Structure;

  package body Min_Data_Structure is
    function Get_Priority
      (Element : Data_Rec)
       return Weight_Type is
      (Element.Weight);
      function Before
        (Left, Right : Weight_Type)
         return Boolean is
        (Left < Right);
   end Min_Data_Structure;
begin
   null;
end Min_Working_Example;
```

When compiling the above, I get the following error:

Builder results

min_working_example.adb

27:7 equality operator appears too late

27:7 instantiation error at a-cuprqu.ads:76

The error is occurring when the builder tries to elaborate the Ada.Containers. Unbounded_Priority_Queues package, where and "=" operator is defined on line 76.

I'm running the following command for build:

gprbuild -q -c -f -gnatc -u -Ptest.gpr min_working_example.adb

using version:

GPRBUILD GPL 2017 (20170515) (x86_64-pc-linux-gnu)

Copyright (C) 2004-2017, AdaCore

I don't know what to do from here. Any help you can offer would be appreciated.

From: joak...@kth.se
Date: Sat, 10 Nov 2018 05:34:49 -0800
Subject: Re: "Equality operator appears too late"
Newsgroups: comp.lang.ada

I've tried the code with GNAT Community Edition 2018 and got the same error message. My spontaneous guess is that this is a compiler bug. The work around would be to rewrite the code not to instantiate the generic package Ada.Containers.Unbounded_Priority_Queues inside a generic compilation unit. I hope somebody else in this forum has a better idea on what to do.

From: JLotty <jlotsp...@gmail.com>
Date: Sat, 10 Nov 2018 06:36:13 -0800
Subject: Re: "Equality operator appears too late"
Newsgroups: comp.lang.ada

> [...]

I should have included that in my original post. If I remove the generic components and add in type definitions instead, it compiles just fine.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Sun, 11 Nov 2018 00:32:29 -0600
Subject: Re: "Equality operator appears too late"
Newsgroups: comp.lang.ada

Use a different compiler? :-) There rarely is a *good* solution when dealing with a compiler bug.

>The error is occurring when the builder tries to elaborate the Ada.Containers. Unbounded_Priority_Queues package, where and >"=" operator is defined on line 76.

The language-defined specification of that package doesn't contain any "=" operator, so the existence of such a thing itself might be the bug. Note that Ada 2012 adopted rules for overriding of "=" for untagged record types similar to those for tagged record types (that was to allow composition to make sense), and thus some "=" declarations that were legal in previous Ada aren't legal anymore. Perhaps a recent GNAT tightened up these rules and caught some of their library code.

From: Simon Wright <si...@pushface.org>
Date: Sun, 11 Nov 2018 20:05:17 +0000
Subject: Re: "Equality operator appears too late"
Newsgroups: comp.lang.ada

> [...]

> The language-defined specification of that package doesn't contain any "=" operator, so the existence of such a thing itself might be the bug.

This is in the package Implementation.

The code compiles OK with GCC 6.1.0 but not with GCC >= 7 or GNAT >= 2016.

Aliased Unchecked_Unions

From: Lucretia <lague...@googlemail.com>
Date: Mon, 12 Nov 2018 09:56:03 -0800
Subject: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

I'm trying to binding the MPC parser library and have come upon an issue with unchecked_unions and aliased.

The error in my minimal sample is:

test.adb:36:07: prefix of "access" attribute must be aliased

Source:

```
procedure test is
   type Errors is null record with
     Convention => C;

   type Errors_Ptr is access Errors with
     Convention => C;

   type Values is null record with
     Convention => C;

   type Values_Ptr is access Values;

   type Results (Success : Boolean) is
record
      case Success is
         when False =>
            Error : Errors_Ptr;

         when True =>
            Output : Values_Ptr;
      end case;
   end record with
     Convention     => C_Pass_By_Copy,
     Unchecked_Union => True;

   type Results_Ptr is access all Results
with
     Convention => C;

   procedure B (R : in Results_Ptr) is
   begin
      null;
   end B;

   Result : aliased Results := (Success =>
False, others => <>);
begin
   B (Result'Access);
end test;
```

From: Lucretia <lague...@googlemail.com>
Date: Mon, 12 Nov 2018 10:29:05 -0800
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

```
>   type Results (Success : Boolean) is
>   record
>      case Success is
>         when False =>
>            Error : Errors_Ptr;
>
>         when True =>
>            Output : Values_Ptr;
>      end case;
>   end record with
>    Convention     => C_Pass_By_Copy;
```

This compiles fine. So it seems impossible to use unchecked_union and access types together.

From: Per Sandberg <per.s.s...@bahnhof.se>
Date: Mon, 12 Nov 2018 19:53:00 +0100
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

You could always do.

```
B (Result'Unrestricted_Access);
```

As a workaround

/P

From: Lucretia<lague...@googlemail.com>
Date: Mon, 12 Nov 2018 11:44:42 -0800
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

Thanks, that works.

From: "Dmitry A. Kazakov" <mai...@dmitry-kazakov.de>
Date: Mon, 12 Nov 2018 21:13:51 +0100
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

What about C_Pass_By_Copy? That looks much inconsistent with access to me.

BTW, why do you want to use unchecked union? In comparable cases I rather simply overload imported functions with whatever arguments:

```
procedure Foo (X : in out int);
procedure Foo (X : in out unsigned);
pragma Import (C, Foo);
```

Ada is so great that it can fix C faults... (:-))

From: Lucretia <lague...@googlemail.com>
Date: Mon, 12 Nov 2018 13:02:15 -0800
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

> [...]

> What about C_Pass_By_Copy? That looks much inconsistent with access to me.

I did try removing that too.

> BTW, why do you want to use unchecked union? In comparable cases I rather simply overload imported functions with whatever arguments:

Because it's not that simple.

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Mon, 12 Nov 2018 18:14:17 -0600
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

Sure, but it's almost always simple enough. We built Claw without using Unchecked_Union at all (it didn't exist in Ada 95), and there weren't many (if any) cases where the result would have been better had it existed.

Worse, Unchecked_Union is an easy way to introduce erroneous execution into one's program accidentally; at least using Unchecked_Conversion or some similar scheme makes that obvious. (Using the "wrong" discriminant is erroneous, even though that isn't checked or checkable.)

I'd keep the use of Unchecked_Union *very* limited, mostly to just cases where it is a component of some outer type. And even that might be better handled with Unchecked_Conversion (for reasons noted above).

From: "Randy Brukardt" <ra...@rrsoftware.com>
Date: Mon, 12 Nov 2018 18:17:01 -0600
Subject: Re: Aliased Unchecked_Unions not seen or taken notice of
Newsgroups: comp.lang.ada

> [...]

> Thanks, that works.

BTW, I'd report a bug to AdaCore, too, as there doesn't seem to be any reason for what you wrote to not work. (If there was, the error message should explain it, not complain that something explicitly declared as aliased isn't aliased -- so there still is a bug to be fixed.)

Conference Calendar

Dirk Craeynest

KU Leuven. 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.

2019

January 03-05 — 19th IEEE **International Symposium on High Assurance Systems Engineering** (HASE'2019), Hangzhou, China. Topics include: formal methods for high assurance systems engineering, domain specific languages, system verification and validation, high assurance systems development, cyber-physical systems, distributed systems, embedded, mobile, and real-time systems, safety, vulnerability, and fault tolerance, reliability, robustness, and resilience, modeling and simulation, security and privacy, autonomous systems and robotics, large-scale systems integration, space and communication systems, etc.

January 08-11 — 31st **Conference on Software Engineering Education and Training** (CSEET'2019), Grand Wailea, Maui, USA. Topics include: curriculum development; empirical studies; personal or institutional experience; team development; software assurance, quality, and reliability education; methodological aspects of software engineering education; global and distributed software development; open source in education; cooperation between industry and academia; etc.

☺ January 13-19 — 46th ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2019), Lisbon, Portugal. Topics include: all aspects of programming languages and programming systems.

> January 14-15 — ACM SIGPLAN **Workshop on Partial Evaluation and Program Manipulation** (PEPM'2019). Topics include: semantics based program synthesis and program optimisation; program and model manipulation techniques (such as partial evaluation, slicing, symbolic execution, refactoring, ...); techniques that treat programs/models as data objects (including metaprogramming, generative programming, embedded domain-specific languages, model-driven program generation and transformation, ...); program analysis techniques that are used to drive program/model manipulation (such as abstract interpretation, termination checking, type systems, test case generation, ...); application of the above techniques (including case studies of program manipulation in real-world (industrial, open-source) projects and software development processes, descriptions of robust tools capable of effectively handling realistic applications, benchmarking); etc.

January 15-18 — 11th **Software Quality Days Conference** (SWQD'2019), Vienna, Austria. Topics include: improvement of software development methods and processes; testing and quality assurance of software and software-intensive systems; domain specific quality issues such as embedded, medical, automotive systems; novel trends in software quality; etc.

♦ February 02 — **Ada Developer Room at FOSDEM 2019,** Brussels, Belgium. FOSDEM 2019 is a two-day event (Sat 2 - Sun 3 Feb). This years' edition includes once more a full-day Ada Developer Room, organized by Ada-Belgium in cooperation with Ada-Europe, which will be held on Saturday 2 February.

February 06-08 — 13th **International Workshop on Variability Modelling of Software-Intensive Systems** (VaMoS'2019), Leuven, Belgium. Topics include: variability management throughout the life cycle; variability-driven runtime adaptation; testing, formal reasoning and automated analysis on variability models; refactoring and evolution of variability intensive software systems; variability mining and reverse engineering approaches; software economic aspects of variability; etc.

February 14-16 12th **Innovations in Software Engineering Conference** (ISEC'2019), Pune, India.

February 16-17 28th **International Conference on Compiler Construction** (CC'2019), Washington DC, USA. Co-located with CGO'2019, HPCA'2019, and PPoPP'2019. Topics include: processing programs in the most general sense (analyzing, transforming or executing input that describes how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques; run-time techniques, including memory management, virtual machines, ...; programming tools, including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers; techniques for specific domains, such as secure, parallel, distributed, embedded, ... environments; design and implementation of novel language constructs, programming models, and domain-specific languages.

☺ February 16-20 24th ACM SIGPLAN **Symposium on Principles and Practice of Parallel Programming** (PPoPP'2019), Washington DC, USA. Co-located with HPCA'2019 and CGO'2019. Topics include: all aspects of parallel programming, including theoretical foundations, techniques, languages, compilers, runtime systems, tools, and practical experience; such as compilers and runtime systems, concurrent data structures, development, analysis, or management tools, formal analysis and verification, parallel programming languages, programming tools for parallel systems, software engineering for parallel programs, synchronization and concurrency control, etc.

February 24-27 26th IEEE **International Conference on Software Analysis, Evolution, and Reengineering** (SANER'2019), Hangzhou, China. Topics include: 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 maintenance and evolution; education related to all of the above topics; etc.

Feb 27 - March 02 50th ACM **Technical Symposium on Computer Science Education** (SIGCSE'2019), Minneapolis, Minnesota, USA.

March 18-21 25th **International Working Conference on Requirements Engineering: Foundation for Software Quality** (REFSQ'2019), Utrecht, the Netherlands.

March 25-28 14th **European Conference on Computer Systems** (EuroSys'2019), Dresden, Germany. Topics include: all areas of computer systems research; such as distributed systems; language support and runtime systems; systems security and privacy; dependable systems; parallelism, concurrency, and multicore systems; real-time, embedded, and cyber-physical systems; tracing, analysis, verification, and transformation of systems; etc. Event includes: 12th European Workshop on Systems Security (EuroSec 2019), 13th EuroSys Doctoral Workshop (EuroDW 2019), 9th Workshop on Systems for Multi-core and Heterogeneous Architectures (SFMA 2019), 2nd International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2019), 6th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC 2019). Deadline for submissions: January 3, 2019 (EuroSec 2019), January 7, 2019 (EuroDW 2019), January 17, 2019 (SFMA 2019), January 20, 2019 (EdgeSys 2019), January 23, 2019 (posters), January 24, 2019 (PaPoC 2019).

March 25-29 IEEE **International Conference on Software Architecture** (ICSA'2019), Hamburg, Germany. Topics include: model driven engineering for continuous architecting; component based software engineering and architecture design; re-factoring and evolving architecture design decisions and solutions; architecture frameworks and architecture description languages; preserving architecture quality throughout the system lifetime; software architecture for legacy systems and systems integration; architecting families of products; software architects roles and responsibilities; training, education, and certification of software architects; industrial experiments and case studies; etc. Deadline for submissions: January 17, 2019 (workshop papers), January 25, 2019 (tutorials). Deadline for early registration: February 28, 2019.

March 25-29 **Design, Automation and Test in Europe Conference** (DATE'2019), Firenze Fiera, Fortezza da Basso, Florence, Italy. Event includes: tracks on design methods & tools, application design, test and dependability, embedded and cyber-physical systems.

☺ April 01-04 **International Conference on the Art, Science, and Engineering of Programming** (Programming'2019), Genova, Italy. Topics include: programming practice and experience; general-purpose programming; distributed systems programming; parallel and multi-core programming; security programming; interpreters, virtual machines and compilers; modularity and separation of concerns;

model-based development; testing and debugging; program verification; programming education; programming environments; etc.

April 06-11 22nd **European Joint Conferences on Theory and Practice of Software** (ETAPS'2019), Prague, Czech Republic. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems).

☺ April 06-07 **VerifyThis Verification Competition** 2019. Topics include: no restrictions on programming language and verification technology used. Deadline for submissions: January 28, 2019 (ideas for verification challenges and problems).

April 07-11 10th ACM/SPEC **International Conference on Performance Engineering** (ICPE'2019), Mumbai, India. Deadline for submissions: January 11, 2019 (work-in-progress/vision papers), January 14, 2019 (posters/demos, tutorials).

April 08-12 34th ACM **Symposium on Applied Computing** (SAC'2019), Limassol, Cyprus.

☺ April 08-12 **Track on Programming Languages** (PL'2019). 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.

April 08-12 **Track on Software Verification and Testing** (SVT'2019). 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, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging, verification and validation of large scale software systems, real world applications and case studies applying software testing and verification, etc.

April 08-12 14th **Track on Dependable, Adaptive, and Trustworthy Distributed Systems** (DADS'2019). Topics include: Dependable, Adaptive, and trustworthy Distributed Systems (DADS); modeling, design, and engineering of DADS; foundations and formal methods for DADS; etc.

April 08-12 **Track on Next Generation Programming Paradigms and Systems** (NGPS'2019). Topics include: runtime verification and monitoring; secure and dependable software; formal models and verification; design, implementation and optimization of high-level programming languages; middleware platforms; scenarios, case studies and experience reports on innovative applications; high-level parallel programming; distributed systems and concurrency; development tools; security, trust and privacy management; etc.

April 08-12 **Embedded Systems Track** (EMBS'2019). Topics include: verification, validation, testing, debugging, and performance analysis of embedded systems; cyber physical systems; multicore, SoC-based, and heterogeneous embedded systems and applications; multithreading in embedded systems design and development; compilation strategies, code transformation and parallelization for embedded systems; reliability in embedded computing and systems; security within embedded systems and embedded systems for security; safety-critical embedded systems; case studies; etc.

April 15-17 23rd **International Conference on Evaluation and Assessment in Software Engineering** (EASE'2019), Copenhagen, Denmark. Topics include: evidence-based software engineering and its implications for software practice. Deadline for submissions: January 10, 2019 (research papers), January 18, 2019 (doctoral symposium plans), January 20, 2019 (emerging results and vision track abstracts), January 27, 2019 (emerging results and vision track papers, short papers and artefacts track abstracts, industry track experience report extended abstracts, industry track empirical studies abstracts), February 3, 2019 (short papers and artefacts track papers, industry track empirical studies papers), February 11, 2019 (industry track talk proposals, industry track panel proposals), February 28, 2019 (posters).

April 15-18 12th **Cyber-Physical Systems Week** (CPS Week'2019), Montreal, Canada.

> ☺ April 16-18 25th IEEE **Real-Time and Embedded Systems and Applications Symposium** (RTAS'2019). Topics include: research related to embedded systems or timing issues ranging from traditional hard real-time systems to embedded systems without explicit timing requirements, including latency-sensitive systems with informal or soft real-time requirements; original systems and applications, case studies, methodologies, and applied algorithms that contribute to the state of practice in the design, implementation, verification, and validation of embedded systems and time-sensitive systems (of any size); etc.

> April 16-18 10th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2019). Topics include: development of technologies, tools, and architectures for building CPS systems; design, implementation, and investigation of CPS applications; etc.

April 22-26 22nd **Ibero-American Conference on Software Engineering** (CIbSE'2019), La Habana & Varadero, Cuba. Event includes Software Engineering Track (SET) and Experimental Software Engineering Track (ESELAW). Deadline for submissions: February 4, 2019 (doctoral symposium).

April 22-27 12th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2019), Xi'an, China. Topics include: formal verification, model based testing, model checking, manual testing practices and techniques, security testing, software reliability, test automation, testability and design, testing and development processes, testing in specific domains (such as embedded, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc. Deadline for submissions: February 8, 2019 (PhD Symposium).

May 01-03 8th **International Conference on Fundamentals of Software Engineering** (FSEN'2019), Tehran, Iran. 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; software specification, validation, and verification; software architectures and their description languages; integration of formal and informal methods; component-based software systems; model checking and theorem proving; software verification; CASE tools and tool integration; industrial Applications; etc.

☺ May 07-09 22nd IEEE **International Symposium On Real-Time Distributed Computing** (ISORC'2019), Valencia, Spain. Topics include: object/component/service-oriented real-time distributed computing (ORC) technology, programming and system engineering (real-time programming challenges, ORC paradigms, languages, ...), trusted and dependable systems, system software (real-time kernel/OS, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, ...), applications (medical devices, intelligent transportation systems, industrial automation systems, Internet of Things and Smart Grids, embedded systems in automotive, avionics, consumer electronics, ...), system evaluation (performance analysis, monitoring & timing, dependability, fault detection and recovery time, ...), cyber-physical systems, etc. Deadline for submissions: January 15, 2019 (main track), March 7, 2019 (posters, demos).

May 07-09 11th NASA **Formal Methods Symposium** (NFM'2019), Houston, Texas, USA. Topics include: identify challenges and provide solutions for achieving assurance for critical systems; formal verification, including theorem proving, model checking, and static analysis; use of formal methods in software and system testing; run-time verification techniques and algorithms for scaling formal methods, such as abstraction and symbolic methods, compositional techniques, as well as parallel and/or distributed techniques; safety cases and system safety; formal approaches to fault tolerance; formal methods in systems engineering and model-based development; etc.

May 20-23 32nd **International Conference on Architecture of Computing Systems** (ARCS'2019), Copenhagen, Denmark. Focus: "architectures for complex real-time systems". Topics include: autonomous control systems, as well as safety and security critical systems; upcoming architectures and technologies, exploitable architectural features, languages, and tooling; architectures for real-time and mixed-criticality systems; programming models for many-core computing platforms; hypervisors and middleware for multi-/many-core computing platforms; support for safety and security; etc.

May 20-24 33rd IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2019), Rio de Janeiro, Brazil.

☺ April 20 24th **International Workshop on High-Level Parallel Programming Models and Supportive Environments** (HIPS'2019). Topics include: the areas of parallel applications, language design, compilers, runtime systems, and programming tools; the areas of emerging programming models for large-scale parallel systems and many-core architectures; new programming languages and constructs for exploiting parallelism/locality; experience with and improvements for existing parallel languages and run-time environments; parallel compilers, programming tools, and environments; programming environments for heterogeneous multicore systems and accelerators such as GPUs, FPGAs, and MICs; etc. Deadline for submissions: January 22, 2019 (full papers), January 29, 2019 (short papers).

April 25-31 41st **International Conference on Software Engineering** (ICSE'2019), Montréal, Québec, Canada. Theme: "The next 50 years for Software Engineering". Deadline for submissions: January 7, 2019 (ACM Student Research Competition), February 1, 2019 (workshop papers), February 7, 2019 (student volunteers).

June 03-07 31st **International Conference on Advanced Information Systems Engineering** (CAiSE'2019), Rome, Italy. Theme: "Responsible Information Systems". Topics include: methods, models, techniques, architectures and platforms for supporting the engineering and evolution of information systems and organizations.

☺ June 04-06 **DAta Systems In Aerospace** (DASIA'2019), Sicily, Italy.

♦ June 10-14 Ada-Europe 24th **International Conference on Reliable Software Technologies** (Ada-Europe 2019), Warsaw, Poland. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN, and the Ada Resource Association (ARA) Deadline for submissions: January 28, 2019 (regular papers, industrial presentation outlines, tutorial and workshop proposals).

June 26-28 18th **International Conference on Software Reuse** (ICSR'2019), Cincinnati, Ohio, USA. Topics include: approaches facilitating reuse in industry; technical debt and reuse; component-based reuse techniques; generative, systematic, and opportunistic reuse; reverse engineering of potentially reusable components; evolution and maintenance of reusable assets; dynamic aspects of reuse (e.g., post-deployment time); retrieval of reusable artifacts and knowledge in large-scale software repositories (e.g., open-source and industrial code bases); etc. Deadline for submissions: January 31, 2019 (papers), February 15, 2019 (Industry Innovation Track papers or extended abstracts).

July 09-12 31st **Euromicro Conference on Real-Time Systems** (ECRTS'2019), Stuttgart, Germany. Topics include: all aspects of real-time systems, such as scheduling design and analysis, real-time operating systems, hypervisors and middleware, memory management, worst-case execution time analysis, formal models and analysis techniques for real-time systems, mixed-criticality design and assurance, programming languages and compilers, virtualization and timing isolation, etc. Deadline for submissions: February 7, 2019 (papers).

☺ July 15-19 33rd **European Conference on Object-Oriented Programming** (ECOOP'2019), London, England. Topics include: original and unpublished results on any Programming Languages topic. Deadline for submissions: January 11, 2019 (papers).

July 15-19 43rd Annual IEEE **Conference on Computer Software and Applications** (COMPSAC'2019), Milwaukee, Wisconsin, USA. Deadline for submissions: open (workshops), January 21, 2019 (abstracts, full papers), April 15, 2019 (workshop papers).

July 29-31 13th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2019), Guilin, China. Topics include: theoretical aspects of software engineering, such as abstract interpretation, component-based software engineering, cyber-physical systems, distributed and concurrent systems, embedded and real-time systems, formal verification and program semantics, integration of formal methods, language design, model checking and theorem proving, model-driven engineering, object-oriented systems, program analysis, reverse engineering and software maintenance, run-time verification and monitoring, software architectures and design, software testing and quality assurance, software safety, security and reliability, specification and verification, type systems, tools exploiting theoretical results, etc. Deadline for submissions: January 4, 2019 (abstracts), January 11, 2019 (papers).

August 26-31	17th **International Conference on Formal Modeling and Analysis of Timed Systems** (FORMATS'2019), Amsterdam, the Netherlands. Topics include: theoretical foundations of timed systems and languages; methods and tools (techniques, algorithms, data structures, and software tools for analyzing timed systems and resolving temporal constraints, such as scheduling, worst-case execution time analysis, optimization, model checking, testing, constraint solving, ...); adaptation and specialization of timing technology in application domains in which timing plays an important role (real-time software, problems of scheduling in manufacturing and telecommunication, ...); etc. Deadline for submissions: April 21, 2019 (abstracts), April 24, 2019 (papers).
August 27-30	30th **International Conference on Concurrency Theory** (CONCUR'2019), Amsterdam, the Netherlands. Topics include: basic models of concurrency; verification and analysis techniques for concurrent systems, such as abstract interpretation, atomicity checking, model checking, race detection, run-time verification, static analysis, theorem proving, type systems, security analysis, ...; distributed algorithms and data structures; theoretical foundations of architectures, execution environments, and software development for concurrent systems, such as multiprocessor and multi-core architectures, compilers and tools for concurrent programming, programming models such as component-based, object-oriented, ...; etc. Deadline for submissions: April 15, 2019 (abstracts), April 22, 2019 (papers).
August 28-30	45th **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2019), Thessaloniki / Chalkidiki, Greece. Topics include: information technology for software-intensive systems; conference tracks on Embedded Systems & Internet of Things (ES-IoT), Software Process and Product Improvement (SPPI), etc.; special sessions on Cyber-Physical Systems (CPS), Software Engineering and Technical Debt (SEaTeD), Model-Driven Engineering and Modeling Languages (MDEML), etc. Deadline for submissions: March 1, 2019 (abstracts), March 15, 2019 (papers).
☺ Sep 10-13	**International Conference on Parallel Computing** 2019 (ParCo'2019), Prague, Czech Republic. Deadline for submissions: February 28, 2019 (extended abstracts), March 31 2019 (mini-symposia), July 31, 2019 (full papers).
September 16-20	17th **International Conference on Software Engineering and Formal Methods** (SEFM'2019), Oslo, Norway. Deadline for submissions: January 11, 2019 (workshops).
October 07-11	23rd **International Symposium on Formal Methods** (FM'2019), Porto, Portugal, aka 3rd World Congress on Formal Methods. Topics include: formal methods in a wide range of domains including software, computer-based systems, systems-of-systems, cyber-physical systems, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, and healthcare; formal methods in practice (industrial applications of formal methods, experience with formal methods in industry, tool usage reports, ...); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, ...); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, ...); etc.
December 03-06	40th IEEE **Real-Time Systems Symposium** (RTSS'2019), Hong Kong.
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

ptc® apexada | ptc® objectada®

Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: **ptc.com/developer-tools**

ptc

Call for Participation 9th Ada Developer Room at FOSDEM 2019 Saturday 2 February 2019, Brussels, Belgium

Organized by Ada-Belgium in cooperation with Ada-Europe

FOSDEM[1], the Free and Open source Software Developers' European Meeting, is a non-commercial two-day weekend event organized early each year in Brussels, Belgium. It is highly developer-oriented and brings together 8000+ participants from all over the world. The 2019 edition takes place on Saturday 2 and Sunday 3 February. It is free to attend and no registration is necessary.

In this edition, Ada-Belgium[2] organizes once more a full day of presentations related to Ada and Free or Open Software in a s.c. Developer Room. The "Ada DevRoom" at FOSDEM 2019 is held on the first day of the event. The program offers introductory presentations on the Ada programming language, as well as more specialised presentations on focused topics, tools and projects. This year FOSDEM has a total of 15 Ada-related presentations by 12 authors from 7 countries!

Program overview:

- *Welcome to the Ada DevRoom*, by Dirk Craeynest, Ada-Belgium
- *An Introduction to Ada for Beginning and Experienced Programmers*, by Jean-Pierre Rosen, Adalog
- *Sequential Programming in Ada: Lessons Learned*, by Joakim Strandberg, Mequinox
- *Autonomous Train Control Systems: a First Approach*, by Julia Teissl, FH Campus Wien
- *Controlling the Execution of Parallel Algorithms in Ada*, by Jan Verschelde, Univ. of Illinois at Chicago
- *Persistence with Ada Database Objects*, by Stephane Carrez, Twinlife
- *Shrink your Data to (almost) Nothing with Trained Compression*, by Gautier de Montmollin, Ada-Switzerl.
- *GSH: an Ada POSIX Shell to Speed Up GNU Builds on Windows*, by Nicolas Roche, AdaCore
- *What is Safety-Critical Software, and How Can Ada and SPARK Help?*, by Jean-Pierre Rosen, Adalog
- *Secure Web Applications with AWA*, by Stephane Carrez, Twinlife
- *Distributed Computing with Ada and CORBA using PolyORB*, by Frédéric Praca, Ada-France
- *Cappulada: Smooth Ada Bindings for C++*, by Johannes Kliemann, Componolit
- *The Azip Archive Manager: a full-Ada Open-Source Portable Application*, by G. de Montmollin, Ada-Swit.
- *Proof of Pointer Programs with Ownership in SPARK*, by Yannick Moy, AdaCore
- (in RISC-V room) *Alternative Languages for Safe&Secure RISC-V Programming*, by F.Chouteau, AdaCore

The Ada at FOSDEM 2019 web-page has all details, such as the full schedule, abstracts of presentations, biographies of speakers, and pointers to more info. For the latest information at any time, contact <Dirk.Craeynest@cs.kuleuven.be>, or see:

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/19/190202-fosdem.html

1https://fosdem.org/2019
2http://www.cs.kuleuven.be/~dirk/ada-belgium

Ada-Europe

24th International Conference on Reliable Software Technologies

10-14 June 2019, Warsaw, Poland

Conference & Program Chair

Tullio Vardanega
University of Padua, Italy
tullio.vardanega@unipd.it

Educational Tutorial & Workshop Chair

Dene Brown
SysAda Ltd, UK
dene.brown@sysada.co.uk

Industrial Chair

Maurizio Martignano
Spazio IT, Italy
maurizio.martignano@spazioit.com

Exhibition & Sponsorship Chair

Ahlan Marriott
White Elephant GmbH, Switzerland
software@white-elephant.ch

Publicity Chair

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

Local Chair

Maciej Sobczak
GE Aviation – EDC Warsaw, Poland
maciej.sobczak@ge.com

General Information

Ada-Europe is pleased to announce that its 24th International Conference on Reliable Software Technologies (Ada-Europe 2019) will take place in Warsaw, Poland. The conference schedule at its fullest includes a three-day technical program and vendor exhibition from Tuesday to Thursday, and parallel tutorials and workshops on Monday and Friday. This edition of the conference inaugurates a major revamp in the registration fees, redesigned to extend participation from industry and academia, and to reward contributors, especially but not solely, students and post-doc researchers.

Schedule

28 January 2019	Submission of papers, industrial presentation outlines, tutorial and workshop proposals (**extended deadline, final**)
1 March 2019	Notification of acceptance to all authors
16 March 2019	Camera-ready version of papers required
30 April 2019	Industrial presentations, tutorial and workshop material required

Topics

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

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

- Design and Implementation of Real-Time and Embedded Systems,
- Design and Implementation of Mixed-Criticality Systems,
- Theory and Practice of High-Integrity Systems,
- Software Architectures for Reliable Systems,
- Methods and Techniques for Quality Software Development and Maintenance,
- Ada Language and Technologies,
- Mainstream and Emerging Applications with Reliability Requirements,
- Achieving and Assuring Safety in Machine Learning Systems,
- Experience Reports on Reliable System Development,
- Experiences with Ada.

Refer to the conference website for the full list of topics.

acm In-Cooperation

SIGAda SIGPLAN SIGBED

Ada Resource ARA Association

http://www.ada-europe.org/conference2019

Program Committee

Mario Aldea, Univ. de Cantabria, ES
Johann Blieberger, Vienna Univ. of Technology, AT
Bernd Burgstaller, Yonsei Univ., KR
António Casimiro, Univ. Lisboa, PT
Barbara Gallina, Mälardalen Univ., SE
Michael González Harbour, Univ. de Cantabria, ES
J. Javi Gutiérrez, Univ. de Cantabria, ES
Jérôme Hugues, ISAE, FR
Hubert Keller, Karlsruhe Institute of Technology, DE
Raimund Kirner, Univ. of Hertfordshire, UK
Franco Mazzanti, ISTI-CNR, IT
Laurent Pautet, Telecom ParisTech, FR
Luís Miguel Pinho, CISTER/ISEP, PT
Erhard Plödereder, Univ. Stuttgart, DE
Juan A. de la Puente, Univ. Pol. de Madrid, ES
Jorge Real, Univ. Pol. de València, ES
José Ruiz, AdaCore, FR
Sergio Sáez, Univ. Pol. de València, ES
Elad Schiller, Chalmers Univ. of Technology, SE
Frank Singhoff, Univ. de Bretagne Occidentale, FR
Jorge Sousa Pinto, Univ. of Minho, PT
Tucker Taft, AdaCore, USA
Elena Troubitsyna, Åbo Akademi Uni., FI
Santiago Urueña, GMV, ES
Tullio Vardanega, Univ. of Padua, IT
Marcus Völp, Univ. of Luxembourg, LU

Industrial Committee

Ian Broster, Rapita Systems, UK
Dirk Craeynest, Ada-Belgium & KU Leuven, BE
Gonçalo Gouveia, Critical Techworks, PT
Thomas Gruber, Austrian Institute of Technology, AT
Andreas Jung, European Space Agency, NL
Ismael Lafoz, Airbus Military, ES
Patricia Lopez Cueva, Thales Alenia Space, FR
Ahlan Marriott, White Elephant, CH
Maurizio Martignano, Spazio-IT, IT
Silvia Mazzini, Intecs, IT
Marco Panunzio, Thales Alenia Space, FR
Paul Parkinson, Wind River, UK
Jean-Pierre Rosen, AdaLog, FR
José Emilio Salazar Marsà, GMV, ES
Helder Silva, Edisoft, PT
Jacob Sparre Andersen, JSA Cons., DK
Pawel Zakrzewski, GE Aviation, PL

Ada Europe 2019

Call for Regular Papers

The regular papers submitted to the conference must be original and shall undergo anonymous peer review. The authors shall submit their work by 28 January 2019, in PDF only, and up to 16 LNCS-style pages in length, via *https://easychair.org/conferences/?conf=ae2019*.

The conference is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar. The authors of the papers that will appear in the conference proceedings will be invited to extend their work for submission to a Special Issue of Elsevier's Journal of Systems Architecture, centered on the conference themes.

Proceedings

The conference proceedings will appear in Springer's Lecture Notes in Computer Science (LNCS) series, and will be available at the conference, both online and in print. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, strictly by 16 March 2019. For format and style guidelines, the authors should refer to *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register at least one author for the conference by that date will prevent the paper from appearing in the proceedings.

Call for Industrial Presentations

The conference seeks industrial presentations that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, of at least 1 page in length, by 28 January 2019, strictly in PDF, via *https://easychair.org/conferences/?conf=ae2019*.

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The authors of accepted contributions shall be requested to submit a 2-page abstract by 16 March 2019, for inclusion in the conference booklet, and be invited to deliver a 20-minute talk at the conference. These authors will also be invited to expand their contributions into articles for publication in the *Ada User* Journal (*http://www.ada-europe.org/auj/*), as part of the proceedings of the Industrial Program of the Conference. For any further information, please contact the Industrial Chair directly.

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Educational Tutorials

The conference seeks tutorials in the form of educational seminars that may include hands-on or practical demonstrations. Proposed tutorials can address any part of the reliable software domain, and may have an academic or industrial slant from technology perspective. All software topics and their application to reliability and safety are welcome. 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), and the intended level of the tutorial (introductory, intermediate, or advanced), and most importantly a statement expressing why it will be worthwhile to attend the tutorial. Tutorial proposals shall be submitted to the Educational Tutorial Chair.

The authors of accepted full-day tutorials will receive a complimentary conference registration. For half-day tutorials, this benefit is halved. The *Ada User Journal* will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

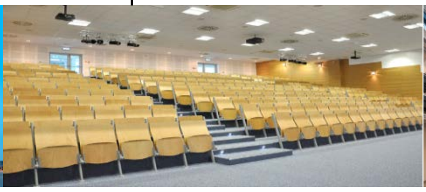
Workshops on themes within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference week. Workshop proposals shall be submitted to the Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the *Ada User Journal*.

Call for Exhibitors

The commercial exhibition will span the core days of the main conference. Interested providers of software products and services should send inquiries to the Exhibition Chair.

Venue

The conference will take place in Warsaw, Poland, at the Engineering Design Center, partnership of General Electric and the Institute of Aviation, one of Europe's largest engineering institutions.



AGILE-R[1]: Agile Software Development for Railways

J. Favaro, G. Ioele, A. Jaku, S. Mazzini, P. Panaroni

Intecs Solutions, Via Umberto Forti 5, Loc. Montacchiello - 56121 Pisa; Italy, Tel: + 39 050 9657 411; email: {John.Favaro, Guido.Ioele, Aida.Jaku, Silvia.Mazzini, Paolo.Panaroni}@intecs.it

U. Paone

Intecs Solutions, Via Giacomo Peroni 130 - 00131 Roma; Italy, Tel: + 39 06 20392 800; email: Umile.Paone@intecs.it

Abstract

In this paper we present AGILE-R, a Scrum based approach defined by Intecs Solutions to combine Agile and EN 50128 for Railway software development.

Keywords: Railway, Agile, Safety, Embedded Systems.

1 Introduction

Agile approaches have their roots in 2001 with the elaboration of the famous “Agile Manifesto”. Subsequently, it gradually gained in popularity over the so called “heavy processes” (waterfall based, the champion being CMMI) and Agile is now the most adopted software development approach worldwide. Even the proponents of CMMI are now proposing a “marriage” with Agile [3].

However, for over a decade there has been much controversy with respect to Agile in real time and safety critical software domains such as avionics, space, railway, automotive, medical devices, etc. Today, however, there are many success stories and considerable experience proving that Agile is not in contradiction with highly critical software, on the condition that agility is applied with rigor and discipline.

“Barriers to using Agile no longer exist. Developments in globally distributed teams, large projects, safety-critical systems, and hardware and systems engineering have shown that Agile technologies are adoptable and adaptable.” is reported in IEEE Software Magazine [10].

An Agile Software Development Handbook [5] has been developed by the European Cooperation for Space Standardization (ECSS), with the support of the European Space Agency, providing detailed guidelines and advice for the adoption of the Scrum software development approach in those space projects where ECSS-E-ST-40 and ECSS-Q-ST-80 are applicable. Detailed mappings between requirements and agile practices have been reported for the avionics sector, while adoption of agile development methods is reported by the NASA Ames Research Center for the development of mission control technology software [6]. Recently a Norwegian study from SINTEF ICT and NTNU has proposed Agile for CENELEC EN 50128, by defining the SafeScrum variant of Scrum with some of the CENELEC requirements outside of the agile approach and some safety requirements added to the agile methodology, together with the involvement of the assessor as early as possible to reduce certification costs [8].

In this paper we present AGILE-R, a practical approach adopted by Intecs Solutions for the application of Scrum to the development of Railway software in accordance with the EN 50128 standard (at least up to safety integrity level SIL2) [1].

AGILE-R has been elaborated by an Intecs Solutions team combining diverse and complementary sets of expertise, including Software Methodologies, Safety Assessments, Quality Assurance, CENELEC Standards, Agile, Scrum, and Project Management. The results have been shared and discussed with external Independent Safety Assessors.

2 Goals

The main goals of AGILE-R are:

- to reduce time to market and improve responsiveness to change, without sacrificing safety and quality. This is mainly achieved by shortening the time between development and bug fixing. Every increment is fully tested and validated. Regression risks are also reduced. While Agile does not reduce the number of total tests to be executed, it distributes them over manageable quantities for any given increment. In this way, the high costs and delays associated with large and late integration of software are avoided.
- to increase the control and predictability of the development process itself, by forcing visible and tangible results at fixed intervals. Progress is measured by the state of the product rather than estimations and presentations.
- to decrease the risk of producing unsatisfactory solutions, with strong involvement of the product owner.

[1] Trademark registered

3 Background

Scrum life cycle (www.scrumprimer.org) is a time-sequenced process (a life cycle) with incremental/iterative time-boxed deliveries as shown in Figure 1.

EN-50128 is a logical sequence of processes (a process model) transforming customer requirements into deliverable software as shown in Figure 2.

From INTECS experience in SW Engineering and ISA's feedbacks some fundamental aspects arise:

- Scrum defines HOW to manage software development projects, it is not a new standard.
- Scrum does not impose specific work products.
- Scrum is not in contradiction with WHAT is required by EN50128.
- Scrum does not sacrifice quality (quality is usually better thanks to early detection of bugs and pair programming).
- Few adaptations are required to best combine the two approaches and achieve the right Balance of Agility and Discipline.

They are not in contradiction: the EN-50128 process model may be executed in an incremental/iterative life cycle like that proposed by Scrum [4].

4 Scrum and EN-50128 in action

Figure 3 provides a high level overview of the recommended Agile-R approach for combining Scrum and the application of EN-50128.

The phases (building blocks) of Agile-R are detailed in the following:

System Level Planning and Analysis
This phase is not detailed in Figure 3 as it is out of scope of Agile-R. Agile-R is proposed only for software development and not at system level. All risk and safety analyses at the system level are performed outside the Agile process, including the analysis needed to determine the SIL level.

Planning (red block in Figure 3)
This initial managerial process is essential to coordinate the software development with all affected stakeholders. At this stage all plans (e.g. quality plan, verification plan, validation plan, etc.) are elaborated, the Scrum team is established and proper tools made available.

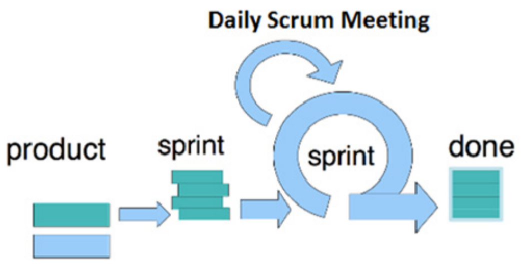


Figure 1 The Scrum life cycle

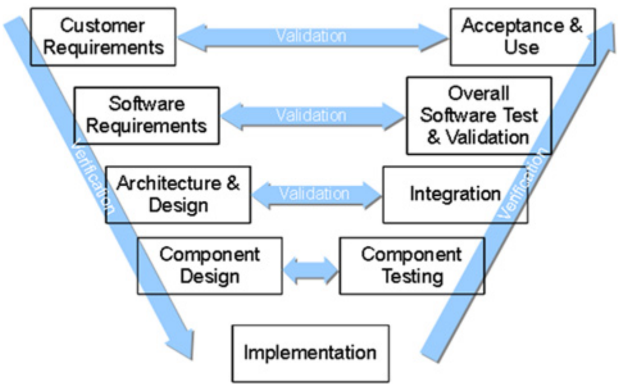


Figure 2 The EN-50128 life cycle

Sprint 0 (warm up sprint) (orange block in Figure 3)
This is a special initial sprint intended to define a preliminary overall architecture and implement some basic software. Sprint 0 is not a new concept, it provides solid foundations for all other sprints.

Sprint N (blue blocks in Figure 3)
This is the development heartbeat, implementing an increment of functionality, fully tested. High-level requirements (user stories) assigned to that sprint are fully implemented.

Integration (black blocks in Figure 3)
At given planned and coordinated events, the software developed in a Sprint may be integrated with software developed with Sprints of other Scrums running in parallel, other available software and/or integrated on the available HW resources. It means that one or a combination of the following integrations may be performed:

- The software with the software of other Scrums;
- The software with other available software;
- The software on the available HW resources;
- The Overall integrated software on the available HW resources.

Release (dark blue blocks in Figure 3)
These are special finalization phases where the software is wrapped up ready to be delivered. Final safety and quality checks shall be performed during these phases.

5 Independent Testing

EN-50128 requires testing to be specified and executed by personnel independent from the development team (the type of independence to a greater or lesser extent depends on the SIL).

We recommend that testing of implemented user stories remains assigned to independent test staff different from developers. We do not stress full organization independence (e.g. a separate test department or a separate company) but at least people independence (i.e. staff shall never test its own developed code). This is compliant with SIL 2 requirements.

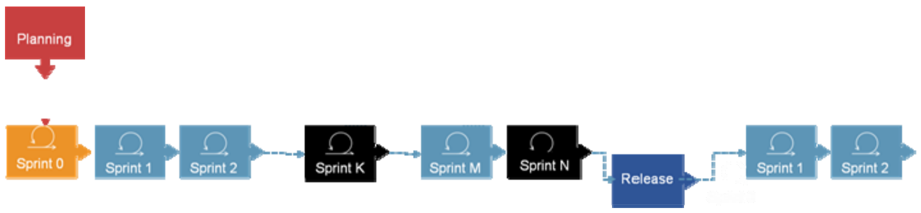


Figure 3 Agile-R High Level Overview

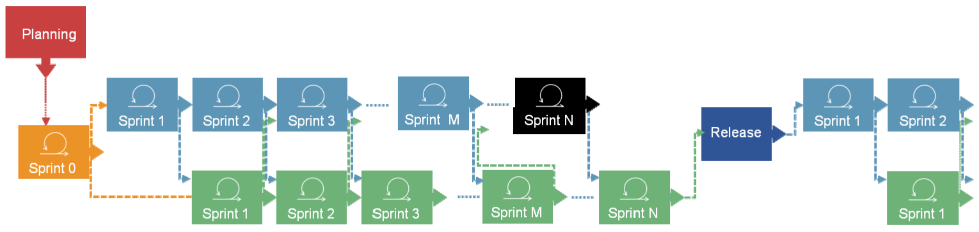


Figure 4 Agile-R for SIL3-4 Development

For higher SILs, where independent test teams get involved, we propose to allow independent tests to be run within validation sprints phased with development sprints as depicted in Figure 4.

The output of a development sprint becomes the input of an independent validation sprint. The possible problems identified by the test sprints will be formalized and entered into the product backlog (as "bug-to-be-fixed") and prioritized and managed by the ordinary development sprint planning as detailed in Figure 5.

The bugs can be fixed as soon as possible at the immediate next development sprint but may even be deferred to a future development sprint.

A running development sprint should not be affected (and distracted) by incoming bugs from validation sprints.

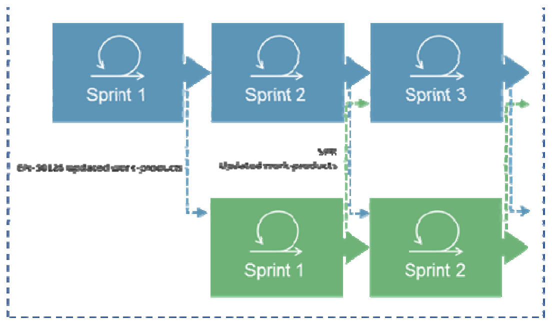


Figure 5 AGILE-R for SIL3-4 - Details

6 Process roles

Scrum process roles are:

- Scrum Master.
- Product Owner.
- Development Team.

EN 50128 Process roles are:

- Requirements Manager (part of development team) (RQM).
- Designer (part of development team) (DES).
- Implementer (part of development team) (IMP).
- Tester (part of development team, with independence's degree related to SIL) (TST).
- Integrator (part of the development team) (INT).
- Verifier (part of development team, with independence's degree related to SIL) (VER).
- Validator (preferably external to the development team, with independence's degree related to SIL) (VAL).
- Assessor (external to the organization) (ASR).
- Project Manager (PM).
- Configuration Manager (CM).
- Quality Engineer (SQA) [2].

The Agile-R roles are detailed in the following table:

Agile-R® Role	EN-50128 Role
Product Owner [PO]	**[N\A]: mapped on the System Engineer**
Scrum Master [SM]	**PM but with major orientation to collaborative work**
Development Sprint Team	**RQM, DES, IMP, INT, TST, VER**
Validation Sprint Team	**VAL**

Note: the EN-50128 Roles distribution inside the Development and Validation Sprint Team could change according to the tailoring of the methodology but always taking into account EN-50128 clauses on independency.

7 Parallel work (Scrum of Scrums)

It is well known that the agile approach does not scale well with large teams (e.g. > 7 people). This is often circumvented by running separate Scrum teams in parallel and setting some "integration/coordination" steps.

[2] *not directly covered by EN-50128 but required by quality*

The aim of these intermediate steps is to agree solutions to interfaces between teams and to negotiate responsibility boundaries, for continuous improvement of the between-team coordination.

AGILE-R relies on the Scrum of Scrums [12] for parallel work.

Many Scrums may run in parallel even working on the same sub-system, assuming given intermediate integration points and final integration and wrap-up for release are defined, as depicted in Figure 6.

However it may be also the case that different Scrum teams work on the same sub-system but with different objectives (e.g. different user stories to be implemented).

While some front-end planning and a proper architecture may minimize interferences among the teams assigning disjoint tasks, it is always required an integration (reconciliation) phase. This is the same pattern as the branch-merge in version management.

A recommendation is to hold a Scrum of Scrums periodic meeting where each Scrum is represented by an "ambassador" and integration issues are discussed.

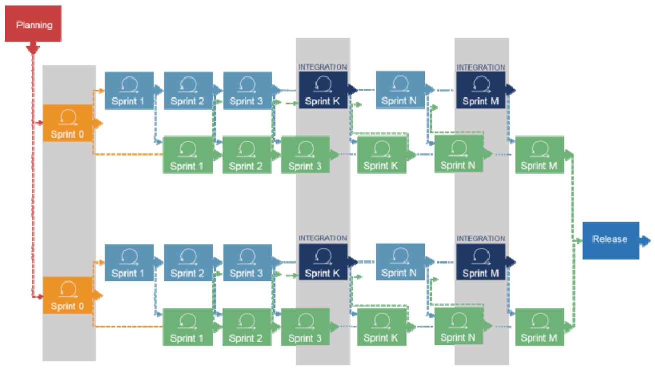


Figure 6 Agile-R for Large Projects

8 Methods, Techniques and Tools

It is of paramount importance to speed up the process of rapid evolution of project documentation through frequent iterations and refactoring. This is best achieved using models (e.g. SysML, UML) rather than plain textual documents. The documents (required by EN-50128 assessor) are then largely automatically generated from models.

A complete list of tools to be used (the so-called toolchain) has to be identified at the planning step with proper qualification actions.

9 AGILE-R in practice

A case study for the application of AGILE-R was defined in the context of the Railway Sirio-LX product. Sirio-LX is an automatic radar-based system for preventing trains from colliding with obstacles on the track at level crossings (see Figure 7). Sirio-LX is designed to ensure the highest level of safety standard CENELEC SIL4.

The experiment has been the development of a software part outside the official development, not starting from scratch, with the execution of 1 week initial planning phase and Sprint 0 and 2 Sprints with time box of 3 weeks. The main goal was to tune the approach getting learning lessons «from the battlefield» and remove some skepticism.

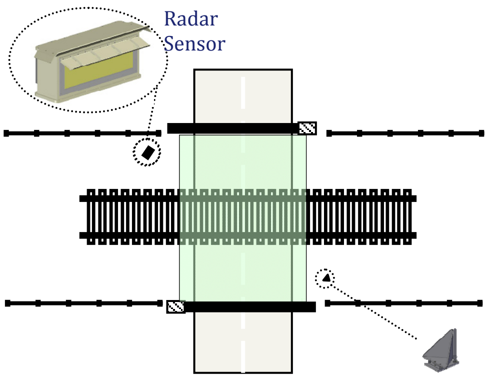


Figure 7 The SIRIO LX Use Case

The impact on EN50128 planning phase was minimal, the approach was welcomed by the team with no resistance. Education on Agile and Scrum principles was straightforward. Globally the staff reported a positive experience. Respect of budget and schedule was maintained.

The use of AGILE-R has confirmed the following:

- Agile is a way to manage the software development life cycle, not a different standard.
- Agile does not impose specific new work products, and all documents of EN 50128 have been adopted to ensure compliance.
- There are no contradictions between the application of AGILE-R and formal assessments.
- Agile does not sacrifice safety and quality (these are even better thanks to early detection of bugs and pair programming).

Project pitfalls, such as wrong or simplistic design, poor tools, and immature test environment, have an impact in the same way as with the traditional approach but you learn it after a short period of time and you can implement some counter-measures in the early stages of the project.

10 Conclusion

AGILE -R has confirmed that: "Barriers to using Agile no longer exist. Developments in globally distributed teams, large projects, safety-critical systems, and hardware and systems engineering have showed that agile technologies are adoptable and adaptable." [11]

Only few adaptations have been recommended to best combine the Scrum and EN 50128 approaches, with the right Balance of Agility and Discipline [2].

Acknowledgement

The authors would like to thanks the colleagues of Intecs that have contributed with their effective review and suggestions, in particular Marco Casucci, Giancarlo Gennaro, Alessandro Brachini and Simone Gianfranceschi.

References

[1] CENELEC (2011), *CENELEC EN 50128:2011 Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems*.

[2] B. Boehm, R. Turner (2004), *Balancing Agility with Discipline, A Guide for the perplexed*, Addison Wesley, ISBN 0-321-18612-5.

[3] CMMI Institute (2008), *CMMI & Agile: why not embrace both!*

[4] J. L. Boulanger (2015), *CENELEC 50128 and IEC 62279 Standards*, Wiley-ISTE.

[5] ECSS (2013), *ECSS-EHB-40A Software engineering handbook*.

[6] J. Trimble, C. Webster (2012), *Agile Development Methods for Space Operations*, Proceedings of SpaceOps 2012 Conference.

[7] S. H. VanderLeest, A. Buter (2009), *Escape the Waterfall: Agile for Aerospace*, Proc of the IEEE/AIAA 28th Digital Avionics Systems Conference.

[8] T. Myklebust, T. Stålhane, N. Lyngby (2015), *Application of an Agile Development Process for EN50128/railway conformant Software*, Proceedings of the 25th European Safety and Reliability Conference.

[9] C. Scholz (2014), *Agile Software Development compliant to Safety Standards?*, Proceedings of the 19th Ada Europe International Conference on Reliable Software Technologies.

[10] C. Ebert, M. Paasivaara (2017), *Scaling Agile*, IEEE Software 2017 (issue 6), p. 98.

[11] IEEE (2017), IEEE Software Magazine, November/December Issue.

[12] J. Sutherland (2001), *Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies*, IT Journal Vol. 14, No 12.

Tools to get you there. Safely.
Ada and The GNAT Pro High-Integrity Family
www.adacore.com
AdaCore
The GNAT Pro Company

ENABLE-S3: On Improving the Verification and Validation of Automated Cyber-Physical Systems

Andrea Leitner
AVL List GmbH, Austria; email: Andrea.Leitner@avl.com
Sergio Sáez-Barona
Instituto Tecnológico de Informática, Spain; email: ssaez@iti.es
Zora Slavik
FZI Research Center for Information Technology; email: slavik@fzi.de
Mika Rautila
VTT Technical Research Centre of Finland, Finland; email: Mika.Rautila@vtt.fi
Nadja Marko
Virtual Vehicle Research Center GmbH; email: Nadja.Marko@v2c2.at
Philipp Rosenberger
Technical University Darmstadt, Germany; email: rosenberger@fzd.tu-darmstadt.de
Dejan Ničković, Willibald Krenn
AIT Austrian Institute of Technology GmbH, Austria; email: {Dejan.Nickovic,willibald.krenn}@ait.ac.at
Michael Siegel
OFFIS e.V.; email: michael.siegel@offis.de

Abstract

ENABLE-S3 is an industry-driven project and aspires to substitute today's cost-intensive verification & validation efforts by more advanced and efficient methods to pave the way for the commercialization of highly Automated Cyber Physical Systems (ACPS). Pure simulation cannot cover physics in detail due to its limitations in modeling and computation. Real-world tests are too expensive, too time consuming and potentially dangerous. Thus, ENABLE-S3 aims at developing an innovative solution capable of combining both worlds in an optimized manner.

1 Introduction

Technologically the development of automated systems (such as automated driving) is satisfactorily understood and witnessed e.g. by millions of test kilometers already traveled by automated cars on public roads. These new technologies are leading to greater safety, lesser accidents as well as more efficient and environmentally friendly traffic. ADAS as well as automated driving (AD) systems are becoming an irreplaceable part of the everyday driving experience. Similar statements are true for other domains as well.

Nevertheless, Watzenig *et.al.* [1] state that new validation methodologies, procedures, and laws are needed in order to successfully incorporate emerging technologies into traffic and thus improve safety, reduce emissions, provide traffic flow optimization and enhanced mobility. Some steps towards this goal have already been taken. The EU made legal obligations on new passenger cars to include certain safety-related ADAS systems (EPS, EBA) and the level of automation will increase in the following years.

However, demonstrating the reliability, safety, and robustness of the technology in all conceivable situations, e.g. in all possible situations under all potential environmental conditions, has been identified as the main roadblock for product homologation, certification and thus commercialization. For automated driving, Winner *et.al.* [2] as well as Wachenfeld *et.al.* [3] predict that more than 100 million km of road driving would be required to statistically prove that an automated vehicle is as safe as a manually driven one. This means that a proven-in-use certification is simply not feasible by physical tests. OEMs currently mainly rely on proving ground or public road testing in order to validate their systems because of the lack of alternatives. ENABLE-S3 proposes a scenario-based virtual V&V approach. This means that more and more aspects are represented in a virtual environment in terms of models. The input for the testing are scenarios. The test scenarios are usually taken from collections generated by engineers, which include the complete scenario description together with the expected response of the system. However, even when utilizing these collections one cannot prove that the system will not fail in a test scenario that was not previously covered. For higher levels of automation the system cannot easily hand over the responsibility to a human, which means that the system needs to reliably handle even unknown situations. Proving ground and real world testing is associated with high costs, low reproducibility and long validation times. Especially reproducibility in a real world setup is challenging because of the difficulty to reach correct initialization, exact traffic behavior, similar environmental influences, and so on. Furthermore, safety is a very important aspect and further limitations arise because some test cases could be dangerous or even impossible to be carried out by human drivers. All these limitations add up and influence the overall time needed to successfully validate an automated system.

Taking further into account the high number of system variants and software versions, it becomes obvious that new approaches are required to validate automated systems within a reasonable time period at reasonable costs. New approaches are needed to reduce the effort required by today's state-of-the-art practices by orders of magnitude in order to become economically acceptable. ENABLE-S3 is an ECSEL JU funded project, which aims for overcoming these challenges.

2 Scope of the project

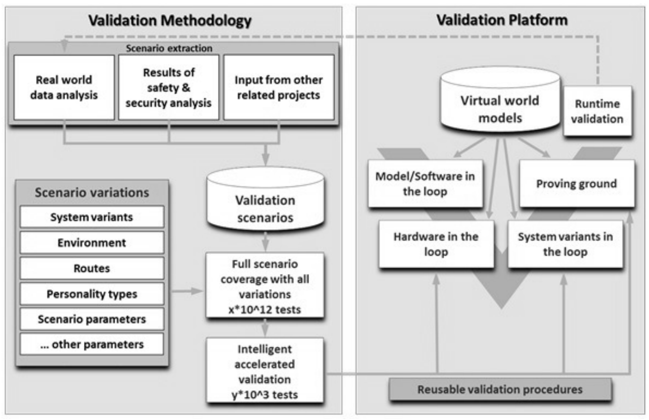


Figure 1: Main scope of the ENABLE-S3 project

Figure 1 shows the main scope of the project: the development of a modular framework for validation and verification of automated systems. The goal is not to have one platform which is capable of solving all problems, but to have reusable technology bricks (tools, methods, models, etc.), which can be used to build up a testing environment for a certain use case. Summarized, this requires covering the aspects of virtualization, modularization, as well as standardization.

Because of the large scope and complexity of the problem, it has been split into two parts. The validation methodologies on the one hand side describe the necessary steps and research on data acquisition and storage, scenario and metrics selection, as well as test generation methods. Since the project is aiming for a scenario-based validation approach, scenarios are an integral aspect. There is a huge number of potential scenarios that are either extracted from recorded data (real world data) or generated synthetically (e.g. based on safety and security analysis). In reality a lot of variations for these scenarios exist (i.e. for different environmental conditions, different persons/traffic participants involved, etc.) leading to an enormous number of test cases. The goal is to provide intelligent methods to select the required test scenarios in a way that ensures sufficient test coverage.

The validation platform on the other side focuses on reusable validation technology bricks, which are able to seamlessly support various development and testing environments (model-in-the-loop, hardware-in-the-loop, system-in-the-loop, e.g. vehicle-in-the-loop, as well as real-world testing). By combining both parts and their respective technology bricks the project aims for a significant reduction of the required test effort.

3 Generic Test Architecture

One major goal of the ENABLE-S3 project is to deliver reusable technology bricks and seamless development environments. The first promotes the development of models and tools that are easily reusable in different contexts. The latter requires to set up a testing environment where virtual representations can easily be exchanged by physical components. For both, the use of a modular structure with well-defined interfaces is essential. In order to achieve this goal, one main result of the first project year is the definition of a generic ENABLE-S3 test architecture. This architecture aims for supporting the integration of different technology bricks in a concrete test system instance. It consists of three main parts and includes the most essential parts for testing automated cyber-physical systems (ACPS), which will be elaborated in more detail in the following. The architecture is also independent of the domain and is therefore applicable for all six ENABLE-S3 application domains (automotive, aerospace, rail, maritime, health care and farming). The concrete characteristics of the blocks depend on the specific use cases. For some use cases, the blocks might be interpreted slightly different or are not required at all.

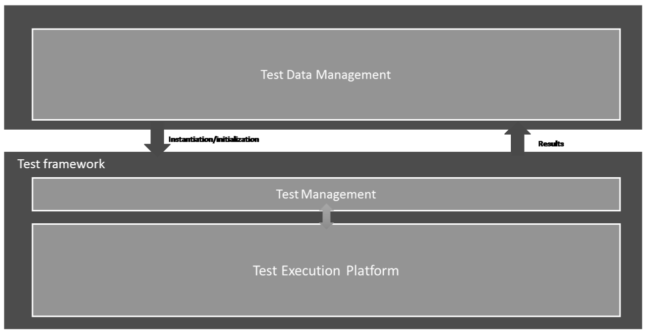


Figure 2: ENABLE-S3 Generic Test Architecture

Figure 2 shows the single blocks, which are described in more detail in the following.

On a high level, we distinguish between the Test Framework and the Test Data Management. The Test Data Management covers all aspects, which are valid across test phases and are reusable for testing different products. The test framework summarizes all aspects required for the planning (Test Management) and execution of tests (Test Execution Platform).

3.1 Test Framework

The test framework is divided into two parts: Test Management and Test Execution Platform. The main aspects are described in more detail in the following.

3.1.1 Test execution platform

The Test Execution Platform covers all relevant aspects for testing an ACPS. The ACPS control system interacts with its environment (e.g. driving on a road, which is shared with other traffic participants, etc.). For the interaction, the ACPS control system has to perceive its environment either via sensors or the communication to the infrastructure or both. The system itself is described by its physical dynamics, which again need to be fed back to the environment and so on. The arrows show the basic interactions of these testing architecture blocks. The concrete description of the interface depends on the application domain as well as on the concrete use case.

Depending on the development stage, there will be different instances of the test platform/architecture. For example, in a MiL environment all components will be available as simulation models. Later simulated components will be successively substituted by real physical components resulting in a mixed environment of real-time and non-real-time components.

In a MiL environment, the ACPS control system describes the main system under test (SUT). In later development stages, more aspects are integrated in the SUT (e.g. real sensors).

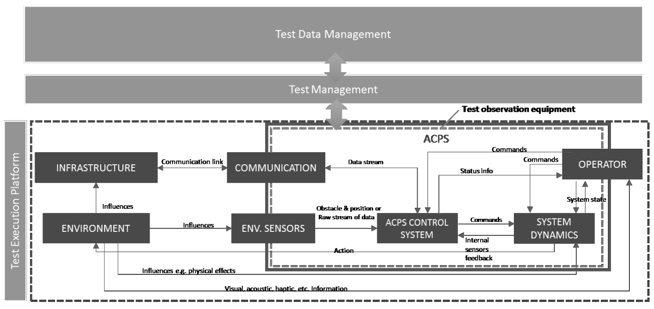


Figure 3: Test Execution Platform

In the following, the single blocks shown in Figure 3 are described in more detail:

ACPS Control System The hardware and/or software that is collectively capable of performing all aspects of the dynamic/automated task (whether part time or full time) of the actual car, train, ship, etc.

System Dynamics The physical model of the SUT that simulates how the actual car, train, or robot physically responds to commands and operator handling.

Operator The human that operates the ACPS (driver, surgeon, etc.). In case of partly automated systems, this mainly includes the hand-off between the operator (e.g. driver) and the system.

Communication Communication between the system under test and other systems (e.g. V2x communication);

Environment Sensors A device that senses physical parameters of the environment.

Infrastructure Includes the facilities that the SUT will communicate with: road-side units, other (autonomous) objects or ACPS, signaling systems. It is the 'active' part of the environment (non-active parts such as road signs are considered as part of the "Environment").

Environment This block describes all aspects around the automated system, which are perceived by the automated system or do have an influence on its behavior. It covers conditions and surroundings intended for the legal operation of the system under test: temperatures, lights, pressures, road friction coefficient, weather conditions, etc.

3.1.2 Test Management

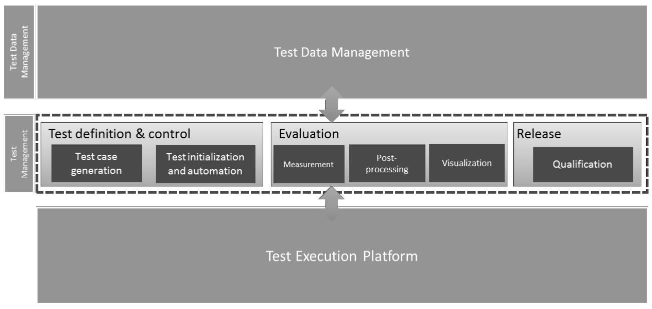


Figure 4: Test Management

The Test Management part is organized in 3 groups: *Test definition and control*, *Evaluation* and *Release*. Figure 4 shows the different aspects of the Test Management part in more detail. The inner blocks are described next.

Test Case generation Generation of a representative set of test cases from scenarios. This means that an interface to a scenario database is required in order to query the required information. This means the identification of relevant scenarios as well as the retrieval of relevant parameters (e.g. weather, type of operator, type of route, equipment, etc.). Depending on the testing purpose this module has to include intelligent methods to select and instantiate the required test cases.

Test initialization & automation Initialization of the test platform and automatic execution of test cases; Controls the order in which tests are performed, in playing out scenarios to the Test Platform.

Measurement Recording of relevant test data; either to be directly processed or to be stored. Post-processing. Definition of certain KPIs and post processing of measured values to calculate the KPIs.

Visualization Visualization of measured data and test results in a way that support the analysis of this data.

Qualification Estimation of the remaining risk and release for operation.

3.2 Test Data Management

This part focuses on all aspects that are valid across different test environments and includes the management of different types of data. In the following, the single blocks are described in more detail.

Test framework instantiation/Variant Management The test framework should be instantiated for a certain test framework or configured for testing a certain product. Variant management can be used in addition to systematically describe the different test framework configurations.

Model management This block describes a model management facility, which is able to store and manage different models (to be reused in different tests). This includes for example information such as which instances of models are used for testing, under what conditions, how long, etc.

Simulation-/Measurement results This block covers a database to store and manage simulation results as well as measurement taken from tests. The amount and type of data usually depends on the purpose of the test.

Real world data base This block summaries databases which are providing real-world data (e.g. real driving data, accident data, operating room data, etc.). It contains a lot of information about what is typically happening in a real world environment. This data can for example be used to retrieve statistics about the frequency and probability of certain scenarios or their criticality.

Scenario generation The ENABLE-S3 project aims for a scenario-based verification and validation approach. A major prerequisite is the existence of a set of scenarios which need to be executed. These scenarios can either be extracted from real-world data as well as generated synthetically. The "scenario generation" block summarizes all activities, methods and tools which are required to generate (virtual) scenarios (e.g. by identifying and transforming critical real-world situations) which can be executed by a simulation engine.

Scenario database This block describes a database which is capable of storing scenarios. In best case, scenarios can be stored in or exported to a standardized format (making them reusable in different tools). Examples for such open formats are OpenDrive and OpenScenario.

SUT Requirements This block covers the System-under-test requirements, which are an essential aspect in the testing process. An essential aspect here is the description of KPIs as an input for the testing process.

Test reports Generation and archiving of test reports.

4 Modularization and standardization

In order to allow the maximum interoperability among the technology bricks that will cover the main aspects of the verification and validation architecture presented above, it is necessary to use of the appropriated standards. This section presents some of the standards that are being used and/or extended in this project.

4.1 Standard management of scenarios

In the context of scenarios we distinguish between two main aspects, the static and the dynamic content. The static content covers everything that does not change frequently such as the road network, traffic signs, buildings, and so on. The dynamic content defines the position and behavior of all the traffic participants involved in such a test run, including the "ego"-vehicle. Generating scenarios can either be done synthetically (i.e. manually based on engineering methods like FMEA) or based on recorded data. This is true for static as well as for dynamic aspects. For dynamic aspects, the former one means to generate test scenarios manually based on safety/security analysis or using existing scenario description e.g. EURO-NCAP test scenarios and reproduce them in a scenario editor. The latter means to mine scenarios from recorded real world data. This approach will at the end lead to a more complete scenario database, since observations of the real world are systematically included in the database.

Not everything can be gathered by recorded data. A lot of existing data sources need to be included as well (e.g. GIS data, map data, pictures, etc.). This means that the various data sources need to be fused in order to get a comprehensive description of the real world. One major problem is the lack of a standardized description of the environment.

In order to promote reusability, open formats and interfaces, such as OpenDrive®[1] and OpenScenario®[2] should be supported.

OpenDrive [4] is an open file format for the logical description of road networks. It is developed and maintained by a team of simulation professionals with large support from the simulation industry. OpenDrive is an already quite established specification for describing the logical view on the environment (i.e. road curvature, lane information, speed limits and directions for single lanes). This specification is supported by various environment simulation tools. Currently, the specification is restricted to automotive applications. Nevertheless, reusing certain aspects and design decisions might be reused in other application domains (e.g. to describe routes for vessels).

OpenScenario is an open file format for the description of dynamic contents in driving simulation applications. The project is in its very early stage and just starts to be supported by environment simulation tools. OpenScenario is targeting the dynamic aspects of the scenario (i.e. traffic participants and their interaction). Again, the specification is currently developed for the automotive domain, but might be adapted for other domains as well.

The main advantage of a standardized scenario description is the reusability of scenarios in various simulation environments. This is especially important since the development of a comprehensive scenario database should be a joint effort by various players. Since each party should still be able to rely on its preferred development and simulation environment a common format is essential. Nevertheless, for sensor models,

[1]http://www.opendrive.org
[2]http://www.openscenario.org

this is still not enough, as materials, surfaces, thicknesses as well as shapes are not standardized and look completely different in every single tool.

The level of detail and the required information for the scenario description typically depends on the purpose of the test. For early stage validation of the trajectory planning, a scenario description with only little level of detail might be fine. For sensor validation or virtual certification, more detailed descriptions are required.

4.2 Co-simulation support

Another important issue is to facilitate the integration of different simulation tools to cooperate in a distributed manner to perform a co-simulation of the ACPS, where each simulation tool focuses in a different part of the simulated scenario, e.g. the environment, sensor behavior, vehicle dynamics, etc.

Currently no standardized interface or protocol specification is available, which allows tools from different vendors to interact with each other during a co-simulation of real-time and non-real-time systems. Therefore, the integration and coupling of heterogeneous systems still require great efforts during the verification process.

To facilitate this integration, this project is promoting the use of the Distributed Co-simulation Protocol [5] (DCP) which is subject to proposal as a standard for real-time and non-real-time system integration and simulation.

The DCP, one of the main results of the ACOSAR project, consists of a data model, a finite state machine, and a communication protocol including a set of protocol data units. It enables the definition, configuration and execution of a wide range of different simulations and test scenarios.

5 Concluding remarks

The ENABLE-S3 consortium, composed by 68 partners from 16 countries of European Union, will present its main results and their applicability to the 13 use cases from 6 different domains in May 2019 in Graz, Austria, during the public event and final review of the project.

References

[1] D. Watzenig and M. Horn (2017), eds., *Automated Driving: Safer and More Efficient Future Driving*, Springer International Publishing.

[2] H. Winner and W. Wachenfeld (2013), *Automatic Driving Protection*, 6, FAS Conference Muenchen, Munich, vol. 9.

[3] W. Wachenfeld and H. Winner (2015), *Die Freigabe des autonomen Fahrens*, in Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte, (M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, eds.), pp. 439–464, Berlin, Heidelberg: Springer Berlin Heidelberg.

[4] M. Dupuis and H. Grezlikowski (2006), *Opendrive®-an open standard for the description of roads in driving simulations*, in Proceedings of the Driving Simulation Conference, pp. 25–36.

[5] M. Krammer, M. Benedikt, T. Blochwitz, K. Alekeish, N. Amringer, C. Kater, S. Materne, R. Ruvalcaba, K. Schuch, J. Zehetner, M. Damm-Norwig, V. Schreiber, N. Nagarajan, I. Corral, T. Sparber, S. Klein, and J. Andert (2018), *The Distributed Co-simulation Protocol for the integration of real-time systems and simulation environments*, in Proceedings of the 50th Computer Simulation Conference, SummerSim '18, (San Diego, CA, USA), pp. 1:1–1:14, Society for Computer Simulation International.

AQUAS: A Project to Bridge the Gaps between Safety and Security Processes

John Favaro, Silvia Mazzini

Intecs SpA, Pisa, Italy; email: {John.Favaro,Silvia.Mazzini}@intecs.it

Peter Popov, Lorenzo Strigini

Centre for Software Reliability, City, University of London, U.K.; email: {ptp,strigini}@csr.city.ac.uk

1 Introduction

We report on an approach to the management of the interplay between the safety and security processes, currently studied in a recently started collaborative European project, AQUAS (Aggregated Quality Assurance for Systems, http://aquas-project.eu/). AQUAS is experimenting with co-ordinating these processes through *"interaction points"*, which will be introduced below, via a set of case studies or "demonstrators". It is motivated by the problems found by industry in combining in a cost-effective way the tasks of ensuring satisfaction of various non-functional requirements (where "ensuring" means "achieving and demonstrating").

Most such problems have been reported with the task of ensuring both safety and security in embedded systems. Companies with established processes for ensuring safety would import processes for ensuring security as well, but problems may arise because on the one hand, the two need to be considered together (e.g. because security violations affect safety, and because design trade-offs may arise between these two sets of goals), but on the other hand, they are the preserves of different technical cultures with their own languages, habitual assumptions in their analyses, etc. It is sometimes said, deprecatingly, that these specialists of different cultures work in "silos", with information flowing vertically within a specialism but not across specialisms. As the SAE J3061 Cybersecurity Guidebook has noted: “A tightly integrated process for Cybersecurity and safety has the advantage of a common resource set, thus, requiring fewer additional resources. However, since both activities require different technical expertise and both activities are resource intensive, it may not be feasible to expect a single team of experts to have the skills to perform both Cybersecurity and safety tasks simultaneously.” It is for this reason that the Guidebook, while recognizing the advantages of the ideal integrated process, makes provisions for non-integrated safety and security processes that communicate in more or less well-defined ways – what in AQUAS we call interaction points (Figure 1).

We call "interaction point" both an activity and the point in a product life cycle (PLC) at which it occurs. The activity is "interaction" in that (a) experts in the various aspects of the system and its properties interact., e.g. security and safety experts; (b) their analyses are combined in some way, that may be anywhere in the range from informal discussion and mutual critique to using mathematical models to assess various measures of interest for alternative design options, or even a single, summary measure to be optimised (e.g., probability of an undesired event); (c) the need for changes or decisions may be recognised that require an integrated view, e.g. because of inevitable trade-offs between desirable properties, and these trade-offs are discussed between the various experts to produce recommendations/decisions, possibly with the aid of the above-mentioned mathematical models.

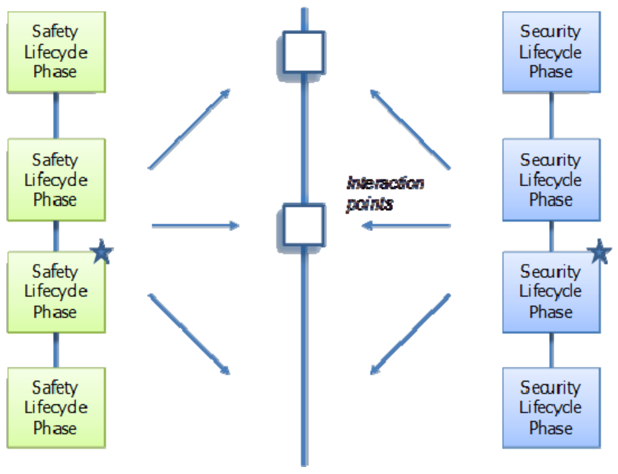


Figure 1 Two separate PLCs with interaction points between them

2 Static versus dynamic interaction points

An important question is when these interactions should take place, to be cost-effective for a given project in a given company. One viewpoint is that the lifecycle model used by the developers should identify from the beginning when interaction points will be needed. These "statically scheduled" interaction points are so scheduled as to achieve a reasonable trade-off between

- The cost of too many interactions for those "lucky" projects that never have conflicts or resulting rework (for these projects, all interactions may be counted in hindsight as unnecessary costs) and

- The cost of too few interactions for the "unlucky" projects, in which conflicts between requirements and unsatisfactory design trade-off are recognised late, requiring expensive rework or causing project failures. For these projects, frequent interaction points would save money by reducing rework.

The standards tend to identify static interaction points, partially through the very nature of the standard as a static text. But the potential improvements through dynamic interaction points are significant. Pre-planned, statically scheduled interaction points are akin to scheduled maintenance of equipment: they happen at predictable times, their cost is factored into the total cost from the beginning, and they are frequent enough to avoid nasty surprises. However, a regime of scheduled maintenance does not necessarily avoid ALL surprises and there is a need to have a design that can deal with failures occurring between maintenance points. If components of a system fail during operation, the system typically needs: means for *failure detection*; means for *diagnosis*; means for *repair* or *reconfiguration, recovery and restart*. In the case of the co-engineered lifecycle, examples of failures and their detection mechanisms might be the following:

- Initial requirements from a client are found to be in conflict during the implementation phase (for instance encryption of data for a particular security standard takes too much time to meet a performance requirement). This may trigger interaction points in the current phase of the PLC, and/or in previous phases (that is, undoing some refinement activity for some system part, going back to change and re-analyse a higher-level design, so as to make the satisfaction of the requirements feasible; or even going back to renegotiate these requirements).
- Inadequate performance may lead to a safety related issue. For example, a machine vision component in an automated system may turn out to be insufficiently robust to adequately recognize a sufficiently large set of risky scenarios and may need to be upgraded for performance. The introduction of new, redundant mechanisms to deliver the needed performance might open up a new attack surface that was previously unanalysed.
- in the process of refining an aspect of design, the design team discovers that they violated some 'contract' established at a previous stage of refinement (e.g., they agreed to implement a certain message encryption as a security control in less than a certain fraction of the main control loop period of a system; but they discover that when implemented it takes longer).
- the safety specialists realise that they may have missed out something important in communicating their proposed architecture to the security team; so, the analysis by the latter that gave the 'all clear' to the architecture may be wrong.
- independently of an on-going development effort (or, alternatively, after deployment), a new vulnerability has been discovered in a component or algorithm. The security team wishes therefore to introduce new controls, which might violate some assumption made by the other teams (e.g. about timing, or about possibility of communication between two components, or authority given to a component) on the basis of the currently specified controls.

In all these cases, the "detection" amounts to some team member becoming aware of something potentially being wrong. Triggering an interaction point (possibly delayed, just as responsive maintenance can be delayed) then serves to perform diagnosis: to decide whether something is indeed wrong, possibly through intermediate steps of more extensive analysis. The interaction point may in turn trigger more extensive analyses (e.g., if our trust that a deadline would not be violated was built simply on extensive statistics of the delays observed in off-line testing, it may trigger another similar round of offline testing), just for the purpose of reaching a diagnosis, and then possibly some rework/redesign, again possibly requiring new analyses on the redesigned system. Analyses of the results of the rework/redesign would be subjected to another interaction point, to check that indeed the problem is resolved. The combination of statically scheduled interaction points and dynamically scheduled ones might prove more cost-effective than a more frequent series of statically scheduled ones.

3 System Design vs. Safety/Security/ Performance Analyses

The evolution of the system through the PLC is captured by models, chosen by the developers. In AQUAS the system models of most of the demonstrators will be based on the OMG SysML/UML formalisms. A significant part of it may be created directly from these models, including by e.g. automatic code generation. Should the system be changed (e.g. fixing faults/vulnerabilities in development or post-deployment), the system model will be modified too, so that the “real system” and the model of it are kept consistent throughout the phases of the PLC.

Assurance about the required non-functional properties of the designed system is achieved by dedicated *methods of analysis* (i.e., Safety, Security, Performance – SSP analysis), focused on assessing whether the system has the required non-functional properties or not. Each one of the various methods used for analysing security, safety and/or performance relies on its *own models*. In some cases, these models coincide with parts of the design documentation: e.g., some verification methods are applied directly to source code or to state machine diagrams used in specifications. But for many SSP analyses, the models they need rely on formalisms that are *very different* from SysML/UML. E.g., performance modelling might use Petri nets or queuing networks. The important point here is that whenever an SSP analysis is needed, a model suitable for it must be extracted or derived from the model of the designed system, available at that particular point in time. Two further important points are worth making here:

- Some methods of analysis (and their respective models) may not be applicable at all before the system model has matured enough (e.g. a tool might need the availability of source code for analysis).
- Some analyses may ignore some details of the designed system even if such details are available. For instance, if one uses a probabilistic state-based model such as Stochastic Petri Nets (SPN) one may be unable to benefit fully from having the full source code of the designed product.
- The design models or design documentation are normally *incomplete* descriptions. For instance, designers may specify the type of a microcontroller or memory chip to use in the system, and so to facilitate verification, appropriate data sheets for these products can be used. But implementation details *inside* these components may have major effects on non-functional properties. E.g., chip mask changes may have undocumented performance implications, or add/remove design faults; the much publicised "Spectre" and "Meltdown" vulnerabilities result from vendor-controlled chip design details that a system designer would typically ignore; and the new security/performance trade-offs required by the fixes for these vulnerabilities were arranged by vendors with limited communication to users. So, analyses for security, safety etc. may require adding extensive "annexes" to system design documentation.

4 Tool Support

Interaction points occur within the context of a number of questions:

- *Why* an interaction point would be needed (e.g. a potential conflict may arise)
- *When* an interaction point should take place (e.g. statically or dynamically determined)
- *What* will take place during the interaction point (e.g. joint examination of a design artefact, trade-off analysis of conflicting design decisions)
- *How* it will take place (e.g. manual observation and discussion, automated tool support, semi-automated tool support)

As challenging as the first two questions are, it is equally challenging to address the second two questions. That is, when an interaction point does occur, there must be a viable set of artefacts (at whatever level of abstraction or lifecycle phase) available.

- *What*. The procedures of e.g. the security and safety analysts can be run independently without difficulty. But they may use different models that are difficult to relate to each other; or, simply, the kind of questions that need to be asked to identity gaps left by the independent analyses are non-obvious. Or e.g. the security analyst may propose a design addition – a subsystem implementing a security control, but specify it in a formalism that makes it hard for the other specialists to analyse. This may create practical difficulties that make a complete analysis too onerous in practice.
- *How*. Even if two artefacts have been created with the same formalism (e.g. SysML), there may be a lack of adequate tools to support the needed analyses (e.g. tools for worst-case performance analysis). More critically, even if the tools are individually available, they may not be able to interact due to poor planning of the overall toolchain framework.

Efficiency of interaction is also an important factor here. People might limit themselves to simpler analyses if it is too time/effort-consuming to do deeper analyses, such as the combined analyses for SSP. Inadequate tool interoperability and inconsistencies of modelling formalisms can severely hamper efficiency, but they can be addressed through emerging interoperability standards. In the end, tool interoperability and judicious automation will improve not only the economics of the work, but also the quality of the result.

Acknowledgment

This project has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737475. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Spain, France, United Kingdom, Austria, Italy, Czech Republic, Germany.

Conclusions

The AQUAS project aims to unlock the traditional approaches and bring co-engineering into mainstream development processes. The project follows a use case driven approach where we are developing the concept and support for interaction points taking place inside and across the product life-cycle. Lessons learned and tool support emerging from the collaborative work in the use cases will be the basis to define the AQUAS methodology.

FED4SAE: A Digital Innovation Hub for the Smart Anything Everywhere Initiative

L. Rioux
Thales Research and Technology, Palaiseau, France
I. Dor
CEA, Grenoble, France

1 Introduction

The introduction of Cyber-Physical Systems (CPS) and Embedded Systems into everything from production facilities and industrial products to everyday products and services provides a wealth of opportunities with long term growth potential. The challenge ahead is for the European industry to seize these opportunities to ensure competitiveness and production capacity.

In this context, many European member states and regions are rolling out various initiatives to promote digital transformation. However, they are neither organized nor financed to act at a Pan-European level due to their local (national/regional) objectives. Cross border interactions are rather limited hampering European digitalization wave across all industries. As stated by Mr. G. Oettinger, European Commissioner: "Digitalisation of industry implies, by nature, cross-border transactions and international presence. No single Member State can resolve the related issues alone, or has the resources to respond to global challenges. We need European champions to win the global game".

The FED4SAE project (www.fed4sae.eu) , whose name is short for Federated CPS Digital Innovation Hubs for the Smart Anything Everywhere (SAE) is in alignment with the European Commission SAE initiative (https:// smartanythingeverywhere.eu/). The ambition of the resulting program is to unleash the creative power of European companies and to support them through their innovation process in various "smart domains" so they can contribute to accelerate the digitization of European industries. FED4SAE permits companies to leverage the resources and the know-how and advanced technologies of some of Europe's top research institutes (research and technology organizations, technology transfer-oriented university institutes) and to provide access to cutting-edge technology platforms offered by leading industrial companies. An additional and new aspect of this program will be to help companies with business modeling and market insights through guidance from conceptual design through market launch. This also includes assistance with access to further funding beyond the initial cascade funding provided by the FED4SAE project.

2 Project Objectives

Due to the complexity of CPS and Embedded Systems, Start-ups, SMEs and Midcaps are very often do not possess all the necessary skills (software, hardware, integration, real time computing…) and technologies (sensors, actuators, communication technologies…) required to successfully develop and bring such systems rapidly to market. The challenges ahead for them are therefore to:

- Identify the role of key CPS technologies in their innovation strategy and understand the added value versus their technology challenges,
- quickly their innovation challenges into technology programs framed in line with market opportunities without necessarily having a strong in-house R&D capacity,
- Select the relevant technologies and their packaging, which implies learning and maturation time,
- Get easy access to state of the art and leading edge technologies with necessary technical support to rapidly exploit these,
- Implement innovation management in their organization to be enable digital market entry especially for new comers,
- Prototype, test, refine and iterate their innovations quickly with end customers to identify and capture the maximum business value,
- Exploit innovative solutions to address their market with innovative business model adapted to the evolution of the industry towards digitization.

The overall ambition of FED4SAE is to boost and sustain the digitization of the European industry in strengthening the European competitiveness in the CPS and Embedded System market by lowering both the technical and business barriers for innovative companies.

Objective 1: To bring innovative CPS technologies to businesses from any sector

The first objective of FED4SAE will be to make available CPS and Embedded System solutions to any European Start-up, SME and Midcap (defined as Third parties) from any sector, understand their innovation project, specify their technology needs and support them to extend their business in developing and exploiting CPS and Embedded System innovative solutions.

Objective 2: To link Third parties to suppliers across value-chains and regions in order to create innovative CPS solutions

The second objective of FED4SAE will be to **initiate** and **boost synergies** between Third parties and established organizations engaged in and connected to the FED4SAE DIHs in order to enable the emergence of innovative CPS and Embedded Systems.

Objective 3: To link Third parties to investors across value-chains and regions in order to accelerate CPS solutions development and industrialization

The third objective of FED4SAE will be to support third parties to reach out to further funding opportunities in order to engage the next step of their development after completing their AE (pre-series, industrialization phase, commercialization…).

Objective 4: To ensure the sustainability of the pan-European DIH network

FED4SAE objective is to ensure both the sustainability of the ecosystem for cross border collaboration but also the access to further cascade funding since this model has already demonstrated its added-value when combined with relevant business case analysis and business driven operation. The sustainability of FED4SAE will combine public and private financing in:

- Encouraging the organization of local hubs with ESIF support in allocating specific resources to FED4SAE pan-European model.

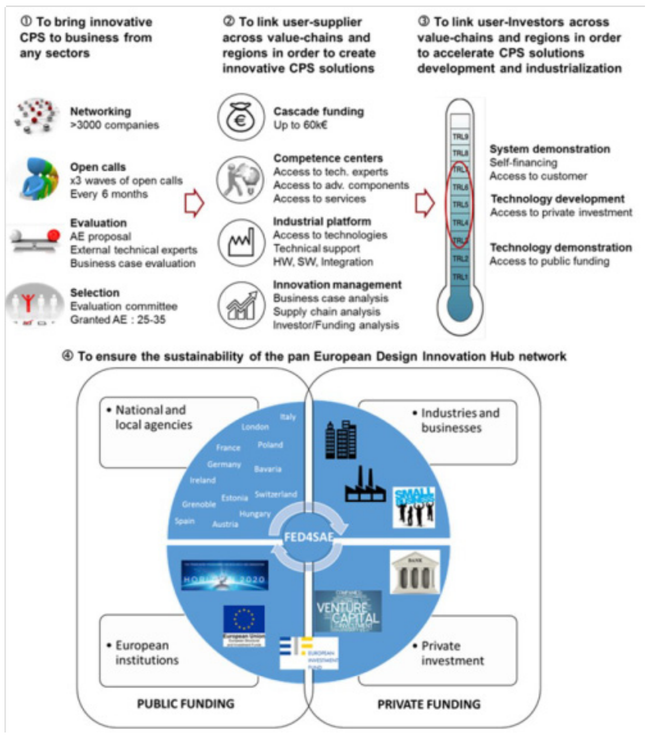


Figure 1 Summary of FED4SAE objectives

- Raising private financing and partnership to support the pan-European DIH network with relevant financial schemes.

3 Overall project concept

FED4SAE concept is dedicated to leverage on the European added-value to accelerate innovation and business growth with a broader adoption of CPS and Embedded Systems innovation while bringing its technical, industrial and business expertise.

In encouraging Third parties to strengthen their product and business differentiation with CPS innovation, FED4SAE will contribute to the digital innovation wave across all of Europe in managing and facilitating the relevant interactions between key innovation stakeholders including regions and local authorities, industries along the entire value chain up to the market as well as private investors. FED4SAE is designed to limit the risk of national silos while focusing on critical mass of actions and communities working all together to adopt, implement and invest in digital innovation.

3.1 Support European companies to lead the CPS based market

The digitization of the European industry is a main challenge and can not only rely on large corporate. Indeed as the European Commission highlights "Small and medium-sized enterprises (SMEs) are the backbone of Europe's economy. They represent 99% of all businesses in the EU. In the past five years, they have created around 85% of new jobs and provided two-thirds of the total private sector employment in the EU." However only a very small part of European SMEs have so far been part of the digital revolution. Only 1.7% of all EU enterprises use advanced digital tools to innovate in products and processes. 41% of European SMEs do not use digital technologies at all according to the European Commission. Many of the smaller businesses don't have the resources to cope with the digital transformation of their business on their own. The main reason is that they are mainly working on incremental innovation, with limited possibility to evolve towards disruptive innovation that drive the creation of new markets and value networks.

3.2 Facilitating innovation adoption in minimizing the valley of death syndrome

One of the main challenges for innovative companies is to face the so called Valley of death6. The valley of death can be defined as the time required to finalize the product development with no or low revenues generation yet. The valley of death is the period when the technical and market risks co-exists. The company has limited resources and has to reach rapidly the market. FED4SAE will support companies and AEs in limiting this valley of death syndrome to accelerate product introduction and in focusing on preparing market penetration. It will be done by providing access to relevant resources including funding for prototyping and production of attractive and innovative solutions. In those early development stages, FED4SAE will start engaging the market introduction with relevant business cases, differentiation and by engaging further investment to support next steps of development up to commercialization.

3.3 Accelerating innovation with proactive actions adapted to start-ups, SMEs and midcaps organization and challenges

A CPS is a system with information processing capabilities that integrates or at least interacts with users, the physical word (through sensors and actuators) and the cyberspace (Server, cloud), as illustrated in Figure 2. Common applications often fall under sensor-based communication-enabled autonomous systems. Other types of CPS include for example autonomous automotive systems, medical monitoring, process control systems, distributed robotics, automatic pilot avionics.

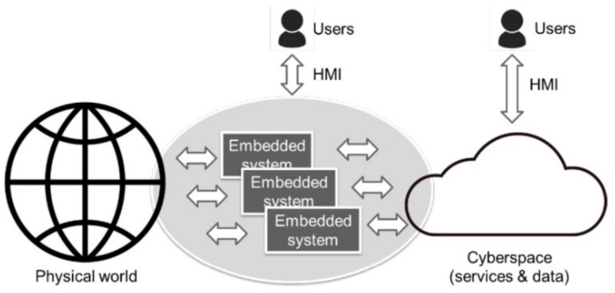


Figure 2 Illustration of ingredients of a CPS

CPS and Embedded Systems are composed from diverse constituent parts (both Software and Hardware) and design practices related to various engineering disciplines making CPS and Embedded System solution development a very complex task.

Today, in a marketplace where rapid innovation is essential, engineers from all disciplines need to be able to explore system designs collaboratively. FED4SAE will act as a value chain aggregator relying on multidisciplinary competencies. FED4SAE DIHs will provide a unique Marketplace organized as a one-stop-shop providing access to technologies, technical expertise, business and financial services to drive successfully innovation projects.

4 Summary

FED4SAE DIHs will provide industrial technology core platforms, research institutes advanced technologies and testbed environments to accelerate Third parties CPS and Embedded System developments.

The industrial technology core platforms are of two kinds: software core platforms and hardware core platforms. Fitting major technological evolutions, the research institutes advanced technologies will bring original concepts and technologies in the CPS challenging domains. Testbed solutions provided by research institutes will use to test the AE prototypes in quasi-real environment of usage. FED4SAE will provide the required competences to enable Third parties to use them and to support their innovative developments. The list of platforms are available on the FED4SAE website (http://www.fed4sae.eu).

Secure Wireless Avionics Intra-Communications: the SCOTT Approach

***Ramiro Samano-Robles*[1] *and José Neves*[2]**

[1]*Research Centre in Real-time and Embedded Computing Systems, Porto, Portugal; email:rasro@isep.ipp.pt*

[2]*GMVIS SKYSOFT SA, Lisbon, Portugal; email:jose.neves@gmv.com*

Abstract

This paper presents the objectives and architecture of the use case of secure wireless avionics intra-communications of the European Project SCOTT (secure connected trustable things). SCOTT aims to build trust of the Internet of Things (IoT) in industrial applications. SCOTT addresses multiple issues such as security, safety, privacy, and dependability across 5 industrial domains: automotive, aeronautics, railway, building and healthcare. The aeronautics use case focuses on the application for active flow control (AFC) based on dense wireless sensor and actuator networks (DWSANs). Topics about security, vulnerabilities and safety in the general field of wireless avionics intra-communications (WAICs) will be addressed. The paper presents preliminary conclusions of the vulnerabilities and security solutions across different entities and layers of the aeronautics IoT architecture.

Keywords: WAICs, security, vulnerability, IoT, Bubble.

1 Introduction

The number of wireless links is growing exponentially. It is estimated that nearly 25 billion devices will be online by 2020 [1]. A high percentage of these devices will use wireless links. Wireless is expanding to areas previously reluctant to this type of communication. In aeronautics, wireless is just recently gaining acceptance for on-board applications. This late adoption is due to reliability and interference issues. Wireless is starting to be used on board for systems that conventionally used only wireline infrastructure (i.e., as replacement of wires). It will also be used for applications which are now only possible thanks to the wireless component (e.g., indoor localization). Recent interference and reliability studies with state-of-the-art wireless standards (see [2]) suggest the feasibility of a relatively new research area called wireless avionics intra-communications (WAICS) [3]. Examples of potential applications of WAICs are: structure health monitoring, fuel tank sensors, automatic route control based on optimized fuel consumption and weather monitoring, automatic turbulence reduction or active flow control, flexible wiring redundancy design, logistics, and in-flight entertainment.

The avionics industry will experience a wireless revolution in the years to come. The concept of "flyby-wireless" [4] opens several issues in design, configuration, security, trustiness, and interference control. Wireless networks are inherently prone to security and privacy threats due to their broadcast nature. Eavesdropping by unintended parties on board or outside the airplane is one of the main issues, which requires appropriate encryption, coding and/or authentication schemes to be minimized. Man-in-the-middle (MiM) and denial of service (DoS) attacks can prevent sensor information about aircraft health from reaching the control cabin, thus posing a threat to the safety of the plane, leading to mal-functioning. Intentional and unintentional jamming can also increase the risk of failure and lack of communication in aircraft. All these vulnerabilities and risks need to be properly studied, so that potential countermeasures can be implemented.

This paper deals with security in the domain of aeronautics of the European ECSEL project SCOTT (secure Connected Trustable Things) [5]. The aeronautics use case exploits the application of active flow control (AFC) using dense wireless sensor and actuator networks (DWSANs) to design secure communications across different layers and entities of the architecture. The objective is to increase the technology readiness level (TRL) of secure wireless solutions in the avionics industry.

SCOTT is a project that aims to boost trust, security, safety, privacy and dependability of the Internet of things (IoT) in industrial applications. SCOTT envisions a trusted, industrial-compliant cloud connectivity for IoT, with high energy efficiency and autonomous operation. SCOTT uses the concept of Bubble from the predecessor project DEWI [6]. The Bubble is a high-level abstraction of an industrial WSAN with enhanced interoperability, dependability, standardized access to sensor readings, and cross-domain development [7]. SCOTT foresees an ecosystem of communicating bubbles in different industrial use cases.

This paper is organized as follows. Section 2 presents the objectives of the aeronautics domain of the project. Section 3 presents the advances with respect to the state of the art. Section 4 presents the application of active flow control and its architecture. Section 5 presents the physical entity model. Section 6 deals with the functionality model. Section 7 presents preliminary vulnerability and security analysis. Section 8 presents the conclusions of the paper.

2 Objectives and measurable indicators

The objectives of the aeronautics domain (Figure1) are [5]:

- Ensure that WAICs are secure, trustable and safe (reduce identified vulnerabilities and security threats in the project of wireless solutions by up to 90%).
- Construct gateways between WAICs and the internal networks of commercial aircraft enforcing multi-level and multi-metric security, privacy and safety.
- Increase fuel efficiency by replacing cables and using dense-WSANs for turbulence and skin drag control.
- Conduct a study of vulnerabilities and potential attacks to the new hybrid wireless/wired avionics infrastructure. Propose countermeasures with a trade-off analysis between complexity and risk.
- Provide guidelines to stakeholders on how to solve common problems of security, privacy, and trustiness.
- Help in the adoption of WAICs in industry (including standardization and certification issues).
- Enable the use of semantics interoperable middleware tools for the development of advanced fleet management and smart avionics applications.

The objectives in terms of measurable indicators are:

- To create a repository of tools, reference implementations and links to middleware and reliability studies of avionics infrastructure.
- Demonstrate secure wireless avionics applications covering different scenarios.
- Development of gateways for avionics applications providing secure and trustable protocol translation.
- Improve the performance of wireless avionics by a factor of 10 in terms of spectral efficiency, also to improving energy efficiency and interference reduction.
- Demonstrate via a prototype, standardization and a reference implementation the reliability and trustiness of commercial wireless standards on board aircraft.
- Provide guidelines to aerospace stakeholders on how to improve privacy and security in WAICs.

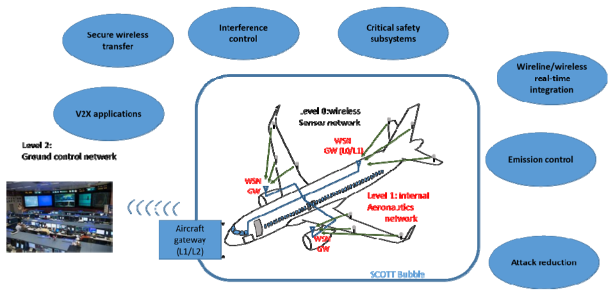

Secure wireless transfer
Interference control
Critical safety subsystems
Wireline/wireless real-time integration
V2X applications
Level 2: Ground control network
.evel 0:wireless Sensor network
WSN GW (L0/L1)
WSN GW
Level 1: internal Aerona.tics network
WSN GW
Aircraft gateway (L1/L2)
SCOTT Bubble
Emission control
Attack reduction

Figure 1 Aeronautics objectives

3 State of the art (SoA) and progress

One major potential advantage of using wireless technology in aeronautics is the reduction of wiring, which is a critical issue in aircraft and spacecraft design [8]. Blackhawk helicopters carry almost 2,000 pounds of wires for computers and sensors [9]. Electrical wiring problems cause on average two inflight fires every month as well as more than 1077 mission aborts and over a hundred thousand lost mission hours per year [10]. Each year, navy spends one to two million man-hours finding and fixing wiring problems [11] . Damages on a wired connection can affect not only the system related to the faulty wire, but also contiguous systems which individually would have been fully operational. Therefore, the use of wireless technology is expected to bring considerable gains to the avionics industry in terms of reduction of cables, more flexibility in the design of redundancy links, and faster troubleshooting. Wireless nodes have also the advantage reaching places of an aircraft that cannot be reached by wires. Furthermore, modern WSNs provide self-configuration, RF tolerance, and maintenance troubleshooting that are much more flexible than their wireline counterparts. In critical avionics applications though, wireless links cannot completely replace wired links due to the high reliability requirement. However, they can replace redundant links, thus increasing reliability and flexibility in the design.

In avionics, wireless technology is well known for several applications such as: air traffic management (ATM), telemetry, aircraft-ground control, satellite localization/ communication, identification of friend-or-foe systems, inter-aircraft communications, and radar. In contrast to these applications, which are relatively mature, WAICs have just recently gained attention. Recent results suggest that existing standardized commercial wireless technologies show potential low levels of interference and thus low impact to on-board systems, as well as reliable performance compatible with existing wireline infrastructure. These results have paved the way for new applications for wireless communications in aircrafts.

Security is an important issue in wireless avionics. In comparison with conventional WSNs, the data of an aircraft, particularly related to aircraft health monitoring, is vital for the good functioning, management and safety of a plane. Therefore, the sensor network should be more robust to different types of attacks either from passengers or entities on board, ground or even from other aircrafts. An extensive analysis of different types of security attacks using an adversary model, where the adversary can be internal or external and the attack can be passive or active are available in the literature. Safety and business threats have been identified such as: data integrity, authenticity, confidentiality, link-key establishment, channel jamming mitigation, secure routing, secure location verification, and robustness to node capture (eavesdropping)[12].

SCOTT intends to leverage wireless technology in the aeronautical industry. This means to effectively implement secure and safe wireless technology in real applications to be used by the aeronautics industry. The objective is to bring the concept of IoT to aeronautical applications thus creating a smart, flexible and automatic environment on board and in different elements of the aeronautics industry, including airports, management of infrastructure, flight control, vehicle-to-infrastructure and/or vehicle-to-vehicle communication, turbulence reduction, etc. The aeronautics domain will present a full analysis of vulnerabilities and potential countermeasures for the hybrid aircraft wireless/wireline infrastructure. SCOTT attempts to create a framework for smart avionics development with different levels of security and trustiness that will enable big data analytics and cloud computation for the optimization of aircraft performance, reduction of fuel consumption, controlled interference, and high spectral efficiency.

Several issues will be addressed, including propagation modelling for reliable transmission and reduced leaking or interference, as well as MAC-PHY cross-layer design to reduce conflicts between different subnetworks in the same aircraft and minimize interference to control subsystems. Secure links will be addressed by minimizing transmissions to potential eavesdroppers or unsafe locations either within the same or in other airplanes. Privacy of data will be also addressed by convenient mechanisms and data-context management with ground control.

The aeronautics industry expects huge benefits from the use of wireless technologies. It is estimated that cables constitute over 70% of aircraft weight. The use of wireless links could reduce this figure down to 55%. In addition, technologies such as AFC enabled by DWSANs can help reduce the effect of skin drag, thus further improving fuel consumption efficiency. A reduction of 10% in fuel consumption is translated into several millions of dollars in savings. It is estimated that the use of wireless technologies will bring a 12% reduction in terms of fuel consumption [13]. Further improvements are possible when combined with other technologies such as winglets, carbon fibre fuselage and improved turbine design. The use of cables has one more benefit in terms of cabling planning tasks. It is estimated that these planning tasks have a cost of 2,200 dollars per kg of aircraft [14]. When considering two types of aircraft the estimated savings are the following [15]: A320/B737-900 6,400 kg x 2,200 \$/kg ≈ \$14 million, and A350-900/B787-9 23,000 kg x 2,200\$/kg ≈ \$50,6 million. It is also estimated that 13% of an aircraft operation cost is related to maintenance, reparation and overhaul. Wireless technologies are expected to have a big impact in the reduction of these costs. Automatic configuration, maintenance and troubleshooting can be performed over the air reducing maintenance service costs.

4 Application case: Active Flow Control based on dense WSANs

The objective of the Bubble AFC is to employ a wireless sensor-actuator and communication bubble for suppression of the turbulent flow and delaying the BL (boundary layer) transition. The sensor network will detect the low-pressure region on the upper wing surface. The position of BL transition zone will be defined, selecting the appropriate actuators to be activated. At the same time, and based on the sensor values, the set of conditions for operation of the actuators (e.g., frequency, amplitude) will be calculated based on existing data (pre-set data). The selected actuators are activated to manage the turbulent flow on the wing surface. The data is stored. A new sensor reading is collected, and the cycle is repeated. The stored data can be analysed to assess system operation during, for example, different flight profiles or moments (e.g., take-off, landing, and cruise). Ground systems can interact with the sensor-actuator and communication bubble to get the data recorded during the flight and process this information to determine actuation plans and analyse the data of the whole fleet.

There are several challenges in the interconnectivity and how to achieve the desired objective in a dependable manner, whilst minimizing energy expenditure. The WSAN requires sensor measurements at high frequency and in a synchronous manner, to be able to correlate sensor readings, especially from sensors in close proximity. The WSAN also needs deal with failures of sensors, and this can be approached by employing reliable data transmission and data delivery mechanisms and also by employing data processing strategies that can deal with sensor failures.

It is important to boost the use of wireless communication systems on board to enable the deployment, as soon as possible, of technologies like Structural Health Monitoring (SHM) and Active Flow Control. To achieve this goal, these wireless networks and sensor systems need to communicate and interact with the main data buses of the aircraft. Hence, the specification of bi-directional bridges between different types of technologies is required. This is still the case if wireless technologies are used as the main data bus of the aircraft. Different wireless networks, with different delivery deadlines and different underlying technologies must operate together without possibility of interference. Bridge protocols and interfaces must be specified considering the constraints of the different networks.

The AFC system uses an architecture with a set of polygonal patches, each patch with a regular grid/array of sensors and actuators. These patches will be located mainly on surface of the wings of the aircraft, and potentially on other surfaces of the fuselage. The objective is to control the turbulence region across the aircraft and reduce losses. All the sensors and actuators inside a single patch will be wired together sharing a single communication and control point. The patches will communicate wirelessly with a relay or access point located conveniently in the aircraft to ensure good communication with several patches. Each patch will be enabled with some sort of intelligence to provide management of all the sensors and actuators inside the patch and to provide convenient communication link with the sink and the control unit inside the Bubble. The architecture of AFC is therefore a hybrid of a wireless and wireline sensor network, which is the most convenient for this application. The information generated by each sensor will be collected by the control unit of each patch (node) which will provide some preliminary filtering, fusion and aggregation functionalities. The refined information will be then relayed towards the control unit (Gateway or relay node). Based on this collected information and based on different flight profiles, the AFC system will decide the type of actions to be performed by the set of actuators on each patch. Each of the flow control actuators is a piezoelectric device (synthetic jet actuator –SJA- or Fliperon). These actuators can delay the turbulence BL and thus help in counteracting the dragging effect in response to the measured information and according to flight profiles.

The size and number of patches, as well as the number of sensors/actuators per patch is optimized using a simulator. These parameters are function of the accuracy of the active flow control system, the range of the wireless technology selected, and the data rate of the wireless sensor nodes. All sensor/actuators nodes will be powered via cables. The patch will be provided with some power saving features too. For example, when a sensor information or actuation is not required from some patches, they can be powered down until they need to be used again, thereby saving energy.

The architecture proposed for the AFC system is relatively new in aeronautics, as it constitutes a hybrid design with wired and wireless components. The number of sensors for this application is expected to be large, more than in common WSNs, being deployed over a relatively small area. This brings up the issue of interference, if each sensor was to be enabled with an individual wireless connection. To solve this, our approach presents an architecture where groups of sensors wired together form a patch that will act as a single wireless transmitter. Each patch will be provided with smart self-configuration and control. Figure 2 shows the possible embodiment of a regular design of sensor and actuators inside a patch. Each patch will have a radio transceiver and a control unit with some intelligence. This node will be in charge of organizing the processing and operations inside the patch, as well as filtering, fusing, and aggregating data to be sent towards the wireless node.

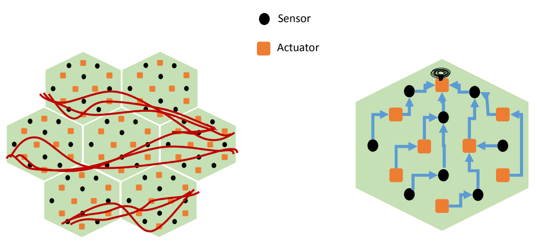


Figure 2 Patch concept for AFC

Another aspect is the interconnection of WAICs into the avionics internal systems as shown in Figure 3. The proposed solution has to be able to pass reliably the traffic from/to the wireless sensor/actuator network to the internal avionics network under different QoS constraints. In general, the AFDX (Avionics Full-Duplex Switched Ethernet) network (or ARINC664) has more stringent QoS requirements, therefore the solution must include an appropriate scheduler that will ensure these QoS constraints of the AFDX traffic are met or conveniently addressed when transported to/from the wireless domain.

4.1 Overview of the architecture

The main physical entity of the SCOTT AFC system is a regular array of wired sensors and actuators also called patch. A possible configuration of this patch and an array of patches are shown in Figure 2. The patch can have hexagonal, rectangular or in general a polygonal shape, depending on the needs of coverage over the aircraft. The patch is mounted over the surface of the fuselage and mainly the wings of the aircraft, where turbulent flow is expected to be formed, particularly at high vehicle speeds and high values of angle of attack (AoA). We recall here that the objective of the dense SAN (sensor and actuator network) implemented by means of patches is to track the formation of turbulent flow and attempt to delay the separation of the boundary layer using actuation policies for different flight profiles or moments of an aircraft mission. All the sensors and actuators inside the patch are controlled by a master unit, which is in charge of intra-patch management, signal relaying, data aggregation, data fusion, compression, and protocol conversion. The sensors and actuators can be connected using a real time technology that can have several characteristics or topologies. One potential configuration is using a microprocessor board controlling one subset of sensors and actuators inside the patch. A network of microprocessors is deployed inside the patch, with a real time transmission technology such as CAN (Controller Area Network) or ARINC 664. Intra-patch routing algorithms can be implemented to allow the information of different sensors to be collected reliably and in real time by the master unit.

Each patch in the network has a wireless transmission unit that is used to communicate with a wireless gateway

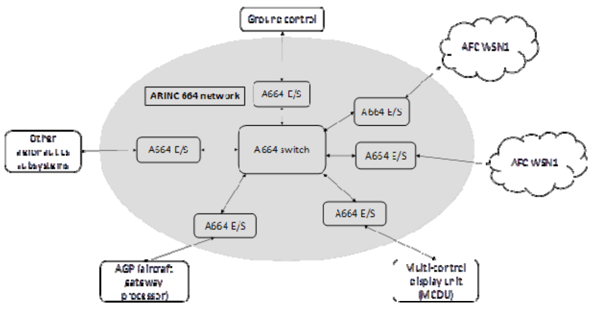


Figure 3 Interconnection with an aeronautical internal network (AFDX)

located conveniently in the aircraft to maximize coverage with a set of distributed patches (see high level architecture displayed in Figure 5). The patch is the basic unit of the proposed AFC system, as it provides modularity, scalability, flexible implementation, as well as advanced management and troubleshooting. Close loop operation occurs at three levels:

1. Directly at the sensor and actuator microprocessor control level to deal with the fast (short term) and spatially correlated variations of the turbulent flow to be sensed.
2. At the level of the internal aeronautics network (see Figure 3). A control unit for the network of patches resides in the internal control operation of the aircraft. The decisions of the medium-term turbulence statistics are taken directly in this close loop control unit on-board the aircraft.
3. All the relevant measurements for different moments of an aircraft mission are relayed from the aircraft to ground control. Ground control contains a database of actuation policies that are optimized over different types of aircraft at different times of the year, routes and weather conditions. This level of control allows operators to optimize routes, as well as actuation policies based on big data analytics that will become more reliable over longer periods of time and with more data of sensor and actuation policies.

5 Physical entity model

Patch of sensors and actuators. The basic unit of the AFC system consists of a regular set of sensors and actuators that communicate with each other in a mesh array or in star formation with a master control unit. The intra-patch communication technology can be real-time or with high reliability to transport all the sensor readings to the master unit, as well as any actuation control policy back from the master unit to the actuators. Each patch has a wireless communication module that allows transmission with an access point or with other patches depending on the configuration. Patches are also allowed to relay the information of other patches towards the destination if necessary. The control unit can also process the sensor data across time and space inside the patch. Other functionalities of the patch include filtering, encoding, encryption, compression, etc. One potential configuration is using a microprocessor board controlling one subset of sensors and actuators inside the patch. A network of microprocessors is deployed inside the patch, with a real time transmission technology such as CAN (Controller Area Network) or ARINC 664.The intra patch communication technology can use secure routing to avoid malfunction or an attack.

Wireless gateway or WAICs access point. This entity implements the PHY and MAC layer transmission and organisation of the WAICS radio technology for communication with patches. The gateway translates the wireless protocol to the internal wireline aeronautics network of commercial aircraft. This translation has several challenges due to the different nature of the unreliable and unsecure wireless world in comparison with the real-time internal avionics network. Part of the analysis is how to make secure this translation from the wireless domain to the wireline real-time operation of the commercial aircraft.

Internal actuation policy control unit. This entity is in charge of the collection of the medium-term statistics of the collected flow information from the network of patches across the entire aircraft. Therefore, it can be used to calculate actuation policies that optimize the delaying of the BL separation for the whole airplane. In this problem it is evident that the whole performance and stability of the aircraft as well as aerodynamic efficiency, and monitoring of other stability issues of the airplane come into place. In addition, for security purposes it is possible to implement intrusion detection, misbehaviour tracking, redundancy coding, authentication of patches, authorisation of actuation policy control, etc.

Ground operator and actuation policy database back end servers. This entity is in charge of the actuation control and optimisation across different aircraft. It is intended to provide airline operators with a means to control, analyse, collect and process sensor data of different routes and aircraft. This processing aims to obtain (using cloud computing tools, for example) optimised actuation policies according to the time of the year, route, type of aircraft, weather conditions, etc. In a generalized scenario, this entity provides consolidated access to sensor and actuation control information for wireless avionics applications. Several security mechanisms can be used in this external access to aircraft information such as authorization, authentication, encryption, tunnelling, intrusion detection, privacy labelling/control, etc.

5.1 Aircraft architecture

The aircraft comprises several systems with different functions defined to achieve several product goals (see Figure 4):

1. Aircraft Control Domain (ACD): contains functions required to maintain the aircraft airworthy providing control to pilot or breathable environment to passengers. Any fail or malfunction jeopardizes the aircraft.
2. Airline Information Services Domain (AISD): used by airline to operate the aircraft providing maintenance information and software and databases updates.
3. Passenger Information and Entertainment Services Domain (PIESD): contains those functions used by passengers during the flight like games, internet connection and access to media.

5.2 Layered model alignment

This section provides the alignment of the physical entity model described in previous subsection with the layered overview of the SCOTT high level architecture. This layered model is closely correlated to the concept of

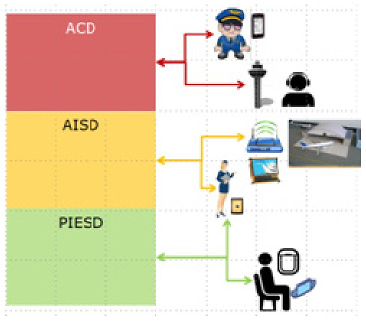


Figure 4 Aircraft domains and users

SCOTT Bubble, which is the basic building block for interoperability and security enhancing for the project. This layered model consists of three levels (one of them optional) that define the intra and extra-bubble space as observed in Figure 5. Level 0 is the wireless domain to provide the last link between the fixed aeronautical infrastructure towards the distributed sensor or object nodes. In the active flow control use case, this wireless technology has actually a hybrid approach using wireline and wireless components under the name of patch. A patch is a wireline entity of sensors and actuators. Each patch uses wireless technology to communicate with the L0 or WSN gateway. The access point is placed on-board the aircraft therefore acting as the translation entity between the wireless domain and the internal network of the aircraft. This internal network of the aircraft acts as the L1 of the SCOTT reference architecture. Many other WAICs applications will use the same approach, particularly those in which the wireless link replaces an existing wireline sensor. In the case of the AFC use case, it is also plausible that L1 is completely independent of the internal network of the aircraft. However, for the sake of covering more generic implementations, it will be multiplexed inside this internal network, which in many current commercial aircraft is a real-time deterministic version of Ethernet technology. This integration into the L1 internal aircraft network, comes at the expense of traffic contention, possible attacks from other points inside the internal network, as well as attacks originated in the wireless network towards other aircraft internal subsystems. This means that the internal critical aircraft network can be subject of an attack coming from the wireless domain, which is a less secure environment. In the SCOTT reference architecture, L1 is an optional level, mainly because in some uses cases it is possible that this interaction with an existing domain network does not exist. The on-board unit acts as the Bubble Gateway, which controls all aspects of the intra-bubble space and provides translation for external user access. This is the boundary of the SCOTT Bubble in aeronautics.

Finally, Level 2 of the reference architecture defines the extra bubble space. L2 is used for external access to the information of Nodes inside the aeronautical Bubble. This is the ground control operation network, where the external user is the airline operator or a smart avionics application collecting information from many different aeronautical Bubbles, inside the same aircraft or located in different aircraft or fleets. The mapping of the aeronautical use case to this layered view of the architecture is shown in Figure 5.

The Bubble is a concept that allows an integration of legacy WSN and local industrial domain technologies into a single point of entry towards the modern Internet cloud. The bubble Gateway can provide transparent access to the objects inside the Bubble, or simply to a summarized version of the information generated inside the Bubble. This concepts allows designers to exercise control over the access to the intra-bubble entities, and therefore enforce higher dependability different from the non-delay sensitive internet-like infrastructure (L2) and also higher security control. In the aeronautics industry, the use of a Bubble confined to one aircraft or sections of the aircraft is a powerful tool to avoid attacks from external entities, while also controlling the permissions granted to L1 internal users. The attacks coming from the passenger entertainment system can also be handled by enabling the bubble gateway with convenient scheduling policies and out-of-band security communication, as well as autonomous operation.

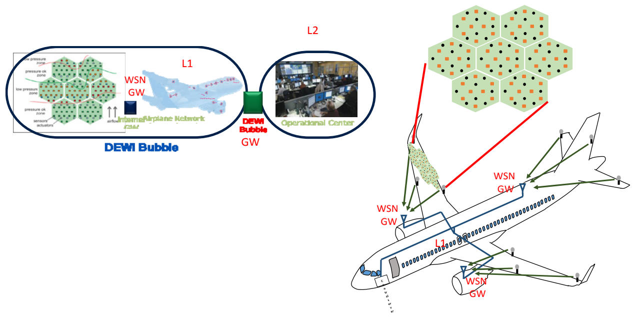


Figure 5: AFC use case architecture

6 Functionality model

The functionality model is derived explicitly from the reference architecture of the project. The explicit functional model for the AFC system is shown in Figure 6, and the hybrid view functional versus layered entity model is shown in Figure 7. Functional layers include: Device Layer (DL), Network Layer (NL), Service Layer (SL), IoT and Virtualization Layer (IOTL), Cloud and application Layer (CAL), and Service Layer Management (SLM) and Cross-Layer Management (CLM).

Each of the physical entities will implement a slight variation of the functional model. The hardware layer in the patch unit focuses on the technology to interconnect sensors and actuators, intra-patch routing, management, compression, redundancy coding, encryption (optionally), authentication, intrusion detection, safe mode operation and troubleshooting. The intra-patch technology is real-time and is used to collect the sensor measurements from the dense mesh of nodes in the master unit of each patch. There are no high-level functionalities here except in the master unit of each patch, which provides protocol translation to the wireless domain.

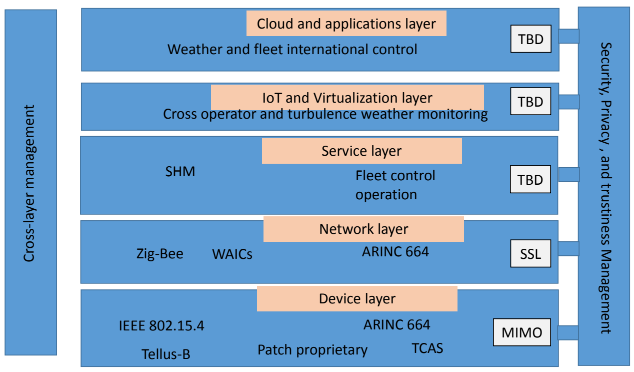
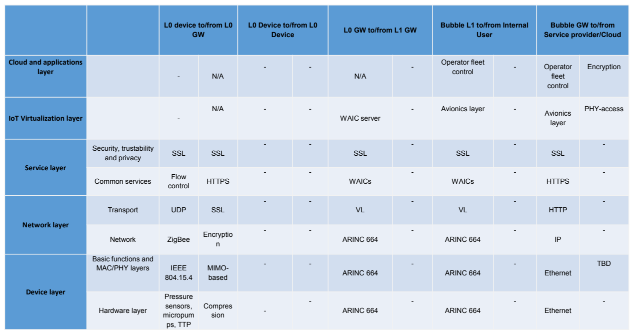

Cross-layer management
Cloud and applications layer
Weather and fleet international control
TBD
IoT and Virtualization layer
Cross operator and turbulence weather monitoring
TBD
Service layer
SHM
Fleet control operation
TBD
Network layer
Zig-Bee
WAICs
ARINC 664
SSL
Device layer
IEEE 802.15.4
ARINC 664
MIMO
Tellus-B
Patch proprietary
TCAS
Security, Privacy , and trustiness Management

Figure 6 Functional entity layered model



		L0 device to/from L0 GW		L0 Device to/from L0 Device		L0 GW to/from L1 GW		Bubble L1 to/from Internal User		Bubble GW to/from Service provider/Cloud	
Cloud and applications layer		-	N/A	-	-	N/A	-	Operator fleet control	-	Operator fleet control	Encryption
IoT Virtualization layer		-	N/A	-	-	WAIC server	-	Avionics layer	-	Avionics layer	PHY-access
Service layer	Security, trustability and privacy	SSL	SSL	-	-	SSL	-	SSL	-	SSL	-
	Common services	Flow control	HTTPS	-	-	WAICs	-	WAICs	-	HTTPS	-
Network layer	Transport	UDP	SSL	-	-	VL	-	VL	-	HTTP	-
	Network	ZigBee	Encryptio n	-	-	ARINC 664	-	ARINC 664	-	IP	-
Device layer	Basic functions and MAC/PHY layers	IEEE 804.15.4	MIMO-based	-	-	ARINC 664	-	ARINC 664	-	Ethernet	TBD
	Hardware layer	Pressure sensors, micropum ps, TTP	Compres sion	-	-	ARINC 664	-	ARINC 664	-	Ethernet	-

Figure 7 Mapping functional vs physical entity

One candidate for intra-patch communication is the protocol TTP (Time Triggered protocol). In the wireless domain for inter-patch communication, several aspects of the functional model are here presented: MAC and PHY communication layers use MIMO (multiple-input multiple output), beamforming, MAC-PHY security, interference rejection, spatial-based authentication, collision resolution by retransmission diversity, multi-packet reception, interference alignment, and dependability control. Optionally, encryption in this link will also be used based on the IEEE 802.15.4 standard. Intrusion detection, and safety hazards identification are also being investigated. Higher layer functionalities include secure routing, tunnelling, patch-authentication using key distribution algorithms, malware detection, and firewall protection to avoid intrusion into the internal network of the aircraft. The inter patch network is focused heavily on secure radio resource management using multidimensional physical and MAC layer diversity (retransmission control), as well as MIMO allocation. Other functionalities in this inter-patch network are troubleshooting, energy management, flow state estimation, and actuation control.

The functionalities in L1 are mainly focused on the scheduling of traffic of the AFC system into the internal commercial real-time network of the aircraft (using the standard ARINC 664). Other functionalities include the following: quality of service control, flow management, secure encryption, traffic analysis to avoid malware intrusion, etc. The Bubble gateway has upper layer functionalities of routing in the internet, sensor data fusion, actuation control/update, sensor node virtualization, tunnelling, authentication of external users, key distribution, intra AFC system management, traffic control, dependability insurance mechanisms for real time internal networks, device management, etc. Secure Socket Layer (SSL) is one of the options in evaluation to be implemented at the L1 and L2 network levels of the aeronautics architecture. An extension of the concept of virtual link (VL) used in the standard ARINC 664 is also under consideration to be used in the wireless domain.

Other associated functionalities to the AFC use case are aircraft collision avoidance using the technology TCAS (Traffic collision avoidance). This refers to the high-level application domain of secure wireless avionics intra communications. The model can also be extended to other structure health monitoring (SHM)-like applications for the aircraft. More details are shown in Figure 7, where some of the interfaces are still under study (TBD- to be defined). The functional view of the reference architecture defines several interfaces between layers as follows:

6.1 Interface DL-NL

The network layer requests the services from the device layer implemented in the patches and the MAC-PHY technology used for the inter-patch communication. The NL is in charge of routing in the network of patches, IP address identification, interoperability with the internal network of the airplane via scheduling, and traffic control. The network layer has also the objective to have a load balance in all the possible AFC networks across the plane, and the matching between the deadlines of the wireline and wireless network. This interface can also host some security functionalities based on IP technology such as IPSEC, tunnelling, secure sockets layer, etc.

6.2 Interface NL-SL

The service layer requests the network layer with the flows of the different patches and wireless networks aggregates of the active flow control system. It is in charge of organizing all the collection of sensor information across the different wireless networks of patches, processing and correcting errors. Intrusion identification is possible by matching the statistics of different networks and comparing to established margins of values. There is also the possibility to detected interference and jammers. Error of the boundary layer tracking or estimation of lift off forces can be used as metrics to estimate malfunction or potential attacks.

6.3 Interface SL-IOTL

The IoT layer allows airliner operator to gather data from aircraft. Authentication of credential of operators, as well as rules for privacy management for integrity or exposure can be implemented in this interface mechanism.

6.4 Interface IOTL-CAL

This interface aims to provide the data of one aircraft to the cloud computing facilities that will calculate optimum actuation policies using the data from different aircraft, airliners and potentially different routes. This will allow us to provide one last level of closed loop control.

6.5 Interface DL-CLM

The main mechanisms for security control in the AFC system are foreseen to be implemented in the MAC-PHY layer. The cross-layer management aims to use this information to improve system performance in different layers. Channel conditions can be used indirectly to estimate flow states and provide redundancy to the sensor information. They can also be used to authenticate, manage and troubleshoot different patches.

6.6 Interface DL-SLM

This interface focuses on the multi-layer security interface with the device layer. Examples of this interface allow MAC-PHY algorithms to identify jammers or directions of eavesdroppers. Node identification using direction of arrival or statistical signal processing are also possible. Redundancy of source and channel coding can be used.

6.7 Interface NL-CLM

In this interface the network layer provides information to cross-layer optimization algorithms, Routes, Addresses, traffic state, quality of service, etc. are some of the metrics and information that can be requested through this interface.

6.8 Interface NL-SLM

The network layer interacts with the security layer management via a set of specific protocols. Tunnelling, virtual links, security layers, etc. are examples of specific implementations of this interface. In the aeronautics use case there is no expected usage of this interface.

7 Vulnerability and attack model(s)

Vulnerability and attack models are being developed for different layers of the aeronautics architecture. A useful reference model used in the SCOTT reference architecture and across the literature of security of IT systems (Common Criteria) is displayed in Figure 8. The important aspect from this framework is to identify the main asset, the associated vulnerability, and potential threats(s). From this information it is possible to define the actions that the stakeholders are willing to implement to reduce risk. The following tables show the vulnerabilities identified so far and potential solutions.

, Table 2, and Table 3 present the vulnerabilities and potential solutions for L0, L1, and L2 layers, respectively. The tables follow the functional model of the SCOTT reference Architecture.

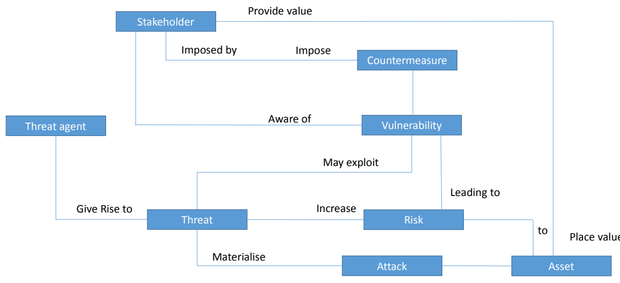


Figure 8 The Common criteria conceptual model for security

Table 1: Vulnerabilities, threats and solutions AFC L0

Layer	Vulnerabilities	Potential solutions
CAL	N/A	N/A
IOTL	N/A	N/A
SL	DDoS	Packet analysis, authentication
NL	DoS, spoofing, MiM	Authentication, encryption,
DL	Jamming, eavesdropping, collision, Integrity.	MIMO, beamforming, blind processing, rotational invariance techniques, multi-objective optimization

Table 2: Vulnerabilities, threats and solutions AFC L1

Layer	Vulnerabilities	Potential solutions
CAL	Spoofing, Identity theft	
IOTL	DoS, latency issues	
SL	Replay attack	
NL	DoS, spoofing, MIM	Authentication, encryption,
DL	Interference, congestion, spoofing	MIMO, scheduling, traffic shaping, authentication, PHY-layer assisted control and sensor aggregation

Table 3: Vulnerabilities, threats and solutions AFC L2

layer	Vulnerabilities	Potential solutions
CAL	Data integrity, lack of privacy, lack of confidentiality, Spoofing, Identity theft	
IOTL	DoS, latency issues	
SL	Replay attack	Firewall L3, tunnelling, Key distribution
NL	DoS, MiM	Authentication, encryption,
DL	Spoofing	PHY-layer assisted control and sensor aggregation, authentication

Figure 9 shows the loop of actuation control and the potential security issues that can be found along that loop and the entities involved in the process of the aeronautics use case. The intra-patch technology can be subject to software and hardware malfunctions, hacking attacks that take over the control of some patches operating system or transmission units. Some software verification, safe-mode operation, or firewalls can be used to avoid these problems inside the patch. The patches aim to reliably collect information of the state of the flow, and also implement the optimum actuation policy with the lowest delay to reduce risks of incorrect operations, or instability of the aircraft. It has been identified in previous deliverables that attacks such as denial of service or jamming that can completely disable the AFC system are not the most serious types of threats, provided the system is identified as unavailable. The most serious threats in the AFC case is when the information collected by the patch has been mismanaged or that its integrity is lost due to man in the middle, spoofing or replay attacks. This means that the control logic of the AFC system will decide actuation policies that are incorrect and therefore will affect the efficiency of the system in terms of loss of lift off forces, reduced efficiency in skin drag reduction, and eventually in fuel consumption increase, reduced range, payload capacity or aircraft speed. Therefore, particular attention will be placed on attacks where the data integrity of the sensors or the loop to disseminate actuation policies is compromised. In the network of patches, MIMO (multiple input multiple output) will be used to construct an efficient way to manage the wireless transmissions to reduce risks of eavesdropping attacks, counteract jamming, identify compromised patches, authenticate and authorize spatially-based transmissions, and provide redundancy to the measurements of the state of the flow aggregated from al the patches across the airplane.

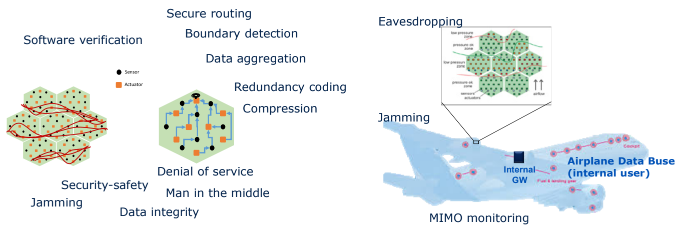


Figure 9 Security analysis of the ACF use case

Currently four attacks and security solutions are being considered in this use case. An interference jamming attack model is being considered using direction of arrival detection, higher layer detection using statistical tools or a simple passive footprint stochastic geometry model to reduce the potential attacks from pre-established directions in the aircraft. This information about the attacker, either active or passive is used in the adaptation, retransmission control, MIMO resource allocation or beamforming solution. These processes are illustrated in Figure 10.

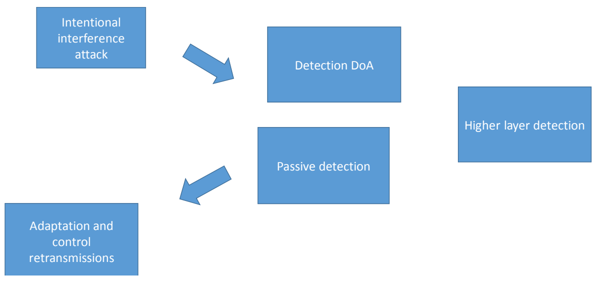

Intentional interference attack
Detection DoA
Higher layer detection
Passive detection
Adaptation and control retransmissions

Figure 10 Interference attack model and countermeasure

Eavesdropping is a passive attack common in wireless applications. When using MIMO to manage the information transmitted in different spatial direction, it is possible to deal simultaneously with the reduction of interference and the leakage of information to insecure directions where eavesdropper might be detected or where there is a high risk. The model is shown in Figure 11.

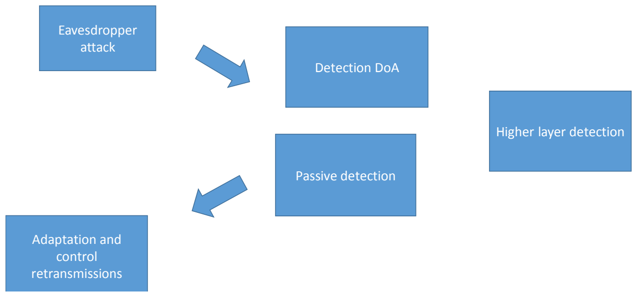


Figure 11 Eavesdropping attack model and countermeasure

Intrusion attack can lead patches to have incorrect or undesirable behaviour, producing data or incorrect feedback to the loop control. Mechanisms are being developed to provide redundancy about the flow state sensed by different patches. These mechanisms are based on a combination of physical MAC and higher layer reasoning processes. The idea is to detect patches that have been compromised and adapt all the network to reduce the influence of a compromised patch. The process is shown in Figure 12.

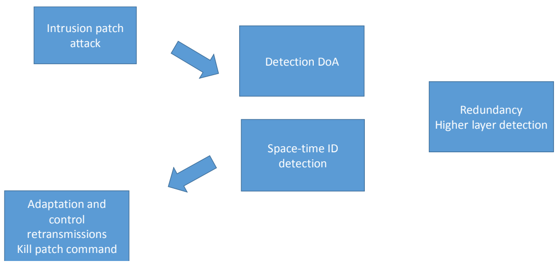

Intrusion patch attack
Detection DoA
Redundancy Higher layer detection
Space-time ID detection
Adaptation and control retransmissions Kill patch command

Figure 12 Intrusion attack and countermeasure model

Higher layer attacks are also being considered. A denial of service attack (see Figure 13) can be launched in the internal network of the aircraft, producing the lack of contact of the patches with the control unit on board the plane. Different approaches are being considered to address this issue, for example the triggering of an autonomous operation by the network of patches, distributed decision making, etc.

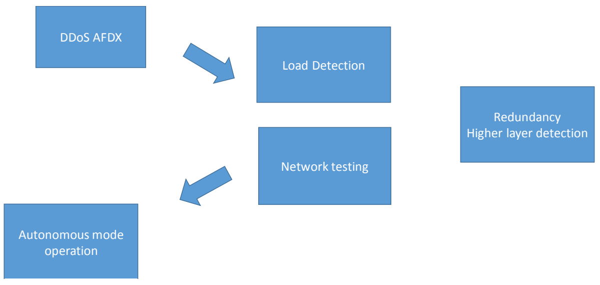


Figure 13 DoS attack and proposed countermeasure model

Conclusions

This paper has presented the architecture of the aeronautics use case for secure WAICs. Interface, objectives, requirements and preliminary vulnerability and security analysis have been conducted. The aeronautics industry will benefit from a detailed security analysis of interfaces in the context of modern IoT systems and architectures. SCOTT expects to cover several aspects in the coming years.

Acknowledgments

SCOTT has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737422. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Spain, Finland, Ireland, Sweden, Germany, Poland, Portugal, Netherlands, Belgium, Norway. Funded also by FCT/MEC, ERDF (European Regional Development Fund) under PT2020, and by CISTER Research Unit (CEC/04234).

References

[1] D. Evans, *The Internet of Things. How the Next Evolution of the Internet Is Changing Everything.* Available at http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[2] N.L. Armstrong and Y. M. M. Antar (2008), *Investigation of the Electromagnetic Interference Threat Posed by a Wireless Network Inside a Passenger Aircraft*, IEEE transactions on Electromagnetic Compatibility, vol.50, no.2, pp.277-284.

[3] ITU-R (2013), *Technical characteristics and spectrum requirements of Wireless Avionics Intra-Communications systems to support their safe operation*, Report M. 2283-0.

[4] D. Dang, A. Mifdaoui and T. Gayraud (2012), *Fly-By-Wireless for Next Generation Aircraft: Challenges and Potential solutions*, Wireless Days (WD) IFIP.

[5] SCOTT JU Grant Agreement incl. Description of Action (DoA), ECSEL Joint Undertaking, Grant Agreement No. 737422, Part B, 2017-05-18.

[6] DEWI JU Grant Agreement Annex 1 – Description of Work, 2016-12-16.

[7] R. Samano-Robles, T. Nodstrom, W. Rom, S. Santoja, and E. Tovar (2016), *The DEWI high-level architecture: Wireless sensor networks in industrial applications*, Eleventh International Conference on Digital Information Management (ICDIM), Porto.

[8] J. Liu, I. Demirkiran, T. Yang, and A. Helfrick (2008), *Communication schemes for aerospace wireless sensors*, IEEE/AIAA 27th Digital Avionics Systems Conference, 2008, 26-30.

[9] L. N. Long and S. J. Schweitzer (2004), *Information and knowledge transfer through archival journals and online communities*, AIAA Paper 2004-1264, Aerospace Sciences Meeting, Reno, NV.

[10] S. Field, P. Arnason, and C. Furse (2001), *Smart wire technology for aircraft applications*, Proceedings of the 5th Joint NASA/FAA/DoD Conference on Aging Aircraft, Orlando, FL.

[11] T. Stone, R. Alena, J. Baldwin, and P. Wilson (2012), *A viable COTS based wireless architecture for spacecraft avionics*, IEEE Aerospace Conference, Big Sky MT, pp. 1-11.

[12] K. Sampigethaya, R. Poovendran, L. Bushnell, L. Mingyan, R. Robinson, and S. Lintelman (2009), *Secure wireless collection and distribution of commercial air-plane health data*, IEEE Aerospace and Electronic Systems Magazine, vol.24, no.7, pp.14-20.

[13] D. Graham-Rowe (2009), *Fly-by-wireless set for take-off,* New Scientist, vol. 203, pp. 20-21.

[14] M. Harrington (2011), *Introduction to wireless systems in aerospace applications*, Proceedings of the CANEUS "Fly-by-Wireless" Workshop, Montreal, Canada.

[15] O. Elgezabal (2010), *Fly-by-Wireless (FBWSS): Benefits, risks and technical challenges*, CANEUS Fly-by-Wireless Workshop, Orono, ME, USA.

Revolutionize your software verification

Efficiency, Automation, Reliability

RAPITA Systems Ltd
A DANLAW Company

Unit testing · System testing · Coverage analysis · Timing analysis
V&V services · Multicore timing services · DO-178C training
Ada · C · C++

www.rapitasystems.com

NORTH - Non-intrusive Observation and RunTime Verification of Cyber-Physical Systems[*]

José Rufino, António Casimiro, Antónia Lopes
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal; email: {jmrufino, casim, malopes}@ciencias.ulisboa.pt

Frank Singhoff, Stéphane Rubini, Valérie-Anne Nicolas, Mounir Lallali, Mourad Dridi, Jalil Boukhobza, Lyes Allache
Lab-STICC UMR CNRS 6285, Université de Bretagne Occidentale, UBL, Brest, France; email: {singhoff, rubini, vnicolas, lallali, mourad.dridi, boukhobza}@univ-brest.fr

Abstract

The increasing usage of autonomous vehicles and other cyber-physical systems has motivated the adoption of Runtime Verification (RV) techniques for embedded systems. This stems from the criticality of such systems, which call for the assurance of correct operation, both on value and time domains. However, traditional RV techniques (mostly based on code instrumentation) may inevitably pose significant overheads, both in performance and timeliness, due to their inherent intrusiveness, which make them clearly unfit for critical systems.

This paper aims at advancing the state-of-art in RV techniques by presenting an innovative research observation and runtime verification method, supported in non-intrusive monitoring machinery. The negative effects of traditional techniques (ranging from function call interception to source code annotation with observation points) are avoided, making this novel approach relevant to virtually all (critical) cyber-physical systems.

1 Introduction and Motivation

Autonomous vehicles are finding their way into more applications every day. For example, drones for surveillance. These vehicles include on-board computing systems that are based on embedded processing elements, performing the necessary control functions to perform the vehicle's mission.

Given that the interaction with the environment may have very high safety requirements, the correctness of the overall system is paramount and should be ensured at all times. This may be envisaged as a general framework of the so-called Cyber-Physical Systems (CPS) with mechanisms controlled and/or monitored by computer-based techniques [1].

In order to satisfy the demanding timing, safety and security properties, some sort of correctness verification procedures are needed during the execution (runtime) stage of the system, assessing its state against a previously defined specification. The systematic and well-defined use of such procedures is called Runtime Verification (RV) [2].

Most of the current RV techniques require the modification of the application source-code. Although such code instrumentation is reasonable for larger systems, the timeliness requirements together with scarce resource availability that characterise autonomous and vehicular systems may pose an unsurpassable challenge for runtime verification in such kind of systems. Other techniques, such as system and/or function call interception, are also not free from intrusiveness [3, 4].

The goal of this work is to enable non-intrusive observation and runtime verification techniques in cyber-physical systems. This calls for advanced models, methodologies and mechanisms. Non-intrusive RV needs a novel approach based on accurate modelling of system components and embedded processing elements. These models will then be strengthened by the usage of formal temporal logics for verification rule definition, based on specifications and model properties. Then, rules are verified at runtime by independent (non-intrusive) hardware observation and monitoring mechanisms.

A comprehensive set of properties, ranging from timeliness to safety and security, shall be monitored. Timeliness and safety are crucial for the correctness of vehicle operation and for mission survivability. Security proprieties and resilience to intrusion attacks is mandatory in vulnerable systems subject to an increasing number of threats. Therefore, novel and innovative RV techniques are in need to be applied to the CPS realm. To avoid the intrusiveness shortcomings of traditional RV techniques, the observation of the system must be made non-intrusively, a fundamental step for the NORTH project.

The paper is organized as follows. Section 2 introduces the NORTH work flow. Section 3 focuses on the non-intrusive NORTH features while Section 4 discusses the evaluation of those features in NORTH-inspired systems. Section 5 describes the related work and, finally, Section 6 presents some concluding remarks and future research directions.

[*]This work was partially supported by FCT, through funding of LASIGE Research Unit, ref. UID/CEC/00408/2013, and by FCT/CAMPUS FRANCE (PHC PESSOA programme), through the transnational cooperation project 3732 (PT) / 37932TF (FR), Non-intrusive Observation and RunTime verification of cyber-pHysical systems (NORTH). This work integrates the activities of COST Action IC1402 - Runtime Verification beyond Monitoring (ARVI), supported by COST (European Cooperation in Science and Technology).

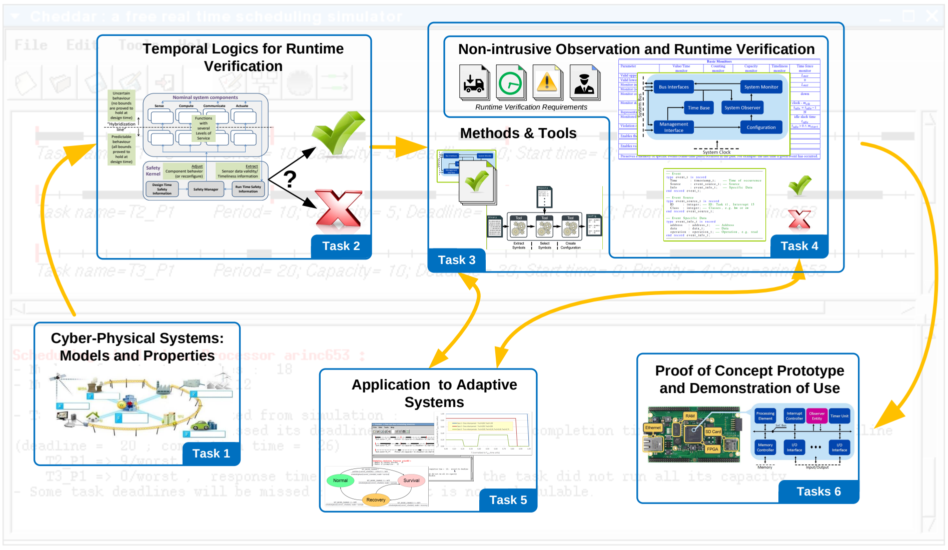

Temporal Logics for Runtime Verification
Task 2
Non-intrusive Observation and Runtime Verification
Runtime Verification Requirements
Methods & Tools
Task 3
Task 4
Cyber-Physical Systems: Models and Properties
Task 1
Application to Adaptive Systems
Task 5
Proof of Concept Prototype and Demonstration of Use
Tasks 6

Figure 1: The NORTH project work flow.

2 NORTH Project Work Flow

Understanding and formalising the properties of a cyber-physical system and how temporal logics can be used to verify those properties, is one key point of research. The gained knowledge will then be consolidated in a methodology and in the architecture of a tool for assisting in the definition and expression of runtime verification mechanisms. These should be combined with mechanisms for self-adaptability, allowing the dynamic definition of new sets of observation points. A proof of concept demonstrator merges the outcomes of the activities, sketched in the diagram of Figure 1.

T1. CPS Component Modelling and Property Extraction this task aims at modelling the several components in a cyber-physical system, both hardware and software, identifying/extracting the properties that can be subjected to runtime verification activities. The hardware may include not only the computing platform but also a relevant set of sensors/actuators. The software part of the system includes the control components. However, it may include also properties of data processing (validity and fusion), (self-)awareness, (self-)adaptability, perception, collaboration, among others. On the environment level, dealing with uncertainty is a must.

T2. Temporal Logics for Runtime Verification of CPS this task aims at exploring the applicability of Temporal Logics to the runtime verification of cyber-physical systems, and therefore study how the properties extracted in the previous task can be verified at runtime by such logics. Properties that should be verified at runtime may include: load bounds, timeliness, safety and security.

T3. Methods and Tools for Runtime Verification of CPS this task aims at defining the architecture and flow for the integration of the results stemming from the previous task into the specification of runtime verification clauses. The definition and design of special-purpose tools, augmenting and extending the scope of existing standard tools (e.g., the GNU binutils), may be required for adequate assistance in this process. This task defines a set of points of interest that should be non-intrusively observed and verified: variables and data-types; dynamically allocated memory (if it exists); system or function calls; exception handling.

T4. Non-intrusive Observation and Runtime Verification this task aims at designing a runtime verification system, using non-intrusive observation and monitoring machinery, implemented in hardware, as a basis for supporting runtime verification. The results from the previous tasks are used to map the temporal logic formulas into sets of data and/or event observation points, to be configured in the hardware machinery and into runtime verification assertions to be checked on the monitoring activities, which may then take a decision on system correctness. The RV system may be restricted to use a set of few fundamental monitors complemented with some additional functional blocks.

T5. Adaptive Non-intrusive Observation and RV of CPS this task aims at designing self-contained hardware-based mechanisms allowing the dynamic definition of new sets of observation points, due to normal changes in the operational conditions of the system, while maintaining the same runtime verification assertions. For example: a running program makes a function call (either recursive or not) that creates a new stack frame; runtime verification needs to define a new set of observation points for the recently created stack frame but the runtime verification assertions (e.g., non-violation of stack frame boundaries) are invariant.

T6. System Prototype for RV and Demonstration of Use this task aims at creating a prototype merging both the toolset architecture and flow to demonstrate the usage of effective runtime verification for cyber-physical systems.

3 Non-intrusive Runtime Verification tools

In the next sections, we focus on one kind of properties monitored and verified in the NORTH project: the timing properties. To evaluate the approach proposed in NORTH for such properties, we have developed two tools. The first one monitors a target system at runtime and triggers the collection of a trace of scheduling events whenever some condition on the sequence of events is not respected. The second tool can verify online more complex temporal scheduling properties from the execution traces.

3.1 Scheduling monitoring tool

More precisely, the aim of the first tool, which is called the health monitor, is to verify, at runtime, the conformity of a cyber-physical real-time system with the specification of its task model. The task model may be defined during the early steps of a design process and specifies the execution sequence and duration of the software tasks. The engineers use this model to verify the schedulability of their design before execution, by means of scheduling simulation tools such as Cheddar [5].

A scheduling simulation result constitutes a deterministic reference that can configure the health monitor. We propose a hardware implementation of a monitor which integrates a micro-sequencer in charge of verifying the occurrence of a timed sequence of scheduling events on the target system. The hardware also includes a module for the event capture, an event recorder, a timestamp generator and some communication logic to transfer the event traces. In case of erroneous behaviour, with respect to the expected task scheduling, the event trace is sent to a supervision station for further analysis.

3.2 Online verification tool

The second tool is a verification tool which performs the temporal scheduling properties verification on system execution traces. It has to be embedded as a component of the health monitor and to be executed in line during the system execution. Therefore, its execution speed has to be compliant with the system requirements and its memory footprint must stay as low as possible. In addition, the monitored systems may have non finite executions or long finite executions. During execution, the verification tool should not consider as input the whole trace, but only a finite fixed size slice of it (by using a transition buffer filled by the monitor hardware part). The verification tool execution time on one slice must be lower than the system execution time corresponding to the next trace slice, otherwise some trace events may be lost. For all these reasons, the verification algorithm performs only one pass through the trace.

The verification tool adopts the same system model and trace model used in the Cheddar tool. Its verification algorithm is based on a representation of the system state (including task and resource states), and starting from an inactive initial state (built from the system model), simulates the execution represented by the trace, event by event. At the same time, and depending on the properties to verify, some checks are

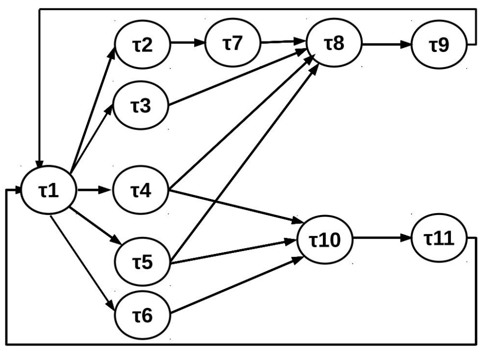


Figure 2: Modeling of the ROSACE application for Cheddar

done on event occurrences or periodically at the end of each time sequence. Periodic checks concern the tasks reaching the end of their period and are needed to cope with possible missing events in the trace, such as missing task activation events. They also allow us to complete the detection of undue locked resources or task missed deadlines.

4 Evaluation

The tools described in the previous section have been validated by several experiments.

The monitoring tool has been implemented on a Xilinx System-On-Chip (SoC) Zynq7000. The hardware monitor has been synthesized from a VHDL[1] model and the resulting design occupies less than 5% of the FGPA resources.

A simple experiment composed of two tasks has been successfully run to demonstrate the ability of the monitor to record the corresponding scheduling event on a Real-Time Executive for Multiprocessor Systems (RTEMS) target [6].

The online verification tool has been implemented in C and applied on several case examples produced from Cheddar. From those examples, four properties have been identified: missed deadlines, deadlocks, priority inversion and lock resources.

Larger experiments based on a mixed-criticality system case study are also planned during the project.

As any mixed-criticality system, the case study is composed of several applications with different levels of criticality [7]. In this context, the different levels of criticality imply different levels of guarantee on the application deadlines. As an example, a two level mixed-criticality system may contain high-criticality applications on which the deadlines must be met and low-criticality applications on which the deadlines are allowed to be missed.

The case study of the NORTH project is a drone system with 3 criticality levels [8]. The highest criticality level is a flight controller software called ROSACE [9]. ROSACE is a data flow oriented application composed of a set of periodic dependents tasks and requires that all task deadlines are met. Figure 2 shows a model of ROSACE made for Cheddar. Each task is defined by the classical periodic task parameters, i.e.

[1]**V**ery **H**igh-Speed Integrated Circuit **D**escription **L**anguage.

WCET (Worst Case Execution Time), period, priority, release time and deadline.

The dependencies between tasks are expressed by a specific priority assignment. The middle criticality level application is a path planning algorithm computing online the path of the drone according to its environment.

Finally, the lowest criticality application is a video application with a high computation need which simulates a video surveillance system embedded in the drone. All the applications run on a POSIX RTEMS [6] target and are written in C.

5 Related Work

The application of non-intrusive runtime monitoring to embedded systems has been discussed in [3, 4] and, more specifically, in safety critical environments [10].

Configurable minimally intrusive event-based frameworks for dynamic runtime monitoring have been developed [11]. Additionally, the RV concept has been applied to autonomous systems [12] and to a AUTOSAR-like RTOS, aiming the automotive domain [13]. A runtime monitoring approach for autonomous vehicle systems requiring no code instrumentation by observing the network state is described in [14].

However, to the extent of our knowledge, no such techniques have been applied to aerospace systems, especially if critical avionic applications are combined with other non-critical applications.

6 Conclusion

In this paper we have presented NORTH. The NORTH project is a collaborative project between the LASIGE/Univ. Lisboa and the Lab-STICC/Univ. Bretagne Occidentale aiming to investigate and evaluate a runtime verification platform for embedded real-time systems.

The NORTH project addresses the study of Non-intrusive Observation and Runtime Verification in a comprehensive way, from conceptual tasks such as component modelling and property extraction to implementation and prototyping, passing through methods and tools for building a (self-)adaptive RV architecture.

Those propositions led to the development, up to the moment, of two tools: a hardware scheduling monitor implemented on a Field Programmable Gate Array (FPGA) board and running a RTEMS target, and a runtime analysis tool written in C and allowing online detection of task scheduling errors.

In the next steps of the project, the tools will be evaluated on a drone case study running a mixed-criticality system. A mixed-criticality system is both composed of high and low criticality tasks and cannot be designed by resource reservation only. Thus monitoring and verification for online resources management is expected to be an interesting approach in this context.

References

[1] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang (2012), *Toward a science of cyber-physical system integration*, Proc. of the IEEE, vol. 100.

[2] M. Leucker and C. Schallhart (2009), *A brief account of runtime verification*, The Journal of Logic and Algebric Programming, vol. 78, pp. 293–303.

[3] C. Watterson and D. Heffernan (2007), *Runtime verification and monitoring of embedded systems*, Software, IET, vol. 1.

[4] T. Reinbacher, M. Fugger, and J. Brauer (2014), *Runtime verification of embedded real-time systems*, Formal Methods in System Design, vol. 24, no. 3, pp. 203–239.

[5] F. Singhoff, J. Legrand, L. N. Tchamnda, and L. Marcé (2004), *Cheddar: a flexible real time scheduling framework*, ACM Ada Letters journal, vol. 24, pp. 1–8.

[6] The RTEMS Project (2017), *Real-Time Executive for Multiprocessor Systems - RTEMS POSIX Application Programmimg Interface Guide*, release 5.0.0 ed. https://www.rtems.org.

[7] S. Vestal (2007), *Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance*, in Proceedings of the 28^{th} Int. Real-Time Systems Symposium (RTSS), IEEE.

[8] L. Allache (2018), *Mise en oeuvre d'un benchmark drone/mix-criticality*, in Master thesis, Master 2 LSE.

[9] C. Pagetti, D. Saussié, R. Gratia, E. Noulard, and P. Siron (2014), *The ROSACE case study: From simulink specification to multi/many-core execution*, in IEEE 20th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 309–318, IEEE.

[10] A. Kane (2015), *Runtime Monitoring for Safety-Critical Embedded Systems*, PhD thesis, Carnegie Mellon University, USA.

[11] J. C. Lee and R. Lysecky (2015), *System-level observation framework for non-intrusive runtime monitoring of embedded systems*, ACM Transactions on Design Automation of Electronic Systems, vol. 20, no. 42.

[12] G. Callow, G. Watson, and R. Kalawsky (2010), *System modelling for run-time verification and validation of autonomous systems*, in Proc. 5th Int. Conf. on System of Systems Engineering, UK.

[13] S. Cotard, S. Faucou, J.-L. Bechennec, A. Queudet, and Y. Trinquet (2012),*A data flow monitoring service based on runtime verification for AUTOSAR*, in Proceedings of the 14th Int. Conf. on High Performance Computing and Communications, Liverpol, UK, IEEE.

[14] A. Kane, O. Chowdhury, A. Datta, and P. Koopman (2015), *A case study on runtime monitoring of an autonomous research vehicle (ARV) system*, in Proc. 15th Int. Conf. on Runtime Verification, Vienna, Austria.

A Real-Time System Monitoring driven by Scheduling Analysis

Stéphane Rubini, Valérie-Anne Nicolas, Frank Singhoff
Lab-STICC UMR 6285, UBO, UBL, Av. Le Gorgeu, 29200 Brest, France; email: {surname.name}@univ-brest.fr
José Rufino
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal; email: jmrufino@ciencias.ulisboa.pt

Abstract

Real-Time system engineers may introduce task scheduling analysis at the early stage of the design process. System temporal behavior and task schedules are strongly related. The noncompliance to an expected schedule is a symptom of an erroneous state that may result in a serious risk for the system integrity. Spreading the design task model, as a timing reference to guide run-time verification, is a kind of extension to the model driven design paradigm.

This paper presents the overall architecture of a non-intrusive hybrid monitor. Configured by the result of a scheduling simulation, the monitor is intended to observe the system execution and raised an alarm in case of divergence with the predicted schedule. To advance this goal, a first experiment shows the scheduling of 2 tasks rebuilt from the events collected by the monitor while the RTEMS OS' *scheduler was executing.*

Keywords: health monitoring, real-time system, scheduling analysis.

1 Introduction

In real-time systems, the scheduling is a set of rules that govern the order of the processing on the system's hardware resources. Beyond the need to provide program codes functionally corrects, the designers must also ensure timeliness of theirs results. To fulfill timing requirements of real-time systems, the scheduling of the tasks must be taken into account at the early stages of the design. Scheduling analysis works from an abstract view of tasks, the task model, which defines their timing behavior independently of the nature of computation they have to do.

We propose to extend the scheduling simulation field of use onto run-time verification of hard real-time systems. Hard real-time systems are characterized by deterministic execution and strict time constraints. From a given task model that specifies the timing parameters to enforce, the analysis tools verify their respect during the design step. They also define a deterministic awaited execution trace of the system tasks at run-time.

A *health monitor* can observe at run-time the sequence of events that describes the evolution of the task states from the schedule point of view, and compare them to those predicted by the simulation. Our goal is to configure the monitor from the task models, and then to use a unified specification from the design of the system to its run-time supervision. This paper focuses on the overall hardware architecture of the monitor. We evaluate its ability to collect and report a trace of scheduling events observed on a target system. Only an initial and restricted version of the scheduling comparison module is presented here as a more complete implementation remains to be developed.

The outline of the paper is the following. The first part characterizes the task models we want to monitor. Next, the architecture of the health monitor is described. Some technical problems about the rebuilding of long term time stamps is emphasized. Section 3 presents the status of the project, and the results of the initial experiments. The last part of the article describes some related works and concludes.

2 Task model and run-time verification

The systems we are targeting for run-time monitoring are time-triggered hard real-time systems on uni-processor execution platform, like the systems that control the critical functions in transport vehicles. The number of tasks and their parameters (e.g. deadlines, release times) are fixed and specified at design time. Tasks are periodic and the complete system itself has a repetitive temporal behavior, eventually achieved after a stabilization time that can be determined by scheduling analysis. We observe such a stabilization phase when the initial release time is not the same for all tasks (i.e. the offset parameter of some tasks is different from 0). The scheduling of the tasks, computed off-line or by static schedulers, must be deterministic.

From the above assumptions, the scheduling trace produced by a scheduling analysis tool from a given task model may serve as a "golden reference" for the verification of a system also implementing this task model. Fig. 1 sums up this approach.

The main interest to do monitoring at the schedule level is the restricted number of event types to observe and their common semantics on multiple systems. As opposed to the operations

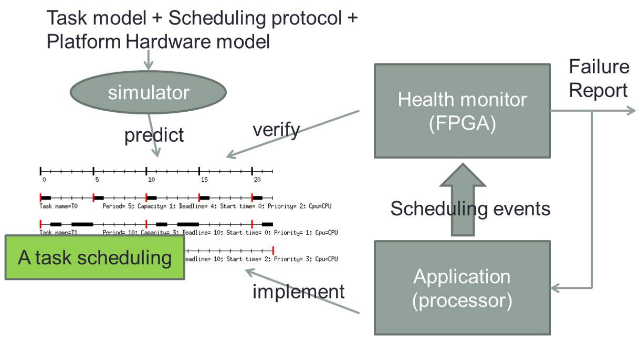


Figure 1: Scheduling analysis as "golden" reference.

performed by the tasks which are generally different for each application, scheduling concepts remain similar.

A challenge of the implementation of this approach is to define how to manage discrepancies between the task model specification, the simulation results and the real execution on the target platform. For instance, the release of a set of tasks can be stated as simultaneous for the scheduling while related events are emitted and detected in a sequential order. The matching of the physical time as approximated in the observed system, in the monitor and in the result of simulation constitutes another example of practical problems that need to be solved.

The first step of the approach is based on a scheduling simulation tool. Our team has already developed such a tool, that is called Cheddar [1]. The second step is to have a monitor which allows us to observe the real-time system by inducing a weak perturbation. The next section presents the architecture of this health monitor, and gives details about the implementation of some of its functions.

3 Monitor Architecture

The monitor verifies that the trace of observed events is conform to the scheduling simulation predictions. Hence, there is no need for reporting the events if the system works as expected.

However, more meaningful information is how the system goes into an erroneous state, that is, from the monitor observation, what is the preceding sequence of events before the failure. This working mode will be named in the sequel *back trace mode*. But, the analysis of the consequence of an error can also be another outcome of the monitor report. In this second case, the monitor switches in a *forward trace mode*, which transfers all captured events to the supervision station.

So, the hardware monitor we are developing is structured following the previous objectives: run-time verification of the scheduling, and reporting of the cause or the consequences of a discrepancy. Fig. 2 shows the 5 main components of the its architecture and their interactions.

The "event capture" component is in charge of events collect, either by observing the behavior of the monitored system from an external point of view, or by receiving the event explicitly transmitted to it. A time stamp is adjunct to each

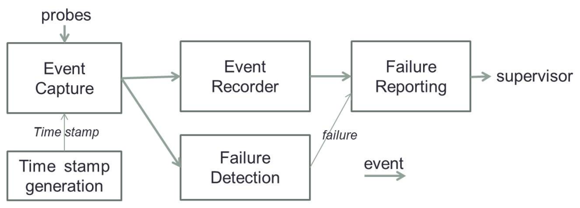


Figure 2: Hardware monitor architecture.

traced event, generated by the "time stamp generator". Time stamped events are stored in the "event recorder", while the "Failure detection" component verifies that the sequence of events respects an expected order and some timing constraints. At last, the failure reporting component aims to extract the event trace from the monitor, to carry it on a supervision station for post-processing and analysis.

The next paragraphs give details about the design and functions of these 5 components.

3.1 Event capture and recording

Inside the recorder, a FIFO buffer, implemented by a circular array, stores the collected events, associated to their time stamps. When the buffer is full, the oldest events are forgotten.

If the monitor enters into reporting mode, the event recorder behavior depends on the chosen trace mode. In back trace mode, event recording is stopped as soon an erroneous event sequence has been detected, and only the events already present in the buffer are transmitted towards the supervision station. In forward trace mode, the monitor resets the array and the event recording will work as a temporary buffer between the event collector and the supervision station. If the buffer becomes full, the event recording stops, and the monitor only flushes the events available in the array. This behavior ensures that the event trace is not corrupted by intermediate missing events.

Event capture The basic interest of hybrid monitoring solutions is in reducing the interference on the observed system.

Hardware event sensors could use a technique like bus snooping [2], which limits the system disturbance. However, its implementation is technically difficult on processing systems that include complex memory hierarchy. Moreover, the point where sensors should listen could be unreachable from the monitor side [3]. Software sensors are easiest to implement but require source or OS code instrumentation. However, software sensors could impact the application temporal behavior.

Currently, we use software probes that write in monitor's memory-mapped registers. An event is coded on a 32 bits word, and is composed of an event type and a source identifier (i.e. a task identifier). The monitor manages the timestamping of an event by a dedicated hardware component (see the next paragraph for details), and so the interference on the target system is expected to be limited (few memory word transfers by task job).

3.2 Failure detection

The failure detection module is in charge of verifying the system is working as expected. A micro-coded sequencer implements this function; the micro-code is included in the hardware configuration (see section 4).

Currently, the sequencer can only detect a periodic, – after the stabilization time –, and totally ordered suite of events. The first constraint is in accordance with the assumption on the target system, whereas the second one should be partially weakened in future designs. Fig. 3 shows the architecture of the detector. The values in the microcode memory defines the sequence of expected events. The events, represented by their source and their type identifiers, must arrive before or after a given time expressed in a micro-instruction.

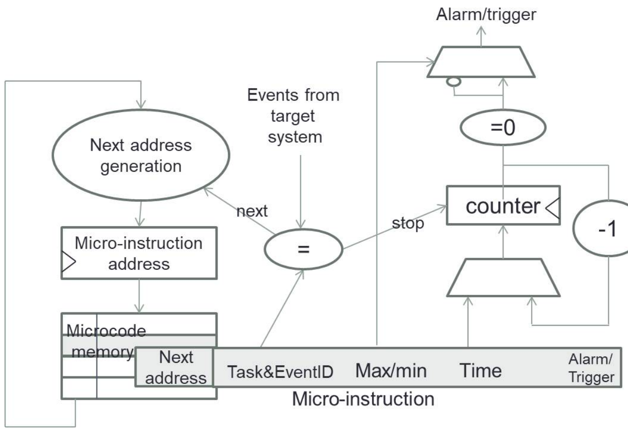


Figure 3: Failure detector (simplified hardware schematic).

We do not give anymore details about this failure detection module, because its architecture remains to be enrich to verify a more extended set of properties on the scheduling event trace.

3.3 Time stamping

Constrains of real-time system executions do not only concern the occurrence of events, but also the instant at which these events have been produced. So, time stamps go with the collected events. The following observations justify the way the time stamps is generated:

- non-intrusive: the access to the current time could imply calls to run-time (OS) services, then disrupting the execution of the observed application. The amount of information transfers to the health monitor must also be restricted to the bare minimum.
- independent; erroneous time management on the observed system could be difficult to analyze if event time stamps are issued from the same time reference.
- adapted: resolution, cycling and representation of time must be adapted to the need and the potential of the hardware monitor implementation. These requirements should be different than those of the target system.

The preceding remarks lead us to generate the time stamps within the hardware monitor, at the moment the events are received. We assume the duration of events collect trough Memory-Mapped register is constant, and therefore the time interval between 2 events is the same in the observed system and in the monitor.

The size, in number of bits, of the time stamp are constant, and must be small enough to limit (1) the storage needs to keep the trace in the monitor, and (2) the communication bandwidth to transfer the trace on the supervision station. A clock produces the time stamp, whose resolution depends on a periodic signal generated by frequency division from the basic system clock.

Fig. 4 shows the synopsis of the time stamp generation circuit: The frequency divisor creates periodic ticks at a frequency defined at the start up of the monitored system. This signal controls the increasing of a counter which gives time stamps when needed for a new event.

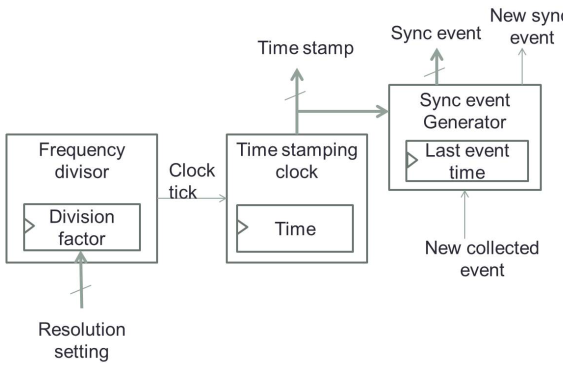


Figure 4: Time stamp management (simplified hardware schematic).

Considering a divisor factor register on 24 bits, a time register implemented on 16 bits, and a basic system clock at $100MHz$, the timer resolution goes from $10ns$ to $160ms$, and the timer counter overflow (cycling) occurs after about $0.6ms$ at the worst case.

The instant t_i at which an event i occurs is $t_i = k_i.t_{cycle} + ts_i.t_{res}$, with $k_i \in \mathbb{N}^+$, t_{res} and t_{cycle} being the timer resolution and the timer cycling period respectively. ts_i is the time stamp bound to the event i; k_i represents the number of counter overflows since the starting of the system.

The supervision station can get ts_i, t_{res} and t_{cycle}, but does not have access to k_i, since the event trace has been collected in the past, and only time stamps ts_i are associated to the events. So, to be able to rebuild the event instant at the level of the supervision station, the monitor must be sure to receive the next event within the counter overflow period subsequent to a given event. With this condition, $k_i = k_{i-1}$ if $ts_i \geq ts_{i-1}$, and $k_i = k_{i-1} + 1$ otherwise.

The instant t_i can be computed from the instant of the previous event by the following equation:

$$t_i = t_{i-1} + \begin{cases} (ts_i - ts_{i-1}).t_{res} & if \quad ts_i \geq ts_{i-1} \\ t_{cycle} + (ts_i - ts_{i-1}).t_{res} & if \quad ts_i < ts_{i-1} \end{cases}$$

To build the series of the event instants, the condition previously stated, i.e. the time between two collected events is less than the timer overflow period, must be ensured by the monitor. The component "Sync event generator" in Fig. 4 produces pseudo "sync" events to respect a minimum rate of event occurrence.

4 Implementation and first Experiments

Hardware platform The development board "ZedBoard" built by *AVNET* has been chosen to evaluate the ability of the hardware monitor to verify the health of a real software at run-time. This board is built around a *Xilinx* System-On-Chip Zynq7000. The Soc Zynq contains a Dual ARM Cortex A9 core processing system, and a programmable logic area of the family *Virtex7*. The connection with the supervision station (a Linux PC) is based on a USB2 serial link (115200 bauds UART emulation).

A VHDL model of the hardware monitor has been synthesized and implemented into the Zynq's FPGA. The current capacity of the event recorder is 1024 events. The failure detection can recognize sequences of 32 events. The micro-instructions that encode the sequence are stored in an internal memory (BRAM), whose the content is currently defined in the VHDL model. However, it is also possible to populate the BRAM by a direct updating of the FPGA configuration. With these parameters, the circuit occupies less than 5 % of the available FPGA resources whatever their type (LUT, BRAM ...).

First experiment: A two-task system This first experiment verifies the monitor ability to collect, time stamp and transfer events to a supervisor station in the forward trace mode. We consider a simple task model composed of 2 tasks, whose the periods are 20 and $5ms$, and the capacities 6 and $2ms$ respectively. The second task has a greater priority than the first one. The RTEMS OS[1] controls the target system. Its *Deterministic Priority Schedule*r [4] has been instrumented to signal scheduling events.

The Cheddar tool[2] is a scheduling analysis tool able to select and apply a set of analysis methods from a given task model and execution platform. Scheduling simulation results can be exported or imported as an XML file which contains the instants of significant scheduling events. Fig. 5 is a visualization of the events collected by the monitor after importation into *Cheddar*. The first period of each task appears as too short, due to time rounding in Cheddar and RTEMS tick resolution ($1ms$ in this experiment).

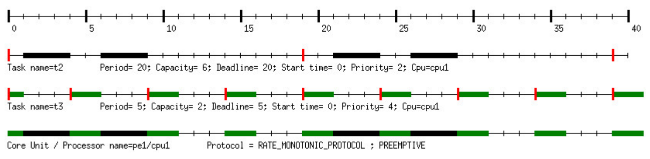


Figure 5: Collected events shown in the Cheddar's tool time line. Axis time unit represents 1 ms.

[1]http://www.rtems.org

[2]http://beru.univ-brest.fr/~singhoff/cheddar/

The transformation of the simulation trace into an expected and timed sequence of events must meet several challenges: How to relate logical simulation time and real execution ones, how to deal with task model abstraction (0-cost task switching for instance), how to order simultaneous simulation events, object matching between the simulation and the execution platform ...

5 Related works

An overview and a classification of monitors focused on timing constraints is established in [5]. Criteria like the adaptability, data collection methods, type of targeted systems or the monitor implementation organize the classification. Following this classification, our monitor is a "hybrid" monitor, based on a "tracing" data collection method and dedicated to the observation of "general" "real-time" and "embedded" system target.

In [6], Bandur et al. show how to implement a timed automaton on a micro-controller. The execution time of instructions in this micro-controller must be deterministic. The supported timed automaton assumes only one clock and that the time interval of concurrent outgoing transitions must be the same. The approach of this article could be a basis for an improvement of the "failure detection" module in our monitor.

Finally, Peters and Parnas argue in [7] that monitors should be based on the design requirements of the observed systems. They identify some necessary condition allowing a monitor to be feasible. The approach we propose follows this idea, although the parameters of a task model can be seen as a derivative of the system specification.

6 Conclusion

This article describes the overall architecture of a hardware monitor. First experiments have shown the ability of the monitor to collect the scheduling events of a sample task model controlled by the RTEMS operating system. The execution time of a software event sensor in the scheduler is less than $200ns$ and its intrusivity is limited.

Our goal is to derive automatically the monitor configuration from the real-time system task model and its scheduling. To achieve this objective, various assumptions or choices must be expressed and then specified in the design models. An architecture description language like AADL [8] can both specify the task model for the scheduling simulator and supply matching information to configure the monitor. We expect that expressing such information should contribute to increase the quality and conformity of the systems implementation.

Acknowledgments This work and Cheddar are supported by Brest Métropole, Ellidiss Technologies, CR de Bretagne, CG du Finistère and Campus France PESSOA programs number 27380SA and 37932TF.

References

[1] F. Singhoff, J. Legrand, L. Nana, and L. Marcé (2004), *Cheddar: a flexible real-time scheduling framework*, ACM SIGAda Ada Letters, vol. 24, pp. 1–8, ACM Press, New York, USA.

[2] J. J. P. Tsai, K.-Y. Fang, H.-Y. Chen, and Y.-D. Bi (1990), *A noninterference monitoring and replay mechanism for real-time software testing and debugging*, IEEE Transactions on Software Engineering, vol. 16, no. 8, pp. 897–916.

[3] M. M. Gorlick (1991), *The flight recorder: an architectural aid for system monitoring*, in ACM SIGPLAN Notices, vol. 26, pp. 175–181, ACM.

[4] G. Bloom and J. Sherrill (2014), *Scheduling and thread management with RTEMS*, ACM SIGBED Review, vol. 11, no. 1, pp. 20–25.

[5] N. Asadi, M. Saadatmand, and M. Sjödin (2013), *Run-time monitoring of timing constraints: A survey of methods and tools*, in Proceedings of the the 8^{th} International Conference on Software Engineering Advances (ICSEA), Venice, Italy, pp. 391–401.

[6] V. Bandur, W. Kahl, and A. Wassyng (2012), *Microcontroller assembly synthesis from timed automaton task specifications*, in International Workshop on Formal Methods for Industrial Critical Systems, pp. 63–77, Springer.

[7] D. K. Peters and D. L. Parnas (2002), *Requirements-based monitors for real-time systems*, IEEE Transactions on Software Engineering, vol. 28, no. 2, pp. 146–158.

[8] P. H. Feiler and D. P. Gluch (2012), *Model-based engineering with AADL: an introduction to the SAE architecture analysis & design language*, Addison-Wesley, 2012.

Hardware Support to Non-intrusive Runtime Verification on Processor Technologies*

José Rufino
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal; email: jmrufino@ciencias.ulisboa.pt

Abstract

Software-based instrumentation probes always disturbs the functional and non-functional properties of a system, even if in a minimal way. To avoid the disturbance of system operation, by instrumentation probes, nonintrusive runtime verification must rely on hardware-based technology. This paper reviews classical processor technology to understand which kind of support is provided on each processor family, its intrusiveness, functionality and offered system support.

Keywords: Hardware-based runtime verification

1 Introduction

The traditional approach to runtime verification is to instrument the software of a functional system with small pieces of code that, acting as observers, assess the software state in runtime. Software-based instrumentation inherently disturbs the functional or non-functional properties of a system, namely with respect to timing properties, which are crucial to embedded and real-time system design [1, 2, 3]. They always exhibit some degree of intrusiveness, even if minimal.

Software-based observing components affect the normal behaviour of the observed system, throughout what is called "the observer effect" or "the probe effect" [4]. The delays implicitly associated with the insertion of software-based probes may affect the timing characteristics of concurrent programs. The removal of such probes from the software, which will lead to shorter program/task execution times, may render a given task set unschedulable, due to changes in the corresponding cache-miss profile [5, 6, 7].

Hardware-based approaches are inherently non-intrusive, i.e. they do not affect system operation. Though hardware-based observation is in essence non-intrusive, monitoring functions, i.e. runtime verification (RV) may have some degree of intrusiveness. Non-intrusiveness, may then be referred to as a RV constraint. RV constraints are not only relevant, but in fact fundamental, for highly critical systems [2].

*This work was partially supported by FCT, through funding of LASIGE Research Unit, ref. UID/CEC/00408/2013, and by FCT/CAMPUS FRANCE (PHC PESSOA programme), through the transnational cooperation project 3732 (Portugal) / 37932TF (France), Non-intrusive Observation and Run-Time verification of cyber-pHysical systems (NORTH). This work integrates the activities of COST Action IC1402 - Runtime Verification beyond Monitoring (ARVI), supported by COST (European Cooperation in Science and Technology).

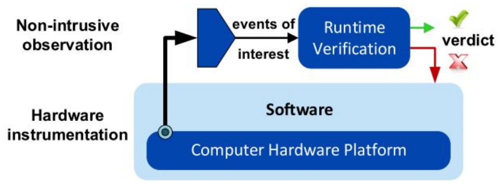


Figure 1: Non-intrusive runtime verification

This paper reviews classical processor technology to understand which kind of support is provided on each processor family, its intrusiveness, functionality and, in general, offered system support.

A comprehensive overview of various hardware (including on-chip), software and hybrid (i.e., a combination of hardware and software) methodologies for system observation and verification of software execution in runtime is provided in [8]. System observing solutions can be designed to be directly connected to some form of system bus, enabling information gathering regarding events of interest, such as data transfers and signalling taking place inside the computing platform, namely instruction fetch, memory read/write cycles and interrupt requests, with no required changes on the target system's architecture, as shown in the diagram of Figure 1. Examples of such kind of hardware-based observation approaches are proposed in [9, 10, 11, 12, 13].

The paper is organized as follows. Section 2 presents a description of the previous related work. Section 3 reviews the classical processor technology looking for non-intrusive runtime verification support. Section 4 describes the evaluation experiment for a particular processor technology (SPARC LEON) and, finally, Section 5 presents some concluding remarks and future research directions.

2 Previous Work

The application of non-intrusive runtime monitoring to embedded systems has been discussed in [8,14] and, more specifically, in safety critical environments [13].

Configurable minimally intrusive event-based frameworks for dynamically runtime monitoring was developed in [15], which was later complemented with a combination of hardware and software observability [3].

Additionally, the RV concept has been applied to autonomous systems [16] and to a AUTOSAR-like real-time operating

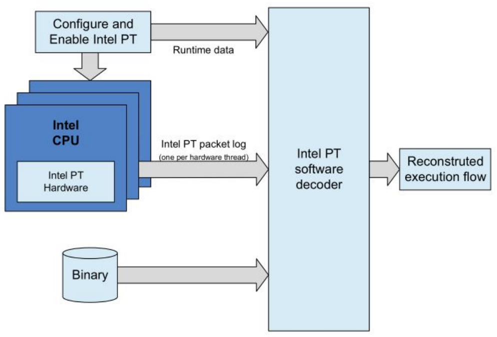

Configure and Enable Intel PT
Runtime data
Intel CPU
Intel PT Hardware
Intel PT packet log (one per hardware thread)
Intel PT software decoder
Reconstruted execution flow
Binary

Figure 2: Intel processor trace (Intel PT)

system aiming the automotive domain [17]. A runtime monitoring approach for autonomous vehicle systems requiring no code instrumentation by observing the network state is described in [18].

High quality trace data in a multi-core environment uses an approach based on non-intrusive full observation, meaning not only the program counter, but also other data read/write cycles, cache and bus operations are included in the trace [9].

A set of first contributions and discussion of technical issues such as metadata management, format and storage on practical examples are addressed in [19]. A description of the fundamentals of a trace are presented in [20].

3 Processor Technology

This section reviews different processor families to determine the support they provide, its intrusiveness and functionality.

3.1 Intel: Processor Trace

The Intel processor trace (PT) [21] is an extension of the Intel Architecture that captures information concerned with software execution, on each hardware thread, using dedicated hardware facilities. So, when an execution completes some special-purpose software can do processing of the captured trace data and reconstruct the exact program flow (Figure 2). Intel PT has an execution overhead cost: though a target less than 5% overhead is desirable, there are some applications with 35% overhead, being 20% an average value.

The captured information is collected in data packets, as described in [22] and summarized next. A set of packets (Packet Stream Boundary, PSB and Paging Information Packet, PIP), act as heartbeats generated at regular intervals (every 4 KiB) and record changes in attributing a linear address to an application. The MODE packet provides the decoder relevant execution information for binary interpretation and trace log and the Overflow (OVF) packet is issued when a processor experiences an internal buffer overflow. Three different packets, ranging different precisions, are used to get time information: Timestamp Counter (TSC) which provides some portion of a software-visible timestamp counter; Mini Timestamp Counter (MTC) which is more frequent and used with TSC to get accurate timestamps for less cost; Cycle Counter (CYC) packets

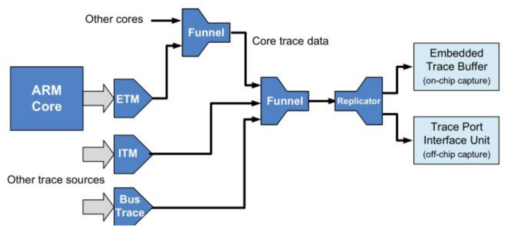


Figure 3: ARM CoreSight

provide even finer grain timestamp information. The Core Bus Ratio (CBR) contains the core bus clock ratio.

In a control flow tracing context, the following packets are used: Taken Not-Taken (TNT) tracks the direction of conditional branches (taken or not taken); Target IP (TIP) record the target value of the IP (Instruction Pointer) register in indirect branches; Flow Update Packet (FUP) provide the value of the IP for asynchronous events (interrupt and exception).

Each packet of the trace output is written to memory in a collection of variable-sized regions of physical memory. Therefore, with the knowledge of binary information, one can reconstruct the entire control flow of the original software, together with the precise timing of each branch.

Since the decoding of the traces is "several orders of magnitude slower than tracing", one may think a proprietary design where the Intel PT decoder memory area is set as a dual-port memory device, thus providing independence and allows non-intrusive runtime verification. However, these schemes are very specialised.

3.2 ARM: CoreSight Technology

The next system we analyse is based on the ARM technology and its non-intrusive observation scheme, generically known as ARM CoreSight [23, 24, 25].

The architecture of ARM CoreSight is represented in Figure 3. The simplest form of trace is that generated by the software executing on the cores. Optimizations on this approach allow writing to the ARM Instrumentation Trace Macrocell (ITM), which streams the trace data direct to a trace buffer, as shown in Figure 3. This provides a high bandwidth channel that allows the delivery of more instrumentation points. However, the drawback of this approach is its natural intrusiveness.

To avoid instrumentation, hardware trace is an option, materialized by the ARM Embedded Trace Macrocell (ETM), is extremely popular. As shown in Figure 3, there is one ARM ETM for each core. In hardware trace, special-purpose logic watches the address, data and control signals within the System-on-Chip (SoC) compresses that information and emits to a trace buffer, which itself can be subdivided in to three main categories: program/instruction trace; data trace; and bus (or interconnect fabric) trace. The ARM ETM is thus a non-intrusive observer.

In terms of cost, for program/instruction trace macrocells can be quite small: only one byte/instruction/processor is required. Unfortunately, the cost of implementing data trace

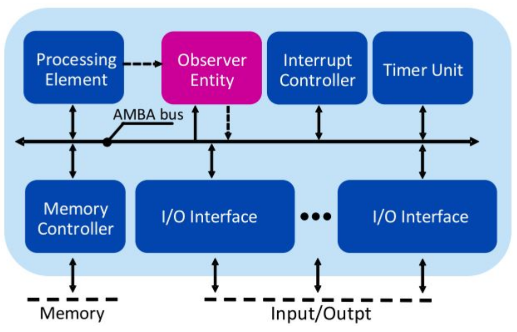


Figure 4: SPARC LEON processor and observer entity

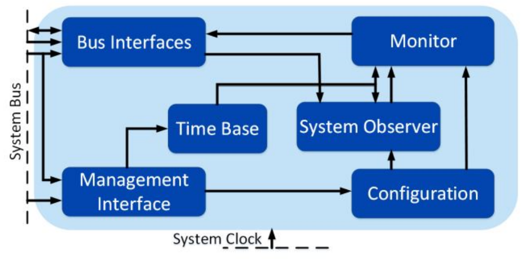


Figure 5: Observer Entity Architecture

is highest: trace macrocells need to be larger, data is more difficult to compress (data trace from an ARM ETM typically requires 1-2 bytes/instruction/processor). Each captured trace data have attached a timestamp.

The collected data is replicated and presented in two different resources: an internal (on-chip) embedded trace buffer; a trace port allowing the captured data to be externally processed.

3.3 SPARC LEON: Dedicated Observer

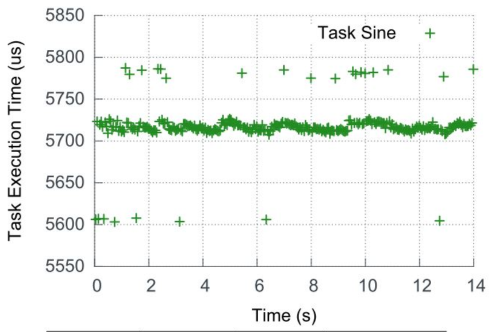
The next system we analyse is embedded in a SoC system with a LEON processor [26], a SPARC CPU [27], embodying a state-of-the-art computing architecture. The LEON is the reference architecture for European Space applications, e.g. satellites, being also used in other real-time control applications. The SoC bus is the AMBA bus [28]. A block diagram with the global system is presented in Figure 4.

Since SPARC LEON does not have specific tools for code observation and tracing, one have designed one (also shown in Figure 4). The Observer Entity (OE) infrastructure can observe the AMBA bus and capture a set of relevant events: instruction fetch; memory read/write cycles; interrupt requests. Alternatively, the OE can be plugged in a cache internal bus, for a more precise observation.

The OE is specified in VHDL[2] and the event capture is independent and made in parallel with the operation of the functional system. Therefore, the OE integrates all the mechanisms required for a non-intrusive observation. The monitor option supports non-intrusive runtime verification.

The OE comprises the modules of Figure 5: Bus Interfaces, capturing all physical bus activity, such as bus transfers or interrupt vectors; Management Interface, enabling observer entity configuration; Configuration, storing a dynamically defined set of events; the System Observer itself, detecting events of interest; Monitor, which detects possible violations to the specified system behaviour; Time Base, which allows to time stamp the events of interest.

[2]Very High-Speed Integrated Description Language.



Minimum μs	Maximum μs	Average μs	Std. Deviation
5603.180	5787.240	5719.536	22.686

Figure 6: Task Execution Time Measurement

4 Evaluation

An example of a runtime monitoring function is presented next, assuming the use of a SPARC LEON processor; as software counterpart an application running on the RTEMS real-time operating system is used [29]. The software system under evaluation is composed by a task, named Task Sine, which produces a sine wave with a given frequency.

The task is executed periodically, with a 50 ms period. The monitoring aims at measuring the execution time of the task as well as its amplitude. Both the execution time and the amplitude are monitored. This data is represented in a graphical manner through Figure 6, together with a table containing its statistical analysis. The null competition for the processing resources allows Task Sine to exhibit a somewhat stable execution time, i.e. with low variance. In this experiment, given the monitoring bounds, no error is detected. This will not be the case if the monitoring values have a lower bound.

5 Conclusion

This paper reviews classical processor technology to understand which kind of support is provided on each processor family (Intel, ARM and SPARC LEON), its intrusiveness, functionality and offered system support.

Each processor family was reviewed and we characterize the offered support to observation. Together with this, we address the non-intrusiveness and functionality.

For the SPARC LEON, which received a freshly designed non-intrusive runtime verification scheme, we have conducted a very simple experiment that evaluate the proposal.

References

[1] M. E. Shobaki and L. Lindh (2001), *A hardware and software monitor for high-level system-on-chip verification*, in Proceedings of the 2nd IEEE International Symposium on Quality Electronic Design (ISQED 2001), (San Jose, CA, USA), pp. 56–61.

[2] L. Pike, S. Niller, and N. Wegmann (2011), *Runtime verification for ultra-critical systems*, in 2nd International Conference on Runtime Verification (RV 2011), (San Francisco, USA), pp. 310–324, Springer.

[3] J. C. Lee and R. Lysecky (2015), *System-level observation framework for non-intrusive runtime monitoring of embedded systems*, ACM Transactions on Design Automation of Electronic Systems, vol. 20, no. 42.

[4] J. Gait (1986), *A probe effect in concurrent programs*, Software - Practise and Experience, vol. 16.

[5] T. Lundqvist and P. Stenstrom (1999), *Timing anomalies in dynamically scheduled microprocessors*, in Proc. of the 20th Real-Time Systems Symposium, IEEE, Dec. 1999.

[6] R. Wilhelm et al (2008), *The worst-case execution time problem - overview of methods and survey of tools*, ACM Transactions on Embedded Computing Systems (TECS), vol. 7, Apr. 2008.

[7] T. H. Nam (2017), *Cache Memory Aware Priority Assignment and Scheduling Simulation of Real-Time Embedded Systems*, PhD thesis, Université de Bretagne Occidentale, Brest, France.

[8] C. Watterson and D. Heffernan (2007), *Runtime verification and monitoring of embedded systems*, IET software, vol. 1, pp. 172–179.

[9] R. Backasch, C. Hockberger, A. Weiss, M. Leucker, and R. Lasslop (2013), *Runtime verification for multicore SoC with high-quality trace data*, ACM Trans. on Design Automation of Electronic Systems, vol. 18.

[10] R. C. Pinto and J. Rufino (2014), *Towards non-invasive runtime verification of real-time systems*, in Proc. 26th Euromicro Conf. on Real-Time Systems - WIP Session, (Madrid, Spain), pp. 25–28, Euromicro.

[11] T. Reinbacher, M. Fugger, and J. Brauer (2014), *Runtime verification of embedded real-time systems*, Formal Methods in System Design, vol. 24, pp. 203–239.

[12] R. Pellizzoni, P. Meredith, M. Caccamo, and G. Rosu (2008), *Hardware runtime monitoring for dependable cotsbased real-time embedded systems*, in Proceedings of the Real-Time Systems Symposium (RTSS 2008), (Barcelona, Spain), pp. 481–491, IEEE.

[13] A. Kane, O. Chowdhury, A. Datta, and P. Koopman (2015), *A case study on runtime monitoring of an autonomous research vehicle (ARV) system*, in Proc. 15th Int. Conf. on Runtime Verification, vol. 9333 of LNCS, (Vienna, Austria), pp. 102–117, Springer.

[14] T. Reinbacher, K. Y. Rozier, and J. Schumann (2014), *Temporal-logic based runtime observer pairs for system health management of real-time systems*, in Proc. 20th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), vol. 8413 of LNCS, (Grenoble, France), pp. 357–372, Springer.

[15] J. C. Lee, A. S. Gardner, and R. Lysecky (2011), *Hardware observability framework for minimally intrusive online monitoring of embedded systems*, in Proc. 18th Int. Conf. on Engineering of Computer Based Systems, (Las Vegas, USA), pp. 52–60, IEEE.

[16] G. Callow, G. Watson, and R. Kalawsky (2010), *System modelling for run-time verification and validation of autonomous systems*, in Proc. 5th Int. Conf. on System of Systems Engineering, (Loughborough, UK).

[17] S. Cotard, S. Faucou, J.-L. Bechennec, A. Queudet, and Y. Trinquet (2012), *A data flow monitoring service based on runtime verification for AUTOSAR*, in Proceedings of the 14th Int. Conf. on High Performance Computing and Communications, (Liverpol, UK), IEEE.

[18] A. Kane (2015), *Runtime Monitoring for Safety-Critical Embedded Systems*, PhD thesis, Carnegie Mellon University, USA.

[19] S. Jaksic, M. Leucker, D. Li, and V. Stolz (2017), *CO-EMS open traces from the industry*, in Proc. of Int. Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RVCuBES 2017), vol. 3, (Seatle, USA.).

[20] G. Reger and K. Havelund (2016), *"What is a trace? a runtime verification perspective*, in Proc. of 7th Int. ISoLA 2016 - Leveraging Applications of Formal Methods, Verification and Validation (T. Margaria and B. Steffen, eds.), vol. LNCS 9953, (Corfu, Greece).

[21] J. Reinders (2013), *Intel Processor Tracing*, Intel Corporation, Sept. 2013.

[22] Intel (2013), *Intel Architecture Instruction Set Extensions Programming Reference*, 319433-017 ed.

[23] W. Orme (2008), *Debug and trace for multicore SoCs*, ARM White paper.

[24] ARM 2013, *CoreSight technical introduction: a quick-start for designers*, ARM White paper EPM-039795.

[25] ARM (2013),*ARM CoreSight Architecture Specification*, 2.0 ed.

[26] Aeroflex Gaisler A.B. (2014), *GRLIB IP Library User's Manual.*

[27] SPARC International Inc. (1992), *The SPARC Architecture Manual.*

[28] ARM Limited (1999), *AMBATM Specification.*

[29] The RTEMS Project (2017), *RTEMS: Real-Time Executive for Multiprocessor Systems - User Manual*, release 5.0.0 (master) ed. http://www.rtems.org.

Verification of Scheduling Properties Based on Execution Traces

Valérie-Anne Nicolas, Mounir Lallali, Stéphane Rubini, Frank Singhoff
Lab-STICC UMR 6285, Université de Bretagne Occidentale, UBL, Av. Le Gorgeu, 29200 Brest, France; email: {surname.name}@univ-brest.fr

Abstract

Despite the use of scheduling analysis when designing hard real-time systems, some erroneous temporal behaviors may still occur at runtime. Monitoring the execution of the system during runtime is a way to spot faulty behaviors. We focus on inline and embedded monitoring for the verification of general but essential temporal properties: scheduling properties.
This paper presents an approach for the temporal scheduling properties verification part of monitoring. The proposed algorithm has been evaluated on a benchmark, detecting missed deadlines, priority inversions, deadlocks and locked resources, in keeping with scheduling analysis and simulation results.

Keywords: *monitoring, trace analysis, scheduling property verification, real-time system.*

1 Introduction

Real-time system correctness depends on its logical and temporal correctness [1]. In the context of hard real-time systems, the system temporal constraints are essential and have to be met. The real-time scheduling theory provides methods and tools to describe, simulate such systems, and to verify temporal properties during the design stage. Despite the large amount of work in design stage modeling and verification of hard real-time systems, enhancing the overall system quality, some erroneous temporal behaviors may still occur at runtime.

Monitoring the execution of the system is thus mandatory to guarantee its integrity during its whole execution [2]. Moreover, to deal with hard timing constraints, the overall monitoring tool should be embedded into the system, while still being as non-intrusive as possible, and sufficiently efficient to adapt the system behavior, when needed, in a restricted delay. A monitoring tool observes the monitored system and builds a trace that constitutes a model of the real execution of the system. There is a number of trace models, depending on the kind of trace events, and in general closely related to the monitor tool, the type of monitored application, the intended properties or behaviors to observe. A processing module deals with the trace to obtain supervision information, for example compliance with specific temporal behaviors. A decision module may take action in line with supervision information, like ending the system execution for the most critical cases.

This paper presents an instance of a processing module applying temporal scheduling properties verification on execution traces as illustrated on Figure 1. We situate within the framework of the Cheddar scheduling analysis project and its associated Cheddar toolset including a scheduling analysis tool, a simulation tool [3], and a simplified architecture description language (called Cheddar ADL [4]). One of the output files when applying the simulation tool is the simulation trace file. This trace is the sequence of time-stamped events generated during simulation. The hereafter proposed verification module is based on the same system and trace models as in the Cheddar tool and the monitor introduced in [2].

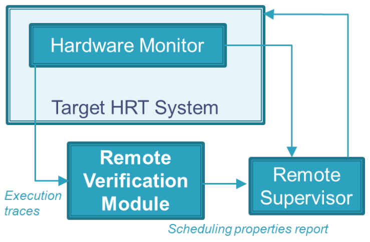


Figure 1: Verification module integration with the monitor

The paper is organized as follows: Cheddar system model, Cheddar trace model, and aimed temporal scheduling properties are described in Section 2. Next, we present the chosen approach to check temporal scheduling properties on execution traces in Section 3. In Section 4, the behavior of the proposed algorithm is illustrated on several simple examples. Then, related work is presented in Section 5. We finally conclude and point out upcoming improvements in Section 6.

2 System Model, Trace Model and Scheduling Properties

Figure 2 shows the verification module software architecture. The targeted systems for runtime monitoring are hard real-time systems on uniprocessor execution platform. The system model exported from the Cheddar ADL system model describes a system by a set of XML markup elements. Markup elements are dedicated to system hardware description (processors, cores, address spaces, scheduling parameters, etc.) and system software description (tasks, resources, resource sharing protocols, etc.) [4]. As an example, tasks are periodic and mostly characterized by their period, capacity, deadline, start time and priority. Resources are mainly characterized by their critical sections and the sharing protocol defining the access rules to the resource if it is shared by several tasks. The critical section for a resource *R* is the set of critical sections for the tasks sharing *R*. The critical section for a task *T*, using the shared resource *R*, is the time interval [*begin_time*, *end_time*] during which *T* uses *R*.

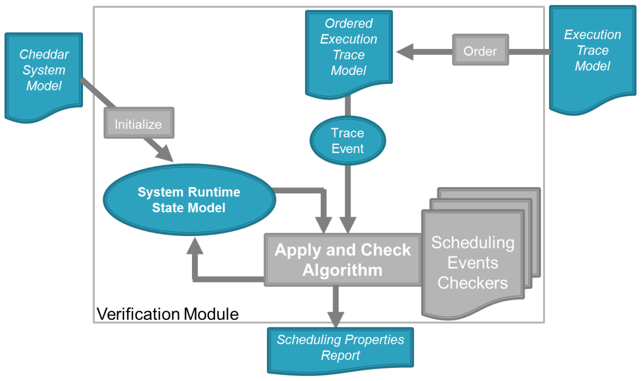

Cheddar System Model
Initialize
Ordered Execution Trace Model
Order
Execution Trace Model
Trace Event
System Runtime State Model
Apply and Check Algorithm
Scheduling Events Checkers
Verification Module
Scheduling Properties Report

Figure 2: Software architecture of the verification module

The XML trace model produced by the Cheddar simulator or the monitor describes a system execution trace by a finite sequence of markup elements for time-stamped events. The types of events, numbering seven, come from the scheduling theory and describe the task states from the scheduling point of view. Events at *time i* for a task *T* (and resource *R*) are:

Task_Activation(i,T)	event sent out each time *i* where a task *T* is activated (ready to run)
Start_of_Task_Capacity(i,T)	event when *T* actually starts running at time *i*
Running_Task(i,T, T current_priority)	event when *T* runs at time *i* (with its priority that may change due to dynamic scheduling or resource sharing protocols)
Allocate_Resource(i,T,R)	event when a resource *R* is allocated to task *T* at time *i*
Wait_for_Resource(i,T,R)	event when a task *T* asks for an already used resource *R* at time *i*
Release_Resource(i,T,R)	event when a resource *R* is released by task *T* at time *i*
End_of_Task_Capacity(i,T)	event when a task *T* finishes its execution at time *i*

An extract of an XML execution trace model is presented in Figure 3 (in Section 3).

From the verification perspective, we are interested in scheduling properties of execution traces, numbering eight. For any given trace *Exe*, we focus on: *P_priority_inversion(Exe)*, *P_deadlock(Exe)*, *P_activation(Exe)*, *P_capacity(Exe)*, *P_deadline(Exe)*, *P_allocate(Exe)*, *P_unlock(Exe)* and *P_wait(Exe)*. The properties *P_deadlock* and *P_priority_inversion* characterize the absence of the corresponding scheduling theory usual concepts. In the simplest case and with a preemptive fixed priority scheduler, two tasks *T1* and *T2* are in deadlock if *T1* locks a resource *R1*, *T2* locks a resource *R2*, and *T1* waits for *R2* while *T2* waits for *R1*. Both tasks prevent each other from accessing the shared resources *R1* and *R2* and therefore are blocked, missing their deadlines. Let see now an example of scheduling when a priority inversion occurs. A priority inversion occurs when two tasks *T1* (a low priority) and *T2* (a high priority) share a resource *R*, a third medium priority task *T3* uses no resource. *T1* begins and owns *R*, then *T2* is activated and preempts *T1*, *T2* later blocks waiting for *R* (still locked by *T1*). *T1* resumes its execution and *T3* is activated before *T1* has released *R*. *T1* is preempted by *T3*. At that point, *T3* (medium priority) can run and thus blocks *T2* (high priority), through *T1*, even though they share no resource.

We now define the other properties investigated in this paper.

P_activation(Exe)	holds true if for each system task, *Task_Activation* events occur at the accurate times (periodically from start time), with no missing or extra *Task_Activation* events in the whole trace *Exe*.
P_capacity(Exe)	holds true if each task job in the trace *Exe* runs exactly for the duration of its capacity.
P_deadline(Exe)	holds true if all task jobs in the trace *Exe* meet their deadlines.
P_allocate(Exe)	holds true if for each *Allocate_Resource(i,T,R)* event in the trace *Exe*, *R* is really needed by *T* at time *i*, *R* is free at time *i* and *i* is the required time for this event.
P_unlock(Exe)	holds true if for each system task in the trace *Exe*, owned resources are released at the required time, and in any case before deadline.
P_wait(Exe)	holds true if for each *Wait_for_Resource(i,T,R)* event in the trace *Exe*, *R* is really needed by *T* at time *i*, *R* is not free at time *i* and *i* is the event required time.

Brought together, all these properties give a fairly complete overview of the expected scheduling behavior of the system.

In the next Section we describe the algorithm for checking these properties, based on the system and trace models presented above.

3 Verification of Scheduling Properties on Execution Traces

The final objective of the verification module is to be embedded into the real-time system and run inline during the system execution. Its execution speed has thus to be compatible with that of the system. Another constraint, even if it is related, is that the monitored real-time systems may have non finite executions, or finite executions but with a great number of events. Therefore, during execution, the verification module does not take as input the whole trace, but a finite fixed size slice of it, using a transition buffer filled by the hardware part of the monitor. The direct induced impact is that the verification module execution time on one slice must be lower than the system execution time corresponding to the next trace slice, otherwise some trace events may be lost. For these reasons, the general frame of our verification algorithm is a one and only one pass through the trace.

As shown on the example of Figure 3 (which is a limited extract of events from a trace for conciseness), trace events are not fully ordered. This is especially the case for *Task_Activation* events. The *Task_Activation* event for a task *T* job is computed at the end of the previous task *T* job and immediately sent out stamped with the time of activation of the future task *T* job. An instance of that is the *Task_Activation* event at time *2* occurring in the trace before events stamped with time *0* or *1*. One may also note that several events may appear at the same time. It is quite common to find at the same time a *Task_Activation* event, a *Start_of_Task_Capacity* event and a first *Running_Task* event for the same task as illustrated by the example at time *0*. The events at a same time may also concern different tasks, as shown at time *3* with a *Wait_for_Resource* event for a first task and a *Release_Resource* event for a second task. There is a number of such possible combinations. Sorting the trace (according to time growing order) is thus imperative to allow to check the properties in a single pass through the trace. To order same time events, we define an order relation *event_order* on events, well suited to the kind of checked properties. For same time events, the order relation *event_order* states that:

$$\boxed{\begin{array}{l} \mathit{End\_of\_Task\_Capacity} \; < \; \mathit{Task\_Activation} \; < \\ \qquad\qquad\qquad \mathit{Start\_of\_Task\_Capacity} \; < \; \mathit{Running\_Task} \\ \wedge \; \mathit{Running\_Task} \; < \; \mathit{Allocate\_Resource} \\ \wedge \; \mathit{Allocate\_Resource} \; = \; \mathit{Wait\_Resource} \; = \; \mathit{Release\_Resource} \end{array}}$$

meaning that, for example, a *End_of_Task_Capacity* event is considered as precedent a *Task_Activation* event even if they occur at the same time in the trace. As stated in Section 2, we target systems on uniprocessor execution platform and thus deal with one single execution trace on the processor. In that framework, the order relation *event_order* is compliant with the trace semantics. Actually, if a task job ends reaching its deadline (a *End_of_Task_Capacity(i,T)* event then follows the last *Running_Task(i-1,T,prio)* event), the task next job will be activated at the same time *i*, and possibly started and first runned also at the same time. On the contrary, by construction, the trace can not exhibit a *Task_Activation* (or *Start_of_Task_Capacity* or *Running_Task*) event and a *End_of_Task_Capacity* event at the same time for a same task job. Regarding resources, task resource allocation (or wait for resource) is first processed at the beginning of the first time unit where the resource is used by the task, whereas resource release is done at the end of the last using time unit. The same time resource related events can not be ordered in the absolute. Each pattern is specific, depending on the real use of resources by tasks. The order relation *event_order* states that the three resource related events are equal, which finally means that the order of these events in the initial trace is preserved.

We now describe the proposed algorithm for verifying scheduling properties in one pass from the time and *event_order* sorted trace. The different points where the properties are checked are depicted in the simplified outline of the algorithm presented in Figure 4. The algorithm is based on a representation of the system state at runtime (including task and resource states), and starting from an inactive initial state (built from the system model), simulates the execution represented by the trace, event by event. At the same time, and depending on the properties to verify, some

```
<event_table>
 <mono_core_processor id="id_2">
  <scheduling_result>
   <result>
    <time_unit>0 </time_unit>
    <time_unit_event>
     <type_of_event>TASK_ACTIVATION</type_of_event>
     <activation_task ref="id_4"> </activation_task>
    </time_unit_event>
    <time_unit>2 </time_unit>
    <time_unit_event>
     <type_of_event>TASK_ACTIVATION</type_of_event>
     <activation_task ref="id_5"> </activation_task>
    </time_unit_event>
    <time_unit>0 </time_unit>
    <time_unit_event>
     <type_of_event>START_OF_TASK_CAPACITY</...>
     <start_task ref="id_4"> </start_task>
    </time_unit_event>
    <time_unit>0 </time_unit>
    <time_unit_event>
     <type_of_event>RUNNING\_TASK</type_of_event>
     <running_task ref="id_4"> </running_task>
     <current_priority>89</current_priority>
    </time_unit_event>
    <time_unit>1 </time_unit>
    <time_unit_event>
     <type_of_event>ALLOCATE_RESOURCE</...>
     <allocate_task ref="id_4"> </allocate_task>
     <allocate_resource ref="id_26"> </allocate_...>
    </time_unit_event>
    <time_unit>1 </time_unit>
    <time_unit_event>
     <type_of_event>RUNNING_TASK</type_of_event>
     <running_task ref="id_4"> </running_task>
     <current_priority>89</current_priority>
    </time_unit_event>
    <time_unit>2 </time_unit>
    <time_unit\_event>
     <type_of_event>START_OF_TASK_CAPACITY</...>
     <start_task ref="id_5"> </start_task>
    </time_unit_event>
    <time_unit>2 </time_unit>
    <time_unit_event>
     <type_of_event>RUNNING_TASK</type_of_event>
     <running_core>core1</running_core>
     <running_task ref="id_5"> </running_task>
     <current_priority>90</current_priority>
    </time_unit_event>
    <time_unit>3 </time_unit>
    <time_unit_event>
     <type_of_event>WAIT_FOR_RESOURCE</...>
     <wait_for_resource_task ref="id_5"> </...>
     <wait_for_resource ref="id_27"> </wait_...>
    </time_unit_event>
    <time_unit>3 </time_unit>
    <time_unit_event>
     <type_of_event>RELEASE_RESOURCE</...>
     <release_task ref="id_4"> </release_task>
     <release_resource ref="id_26"> </release_...>
    </time_unit_event>
    <time_unit>9 </time_unit>
    <time_unit_event>
     <type_of_event>END_OF_TASK_CAPACITY</...>
     <end_task ref="id_4"> </end_task>
    </time_unit_event>
   </result>
  </scheduling_result>
 <mono_core_processor id="id_2">
</event_table>
```

Figure 3: Extract of an XML execution trace model

checks are done on specific event occurrences and some others periodically at the end of each same time sequence of events. Periodic checks concern the tasks reaching the end of their period, and are needed to cope with possible missing events in the trace, such as missing *Task_Activation* events (thus contributing to *P_activation*) or *End_of_Task_Capacity* events. It also allows to complete the detection of undue locked resources (*P_unlock*), or task missed deadline detection (*P_deadline*). Otherwise, when dealing with a specific

```
Algorithm: Apply&Check (system_runtime_state S, trace T)
foreach event E of trace T do
    state_update_with_event(S,E);
    switch E do
        case Task_Activation do
            P_activation_TActivEvt_Check(S,E);
            P_deadline_TActivEvt_Check(S,E);
        case Start_of_Task_Capacity do
            Start_of_Task_Capacity event error detection;
        case Running_Task do
            Running_Task event error detection;
            P_capacity_RunTaskEvt_Check(S,E);
            P_deadline_RunTaskEvt_Check(S,E);
            P_priority_inversion_RunTaskEvt_Check(S,E);
        case End_of_Task_Capacity do
            End_of_Task_Capacity event error detection;
            P_capacity_EndTaskCapaEvt_Check(S,E);
            P_deadline_EndTaskCapaEvt_Check(S,E);
            P_unlock_EndTaskCapaEvt_Check(S,E);
        case Allocate_Resource do
            P_allocate_AllocResEvt_Check(S,E);
        case Release_Resource do
            Release_Resource event error detection;
        case Wait_for_Resource do
            P_wait_WaitResEvt_Check(S,E);
            P_deadline_WaitResEvt_Check(S,E);
            P_deadlock_WaitResEvt_Check(S,E);
    end
end
Periodic_P_activation_Check(S);
Periodic_P_unlock_Check(S);
Periodic_P_deadline_Check(S);
End
```

Figure 4: Apply&Check Algorithm

event, the algorithm checks that no property is violated by this event by calling the procedures associated to the event type adequate properties. These procedures are thus named *PropertyName_EventType_Check(S,E)*, meaning that they check that the event *E* of type *EventType* does not violate the property *PropertyName* in the system runtime state *S*. A specific type event has an impact on only some of the eight studied properties. Thus, only the procedures associated to the potentially impacted properties for the considered type of event are called. For example, a *Allocate_Resource* event may solely affect the *P_allocate* property whereas a *Task_Activation* event may affect the *P_activation* and *P_deadline* properties, and a *Wait_for_Resource* event the *P_wait*, *P_deadline* and *P_deadlock* properties. To give a more precise idea of the content of the checking procedures, here are some details about the *P_activation_TActivEvt_Check* and *P_deadline_TActivEvt_Check* procedures that are called when processing a *Task_Activation* event.

P_activation_TActivEvt_Check(S,E) :

- if the event *E* is a task first activation: checks that the event timestamp is not too late or too early,
- else checks that the previous task job activation event is not missing and that there is not extra activation event for the task in the interval.

P_deadline_TActivEvt_Check(S,E) :
checks that the previous task job did not miss its deadline.

The algorithm has been implemented in C in order to fit with the monitoring constraints: embedded into the system and efficiency.

In the next Section, the behavior of the algorithm is illustrated on several simple trace examples.

4 Evaluation of the Verification Module

The algorithm described in Section 3 has been evaluated on a benchmark of nine system and trace examples. This benchmark mainly comes from a Cheddar tutorial [5]. Each example is made of a system model and a trace model resulting from the Cheddar simulation tool. For all the examples, the verification algorithm results are compliant with Cheddar scheduling analysis and simulation tools. Among the nine examples, four exhibit erroneous behaviors (missed deadlines, deadlocks, priority inversions or locked resources).

For brevity, we here only present two mistaken examples whose system and trace models can be accessed online [6]. For each of them, we assume a preemptive fixed priority scheduling policy and priorities are assigned according to Rate Monotonic. In the first example, a system with three periodic tasks, synchronous and with deadlines on request is considered.

Task	Period	Deadline	Capacity	Start time
T1	6	6	2	0
T2	8	8	2	0
T3	12	12	5	0

Tasks *T1* and *T3* share a resource S with mutual exclusion access: *T3* needs S during all its capacity, *T1* needs S during the 2nd unit of time of its capacity only. There is no specific priority inheritance protocol, blocked tasks are thus stored in a FIFO queue. The trace contains 75 events and expresses the system behavior over its feasability interval, that is from *0* to the *tasks periods Least Common Multiple (LCM)* as the tasks are synchronous [7], thus from time *0* to time *24*. When executing our verification algorithm, a priority inversion between tasks *T1* and *T2* is detected at times *8* and *9*, and a missed deadline for the task *T1* is detected at times *12* and *13*. Changing the sharing resource protocol by PIP (Priority Inheritance Protocol) leads to a correct behavior of the system, attested by the execution of the verification algorithm which finds no more errors.

The second example is a system with two asynchronous periodic tasks and one shared resource.

Task	Period	Deadline	Capacity	Start time
T1	20	20	10	0
T2	10	10	4	1

Tasks *T1* and *T2* share a resource *R1* with mutual exclusion access: *T1* needs *R1* from the the 1st unit of time of its capacity up to the 4th (included), and from the 3rd unit of time of its capacity up to the 6th (included). *T2* needs *R1* from the 1st unit of time of its capacity up to the 2nd (included). There is no specific priority inheritance protocol. Here, tasks are not synchronous and the feasability interval is defined from *0* to the *maximum of tasks start times + 2 * LCM(tasks periods)* [7]. The trace contains 85 events and expresses the system behavior over its feasability interval, that is from time *0* to time *41*. When executing our verification algorithm, a deadlock on *R1* for task *T1* is detected at all times from *2*, a missed deadline for the task *T2* is detected at all times from *11* (while waiting for *R1*), an unlock error is detected on *R1* for *T1* at time *19* and *39*, a missed deadline for the task *T1* is detected at all times from *20* (while waiting for *R1*).

On this benchmark, results confirm that the whole set of considered properties give a fairly complete overview of the scheduling behavior of the system, similar to scheduling analysis and simulation results.

5 Related Work

Several works have been proposed for runtime verification/-monitoring of timed properties based on execution traces. [8] proposes a runtime verification framework for SoC (Systems on Chip) model. This framework allows the verification of temporal properties described in PSL (Property Specification Language), and the analysis of verification results. The authors of [9] present a software architecture based on Logic-Labeled Finite-State Machine (LLFSM) and regular expressions to perform runtime monitoring and verification of robotic system behaviors. [10] proposes a runtime verification approach for timed systems based on executable models. They define an on-the-fly conformance relation (between implementations and specifications) used for runtime verification, and they suggest an on-the-fly matching for timed traces. The proposed method has been implemented in an open-source toolkit which has been experimented on the verification of some units of different industrial microprocessors. [11] presents a predictive runtime verification framework for systems with timing requirements. Unlike the previous approaches, this predictive verification is related to a system which is not monitored as a black-box (some information about the system behavior is known).

Previous works propose their own verification framework and/or architecture that are not integrated as a part of the real-time system monitoring. In addition, these works deal with general temporal properties. In our case, we focus on scheduling properties verification for inline and embedded monitoring, and we aim at using our verification module as a part of an inline embedded health monitor.

6 Conclusion

In this paper, an approach for the verification of scheduling properties on uniprocessor hard real-time system execution traces has been presented. This verification module has been implemented in C and evaluated on a simple benchmark. Testing showed that verification module results were compliant with Cheddar scheduling analysis and simulation results, thus strengthening confidence in the algorithm pertinence and confirming that the set of considered properties gives an accurate overview of the expected scheduling behavior of the system. Currently, the verification module deals with one slice of execution trace. Next improvement is to enchain the processing of several execution trace slices.
After what the objective is to use this verification module as a part of an inline embedded health monitor [2]. Further work is needed to evaluate the verification module on more consistent and realistic examples, so as to assess its efficiency when embedded into a real-time system.

Acknowledgments This work and Cheddar are supported by Brest Métropole, Ellidiss Technologies, CR de Bretagne, CG du Finistère and Campus France PESSOA programs number 27380SA and 37932TF.

References

[1] C. L. Liu and J. W. Layland (1973), *Scheduling algorithms for multiprogramming in a hard-real-time environment*, Journal of the ACM (JACM), vol. 20, no. 1, pp. 46–61.

[2] S. Rubini, V.-A. Nicolas, F. Singhoff, and J. Rufino (2018), *A real-time system monitoring driven by scheduling analysis*, RUME'18 Workshop.

[3] F. Singhoff, J. Legrand, L. Nana, and L. Marcé (2004), *Cheddar: a flexible real-time scheduling framework*, ACM SIGAda Ada Letters, vol. 24, pp. 1-8, ACM Press, New York, USA.

[4] C. Fotsing, F. Singhoff, A. Plantec, V. Gaudel, S. Rubini, S. Li, H. N. Tran, L. Lemarchand, P. Dissaux, and J. Legrand (2014), *Cheddar architecture description language*, Lab-STICC technical report.

[5] F. Singhoff (2015), *Tutorial about cheddar : an example of real-time scheduling analysis with cheddar*, Lab-STICC technical report.

[6] http://beru.univ-brest.fr/svn/CHEDDAR/trunk/docs/publications/nicolas18.

[7] J. Y.-T. Leung and M. Merrill (1980), *A note on preemptive scheduling of periodic, real-time tasks*, Information Processing Letters, vol. 11, no. 3, pp. 115 – 118.

[8] L. Pierre and M. Chabot (2017), *Assertion-based verification for soc models and identification of key events*, 2017 Euromicro Conference on Digital System Design (DSD), pp. 54–61.

[9] V. Estivill-Castro and R. Hexel (2016), *Run-time verification of regularly expressed behavioral properties in robotic systems with logic-labeled finite state machines*, 2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), pp. 281–288.

[10] M. M. Chupilko and A. S. Kamkin (2013), *Runtime verification based on executable models: On-the-fly matching of timed traces*, Proceedings Eighth Workshop on Model-Based Testing, Rome, Italy, 17th March 2013, pp. 67–81.

[11] S. Pinisetty, T. Jron, S. Tripakis, Y. Falcone, H. Marchand, and V. Preoteasa (2017), *Predictive runtime verification of timed properties*, J. Syst. Softw., vol. 132, pp. 353–365.

Non-intrusive Runtime Verification within a System-on-Chip*

José Rufino, António Casimiro
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal; email: jmrufino@ciencias.ulisboa.pt

Felix Dino Lange, Martin Leucker, Torben Scheffel, Malte Schmitz, Daniel Thoma
Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Lübeck, Germany; email: lange,leucker,scheffel,schmitz,thoma@isp.uni-luebeck.de

Abstract

This paper describes how to enrich a System-on-Chip (SoC) design by flexible monitoring capabilities allowing to analyze the system's execution for ensuring safety requirements. To this end, a general SoC architecture is described enriched by observation means. Moreover, it is described how verification properties expressed in a temporal stream-based specification language can be translated into a monitor expressed in a hardware description language (Verilog) checking the underlying property. Finally, the link between the SoC and the monitoring unit is explained. Overall, a self-observing system is obtained that works coherently with the SoC.

Keywords: SoC runtime verification

1 Introduction and Motivation

Autonomous vehicles are paving their way into application domains as diverse as: terrestrial, aerospace, maritime and submarine. They include a System-on-Chip (SoC), hosting an on-board computing system, to control the vehicle and ensure the fulfillment of its mission.

In general, those control functions are extremely complex, with strict real-time requirements. Interaction with the environment and operation in harsh or uncertain contexts are potential sources for lack of determinism. In any case, the correctness of the overall system is paramount, and safety should be ensured at all times.

Runtime Verification (RV) [1, 2] assumes herein great relevance, since it adds an extra layer of protection, assessing the system against a previously defined specification, checking whether timing and safety properties are satisfied or violated. Most of the current RV techniques require the modification of the application source code. Although software-based instrumentation is reasonable for larger systems, the requirements that characterise these vehicular systems may pose an unsurpassed challenge for runtime verification in such kind of systems. Other techniques, such as system and/or function call interception, are also not free from intrusiveness.

In this context, the concept of Hardware-based Observability, a non-intrusive observation and runtime verification technique, assume particular relevance. More precisely, the underlying idea is that a safety-critical system should be enriched by observation and analysis/monitoring techniques directly on the core system itself. Thus, they should become a part of the SoC. The direct combination allows perfect observability of the functional system. The allocation of hardware resource for the analysis further ensures that the monitoring does not affect the execution of the functional system.

While SoC are traditionally specified in hardware description languages like VHDL [3] or Verilog [4], the specification of verification properties should ideally be performed in some high-level domain specific language. Recently, the authors hosted at Lübeck introduced the temporal stream-based specification language TeSSLa [5] which is especially designed for specifying correct program executions. In this paper, we describe how TeSSLa specification can be translated into a hardware description language and integrated into a SoC for performing basic verification tasks. Overall, we obtain a self-observing system that works coherently with the SoC.

The paper is organized as follows. Section 2 presents hardware-based observability monitors. Section 3 focuses on an introduction to TeSSLa while Section 4 discusses its translation into a Verilog format. Section 5 evaluates the work done. Section 6 describes the related work and Section 7 presents some concluding remarks and future research directions.

2 Non-Intrusive Observation and Runtime Verification

The classical approach to runtime verification implies the instrumentation of the functional system software components: small pieces of software, acting as observers, are added to

*This work was partially supported by FCT, through funding of LASIGE Research Unit, ref. UID/CEC/00408/2013, and by FCT/CAMPUS FRANCE (PHC PESSOA programme), through the transnational cooperation project 3732 (PT) / 37932TF (FR), Non-intrusive Observation and RunTime verification of cyber-pHysical systems (NORTH). This work integrates the activities of COST Action IC1402 - Runtime Verification beyond Monitoring (ARVI), supported by COST (European Cooperation in Science and Technology). This work was partially supported by the BMBF project ARAMIS II with funding ID 01 IS 16025 and EU H2020, through project 732016, COEMS (Continuous Observation of Embedded Multicore Elements).

assess their state in runtime. Software-based instrumentation inherently disturbs the system, namely with respect to timing properties, which are crucial to system design.

2.1 Hardware-based Observability

The demand for non-intrusive observability justifies, per se, the interest in hardware-based methods, powered by: the usage of reconfigurable logic, supported on FPGA special purpose observers [6, 7, 8]; the raw availability of integrated observation resources [9, 10]. By nature, hardware-based system observation is completely non-intrusive and can be made, by design, extremely effective.

The architecture described in Figure 1 describes the functional system platform, implemented as a SoC architecture and how runtime observation and monitoring features can be integrated non-intrusively, meaning execution of runtime verification actions does not disturb the execution of the functional system software components. Probing the processor-cache interfaces should allow an higher accuracy in the observation of software components execution.

2.2 Observer Entity

The Observer Entity defined by the architecture of Figure 2 aims to support the non-intrusive observation and runtime verification of an associated functional system, therefore enabling the verification in runtime that its properties are being fulfilled and that no design assumption is being violated.

The Observer Entity is plugged to the platform where the functional system software components execute, and comprises the hardware modules of Figure 2: Bus Interfaces, capturing all physical bus activity, such as bus transfers or interrupt vectors; Management Interface, enabling observer entity configuration; Configuration, storing the dynamic set of events; the System Observer itself, detecting events of interest; Monitor, which detects possible violations of the specified system behaviour; Time Base, which allows to time stamp the events of interest.

2.3 System Observing Mechanisms

The System Observer collects, in runtime, from the functional system bus interfaces, all the addressing/data information to detect events of interest set by configuration, performed statically (offline) or dynamically, while the system is executing.

When an event of interest (e.g., the fetch of a specific instruction or a read/write access to a given variable in the memory) is detected, it is timestamped with the instant of occurrence, as obtained from the Time Base module, and supplied to every downstream block awaiting for that event. A unique identifier (obsID) is assigned to each observed event, being an event composed by the tuple:

$$evt_{obsID} =< a_{obs}, v_{obs}, t_{obs} >$$

where: a_{obs} is the address observed from the functional system bus interface that matches a given event specification; v_{obs}, the corresponding observed value (e.g., instruction coding or data value); t_{obs}, is the attached timestamp.

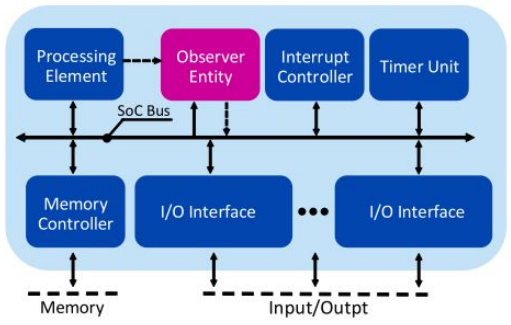


Figure 1: Generic SoC architecture and Observer Entity

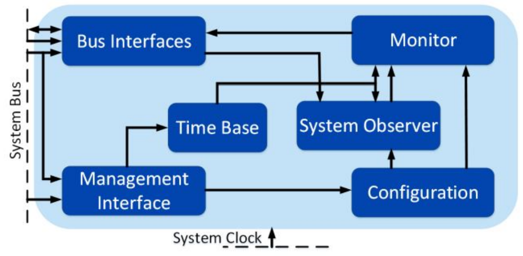


Figure 2: Observer Entity architecture

2.4 Monitoring Mechanisms

A divide and conquer strategy is used in the definition and design of a minimal set of hardware-based essential blocks for the synthesis of runtime verification mechanisms. A set of basic monitors, encompassing essential runtime verification actions, in both value and time domains, is detailed in [11]. These monitors can be instantiated as required. Additional blocks (selectors, transformers and past-time event registers) complement and enlarge the functionality provided by the basic monitors. The right combination of these building blocks should be able to provide the necessary and sufficient mechanisms for the runtime verification of any functional system.

3 An introduction to TeSSLa

TeSSLa [5] is a temporal stream-based specification language that is designed for monitoring real-time signals and has already been used to build monitors for Runtime Verification [12]. TeSSLa reasons over asynchronous input streams and provides a rich data domain (Boolean, integer, real). Monitors specified in TeSSLa can observe events, that were emitted with different speeds and with different delays. TeSSLa supports signals and event streams. An event stream is only allowed to be defined for a finite number of timestamps in a finite interval, while a signal stream defines a value for every point in time.

The basic concept of TeSSLa is deriving internal or output streams by applying functions to already existing streams. A stream can be defined declaratively as can be seen in the following example of a TeSSLa specification:

```
def maximum := max(x1, x2)
def max(a,b)  := if a > b then a else b
```

The specification contains two input streams $x1$ and $x2$ and creates a new stream $maximum$ which always contains the larger value of $x1$ and $x2$. Note that it is possible to define macros (i.e. $max(a, b)$) that can be used to define more sophisticated properties.

A complete list of all functions can be found in [5]. For example can basic arithmetic function (like the comparison of two numbers) be lifted. The lifted function is able to reason over streams instead of i.e. integers. For timing properties it is possible to generate a stream of timestamps corresponding to the current value of another stream:

```
def timeOfx := time(x)
```

TeSSLa is useful for the approach of hardware-based monitoring because it is especially designed for monitoring streams and can be directly translated into hardware descriptions as is explained in the following.

4 Translation into Verilog

Figure 3 shows the approach of Non-intrusive Runtime Verification within a System-on-Chip. The TeSSLa compiler translates the TeSSLa specification into a dependency graph. The dependency graph contains the necessary information to generate Verilog code, which is used to synthesize the monitor in FPGA hardware. There are five different operators that have to be considered for the translation. Every operator can directly be implemented as a node in the dependency graph of a TeSSLa specification and the nodes can be connected via message parsing. To show that this direct translation is generally possible, two cases have to be considered.

Without recursion: If there are no recursions in a TeSSLa specification, its dependency graph is known to be a directed, acyclic graph [12]. To make sure that there is always at least one node that is able to write, extra events, called progress events, are introduced. Their purpose is to inform nodes downstream about the absence of events. From that follows a constant event throughput at all times. A formal proof can be found in [13].

With recursion: There are two operators in TeSSLa that have recursive behaviour. The $last()$ operator returns a stream with the last value of another stream based on a trigger signal. The $delay()$ operator delays a stream by a given amount of time and can be reset by a signal stream. Because both the trigger signal stream and the reset signal stream cannot be recursive, it is guaranteed that a progress exists at all times.

This shows that every TeSSLa specification produces the same output independently of timed reordering and it is therefore possible to translate into evaluation engines implemented in Verilog.

5 Use Case Integration

A use case in the domain of aerospace is the observation of a navigation system of a satellite. The execution time of different tasks with different priorities has to be observed, because sometimes the execution takes longer than expected. If a task exceeds the expected execution time, it has to be canceled so other task can be executed in time. However, if the same task is failing three times in a row, this is considered an error, because the calculation of the trajectory of the satellite needs the result of this task at least every third execution.

In order to monitor this behavior, we need to check the runtime of the task, compare the timestamps and count the number of failed task executions. The runtime of the task can be gathered by instrumenting the hardware of the Leon processor as described in Section 2 and can be passed as streams of events to a monitor on the FPGA. The two streams contain the events of starting ($call$) and finishing ($return$) the task. The runtime of the tasks can be calculated and compared with TeSSLa:

```
def runtime := on(return,
   time(return) - time(call))
def count_violations :=
   if runtime > threshold
   then resetcount(runtime, false)
   else
   resetcount(runtime, true)
```

Note that the macro $on(x, y)$ assures that the stream $runtime$ is only updated, if a new $return$ event was sent. $resetcount(trigger, reset)$ returns the counted number and is reset every time the second argument is $true$. The full code for the macros is not shown in this paper due to paper size limitations. An error is declared, if the threshold is violated three times in a row:

```
def error :=
  if count_violations > 3 then true
  else false
```

This specification is then translated into a dependency graph by the TeSSLa compiler. The dependency graph can be used to generate a hardware specification as described in section 4. With this setup it is possible to observe the activity of the satellite for an unlimited amount of time and gather information about the runtime violations of certain tasks.

6 Related Work

The application of non-intrusive runtime monitoring to embedded systems has been discussed in [6,14] and, more specifically, in safety critical environments [15]. Configurable minimally intrusive event-based frameworks for dynamic runtime monitoring have been developed [16]. Additionally, the RV concept has been applied to autonomous systems [17] and to diagnose multi-processor SoC [18]. However, to the extent of our knowledge, no previous work has exploited how a TeSSLa specification can be translated into a hardware description language and integrated into a SoC.

7 Conclusion

We propose an approach on how to combine hardware-based non-intrusive observation of a System on Chip with the high-level temporal stream-based specification language TeSSLa. With its easy to read C-style TeSSLa can be used to describe properties much more intuitively than directly in hardware

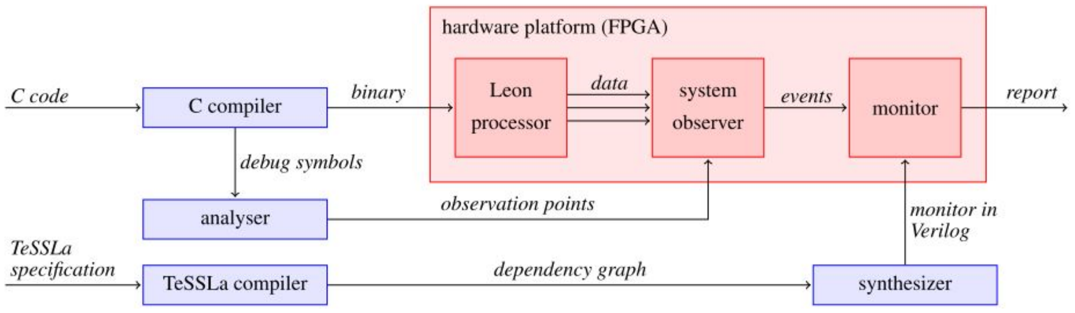


Figure 3: An overview over our approach

description languages. It can be shown that TeSSLa specifications are directly translatable into a hardware description language like Verilog.

Hardware-based observation is especially useful in domains with long observation times. Therefore we introduce the use case of task runtime observation of a satellite navigation system to show a possible application of this approach.

This paper is the first step towards integrating TeSSLa into a SoC. The use case prototype, showing the feasibility of hardware-based observation within a SoC, needs further work, namely with regard to: the exploitation of the monitoring infrastructure [11]; the translation from TeSSLa to a hardware description language; the definition of an effective algorithm for the direct translation from the TeSSLa specification.

References

[1] M. Leucker and C. Schallhart (2009), *A brief account of runtime verification*, The Journal of Logic and Algebric Programming, vol. 78, pp. 293–303.

[2] Y. Falcone, K. Havelund, and G. Reger (2013), *A Tutorial on Runtime VerificationEngineering*, in Dependable Software Systems, vol. 34, pp. 141–175. Marktoberdorf, Germany: IOS Press Ebooks.

[3] IEEE (2018), *1076.1-2017 - IEEE Standard VHDL Analog and Mixed-Signal Extensions*.

[4] IEEE (2018), *1800-2017 - IEEE Standard for SystemVerilog– Unified Hardware Design, Specification, and Verification Language*.

[5] L. Convent, S. Hungerecker, M. Leucker, T. Scheffel, M. Schmitz, and D. Thoma (2018), *TeSSLa: a temporal stream-based specification language*, in International Colloquium on Theoretical Aspects of Computing (ICTAC). Submitted for publication.

[6] C. Watterson and D. Heffernan (2007), *Runtime verification and monitoring of embedded systems*, Software, IET, vol. 1, Oct. 2007.

[7] J. C. Lee, A. S. Gardner, and R. Lysecky (2011), *Hardware observability framework for minimally intrusive online monitoring of embedded systems*, in Proc. 18th Int. Conf. on Engineering of Computer Based Systems, (Las Vegas, USA), pp. 52–60, IEEE.

[8] R. C. Pinto and J. Rufino (2014), *Towards non-invasive runtime verification of real-time systems*, in 26th Euromicro Conf. on Real-Time Systems - WIP Session, (Madrid, Spain), pp. 25–28.

[9] ARM (2013), *ARM CoreSight Architecture Specification*, Cambridge, England, 2.0 ed.

[10] R. Backasch, C. Hochberger, A. Weiss, M. Leucker, and R. Lasslop (2013), *Runtime verification for multicore SoC with high-quality trace data*, ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 18, p. 18.

[11] J. Rufino (2018), *Runtime verification monitors*, tech. rep., Faculdade de Ciências da Universidade de Lisboa, Portugal.

[12] N. Decker, P. Gottschling, C. Hochberger, M. Leucker, T. Scheffel, M. Schmitz, and A. Weiss (2017), *Rapidly adjustable non-intrusive online monitoring for multicore systems*, Brazilian Symposium on Formal Methods, pp. 179–196, Springer.

[13] M. Leucker, C. Sánchez, T. Scheffel, M. Schmitz, and A. Schram (2018), *TeSSLa: Runtime verification of nonsynchronized real-time streams*, in ACM Symp. on Applied Computing (SAC), (Pau, France), ACM.

[14] T. Reinbacher, M. Fugger, and J. Brauer (2014), *Runtime verification of embedded real-time systems*, Formal Methods in System Design, vol. 24, no. 3, pp. 203–239.

[15] A. Kane, O. Chowdhury, A. Datta, and P. Koopman (2015), *A case study on runtime monitoring of an autonomous research vehicle (ARV) system*, in Proc. 15th Int. Conf. on Runtime Verification, (Vienna, Austria).

[16] J. C. Lee and R. Lysecky (2015), *System-level observation framework for non-intrusive runtime monitoring of embedded systems*, ACM Transactions on Design Automation of Electronic Systems, vol. 20, no. 42.

[17] G. Callow, G. Watson, and R. Kalawsky (2010), *System modelling for run-time verification and validation of autonomous systems*, in Proc. 5th Int. Conf. on System of Systems Engineering, (Loughborough, UK).

[18] P. Wagner, T. Wild, and A. Herkersdorf (2017), *DiaSys: Improving SoC insight through on-chip diagnosis*, Journal of Systems Architecture, vol. 75.

Non-intrusive Observation and Runtime Verification of Avionic Systems *

José Rufino
LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal; email: jmrufino@ciencias.ulisboa.pt

Abstract

Unmanned autonomous systems (UAS) avionics call for advanced computing system architectures fulfilling strict size, weight and power consumption (SWaP) requisites. The AIR (ARINC 653 in Space Real-Time Operating System) defines a partitioned environment for the development and execution of aerospace applications, preserving application timing and safety requisites.

This paper intensively explores the potential of nonintrusive runtime verification (NIRV) mechanisms, currently being included in AIR, to the overall improvement of system safety.

Keywords: Avionic and runtime verification

1 Introduction and Motivation

Avionic systems have strict safety and timeliness requirements as well as strong size, weight and power consumption (SWaP) constraints. Modern unmanned autonomous systems (UAS) avionics follow the civil aviation trend of transitioning from federated architectures to Integrated Modular Avionics (IMA) [1] and resort to the use of partitioning.

Partitioning implement the logical separation of applications in criticality domains, that we named partitions, and allow hosting both avionic and payload functions in the same computational infrastructure [2, 3].

However, partitioned architectures in general, and those designed using AIR (ARINC 653 in Space Real-Time Operating System) [4], in particular, tend to have their complexity and may largely benefit of their combination with a runtime verification and monitoring infrastructure [5].

This paper explains how fundamental runtime verification (RV) mechanisms can be combined with advanced time-and space-partitioned (TSP) systems. To reduce the temporal overhead of such mechanisms in the operation of onboard systems an innovative non-intrusive design approach is followed.

*This work was partially supported by FCT, through funding of LASIGE Research Unit, ref. UID/CEC/00408/2013, and by FCT/CAMPUS FRANCE (PHC PESSOA programme), through the transnational cooperation project 3732 (PT) / 37932TF (FR), Non-intrusive Observation and RunTime verification of cyber-pHysical systems (NORTH). This work integrates the activities of COST Action IC1402 - Runtime Verification beyond Monitoring (ARVI), supported by COST (European Cooperation in Science and Technology).

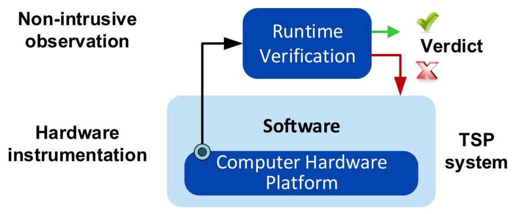


Figure 1: Non-intrusive observer and runtime verification

The paper is organized as follows. Section 2 describes the non-intrusive RV features being introduced while Section 3 presents the AIR architecture for TSP systems. Section 4 describes how to integrate RV mechanisms with the AIR architecture and Section 5 performs their evaluation. Section 6 describes the related work and, finally, Section 7 issues concluding remarks and future research directions.

2 Mechanisms for Non-intrusive Observation and Runtime Verification

Runtime verification obtains and analyses data from the execution of a system to detect and possibly react to behaviours, either satisfying or violating the system specification. Runtime verification implies that small components, which are not part of the functional system, acting as observers, are added to monitor and assess the state of the system in runtime.

The usage of reconfigurable logic supporting versatile platform designs (e.g., soft-processors), as depicted in Figure 1, enables innovative approaches to RV [6]. In the context of TSP systems: the computer hardware platform is instrumented with non-intrusive observers; the runtime verification is secured by an independent hardware module, with no system actions (unless there is an error).

An enhanced AIR architecture uses an AIR Observer and Monitor (AOM) featuring: non-intrusiveness, meaning system operation is not adversely affected and code instrumentation with RV probes is not required; configurable, being able to accommodate different event observations.

The AOM hardware is plugged to the platform where the AIR software components execute, and comprises the modules depicted in Figure 2: bus interfaces, capturing all physical bus activity, such as system bus and cache bus transfers or interrupts; management interface, enabling AOM configuration; configuration, storing the patterns of the events to be detected;

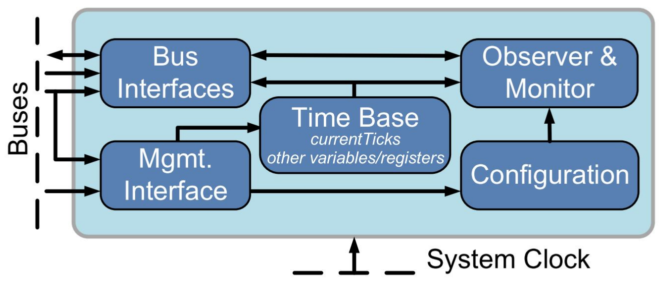


Figure 2: AIR Observer and Monitor architecture

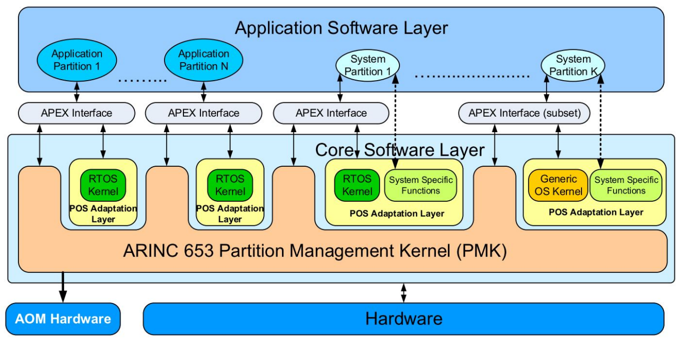


Figure 3: AIR architecture with AOM hardware

observer, detecting events of interest based on the registered configurations and monitor, performing the required runtime verification actions.

Though RV concepts can be applied to both time and space partitioning, this paper is restricted to temporal issues. Thus, it is assumed that a robust time base[1] accounts for, in the AOM hardware (Figure 2), the number of POS-level clock ticks elapsed so far, to which AIR components have access, through the read only current Ticks variable/register (used in Algorithm 1). Other variable/registers may need to be stored within the scope of the AOM hardware.

3 AIR Technology for TSP systems

The AIR design aims at providing high levels of flexibility, hardware- and OS-independence, easy integration and independent component verification, validation and certification [4]. The AIR architecture is depicted in Figure 3.

The AIR Partition Management Kernel (PMK) is a core software layer, enforcing robust TSP properties, together with partition scheduling and dispatching, low-level interrupt management, and interpartition communication support. Robust TSP implies that the execution of functions in one partition does not affect other partitions' timeliness and that separated addressing spaces are assigned to different partitions.

Each partition can host a different OS (the partition operating system, POS), which in turn can be either a real-time operating system (RTOS) or a generic non-real-time one. The AIR POS Adaptation Layer (PAL) encapsulates the POS of each partition, providing an adequate POS-independent interface.

[1]The design and engineering of AIR robust timers is out of the scope of this paper. It will be addressed in a future work.

The Portable Application Executive (APEX) interface [7] provides a standard programming interface derived from the ARINC 653 specification [1], with the possibility of being subsetted and/or adding specific functional extensions for certain partitions [8].

The architecture of Figure 3 also includes the AOM hardware module that we will intensively exploit in our design.

4 Integrating Non-intrusive Observation and Runtime Verification

The integration of RV features in the AIR architecture is, in essence, concerned with the operation of the AIR Partition Scheduler and Dispatcher and uses a dual approach:

- operation enforced in hardware, either totally or with some degree of assistance from software components, being the RV actions performed in software, being this kind of action only seldom used;
- operation achieved through the execution of software components, with RV actions enforced in hardware, the normal operating behaviour.

4.1 Partition scheduling

The original ARINC 653 notion of a single fixed Partition Scheduling Table (PST) [1], defined offline, is limited in terms of timeliness, as well as safety and fault-tolerance control. To address this primary limitation, the AIR design incorporates the notion of mode-based partition schedules, inspired by the optional service defined within the scope of ARINC 653 Part 2 specification [9].

The system can now be configured with multiple PSTs, which may differ in terms of their Major Time Frame (MTF) duration. The different PSTs may specify which partitions are scheduled on each mission phase, and of how much processor time is assigned to them [4], as shown in Figure 4. The system can then switch between these PSTs; a PST switch request is only effectively granted at the end of the ongoing MTF [4].

4.2 Mode-based schedules

The AIR RV architecture uses an hardware-assisted approach for selecting the partition scheduling switch instants, which are programmed at the AOM, whenever a partition is dispatched: the next partition preemption point is inserted in the AOM configuration; when this instant is reached, an AOM's hardware exception triggers the execution of Algorithm 1.

The RV actions of Algorithm 1 check, from the active PST, if the current instant is a partition preemption point (line 3). If that is not the case, a severe system level error has occurred and the Health Monitor is notified (line 4) to handle the situation. The AIR Health Monitor is a component, not represented in Figure 3, that aims to contain faults within their domains of occurrence, to provide the corresponding error handling capabilities and that it spreads throughout virtually all of the AIR architectural components. The remaining lines (6-12) implement the partition switch actions of [4], checking (line 6) if there is a pending scheduling switch to be applied

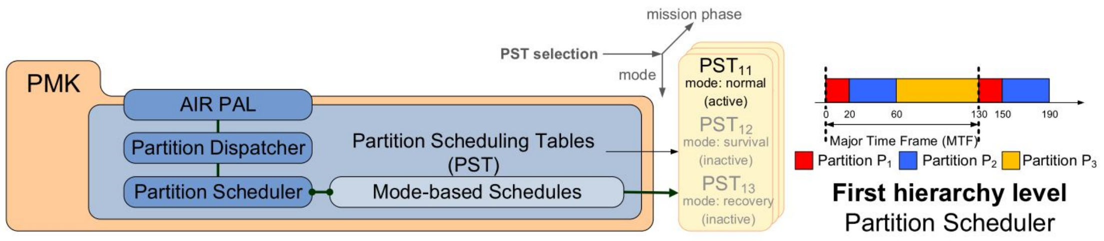


Figure 4: Partition scheduling featuring mode-based schedules

Algorithm 1 AIR Partition Scheduler, with Runtime Verification featuring mode-based schedules

```
1:  ▷ Entered upon exception: partition preemption point detected
2:  ▷ Runtime verification actions
3:  if $schedules_{currentSchedule}.table_{tableIterator}.tick \neq$
        $(currentTicks - lastScheduleSwitch)$ mod
        $schedules_{currentSchedule}.mtf$ then
4:      HEALTHMONITOR($activePartition$)
5:  else    ▷ Partition switch actions
6:      if $currentSchedule \neq nextSchedule \wedge$
            $(currentTicks - lastScheduleSwitch)$ mod
            $schedules_{currentSchedule}.mtf = 0$ then
7:          $currentSchedule \leftarrow nextSchedule$
8:          $lastScheduleSwitch \leftarrow currentTicks$
9:          $tableIterator \leftarrow 0$
10:     end if
11:     $heirPartition \leftarrow$
        $schedules_{currentSchedule}.table_{tableIterator}.partition$
12:     $tableIterator \leftarrow (tableIterator + 1)$ mod
        $schedules_{currentSchedule}.numberPartitionPreemptionPoints$
13: end if
```

Algorithm 2 AIR Partition Dispatcher, with Runtime Verification updating partition preemption points

```
1: ▷ Entered from the AIR Partition Scheduler after partition switch actions
2: SAVECONTEXT($activePartition.context$)
3: $activePartition.lastTick \leftarrow currentTicks - 1$
4: $elapsedTicks \leftarrow currentTicks - heirPartition.lastTick$
5: $activePartition \leftarrow heirPartition$
6: REPLACEPREEMPTIONPOINT($heirPartition.tick$)
7: RESTORECONTEXT($heirPartition.context$)
```

Algorithm 3 AIR Event Observer

```
1:  ▷ Input: Clock - system_clock_tick; Bus - raw event
2:  ▷ Output: Event - event
3:  for $system\_clock\_tick$ do
4:      $numTicks \leftarrow numTicks + 1$
5:      if $newEvent(Bus)$ then
6:          if $\exists\, id \in Config : Config[id] = Bus.trf$ then
7:              $event.time \leftarrow numTicks$
8:              $event.id \leftarrow id$
9:              $outputEvent(event)$
10:         end if
11:     end if
12: end for
```

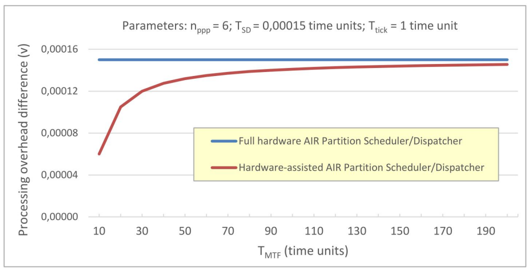


Figure 5: Analysis of processing time overheads

and the current instant is the end of the MTF. If these conditions apply, a different PST will be used henceforth (line 7). The processing resources are assigned to the heir partition, obtained (line 11) from the PST in use. The Partition Scheduler is set (line 12) to access the heir partition parameters.

4.3 Partition dispatching

The execution is followed by the AIR RV Partition Dispatcher specified in Algorithm 2. Two significant differences do exist from the software-based approach of [4]: elapsed clock ticks settings is no longer needed because the partition dispatcher is always invoked after a partition switch; insertion of the next partition preemption point in the hardware-assisted AOM configuration (line 6). The remaining actions in Algorithm 2 are related to saving and restoring the execution context (lines 2 and 7) and evaluation of the elapsed clock ticks (line 4).

4.4 Observation of application components

Besides the AIR RV Partition Scheduler and Dispatcher, two fundamental parts of our system, one dedicate our attention to the monitoring of other components, such as the applications. Through the use of the AOM module, observation and monitoring continues to be non-intrusive. This is done through Algorithm 3, the AIR Event Observer.

The AOM observes the Bus, compares (line 6) the transfer operations Bus.trf with a configured set of observation points, Config. Upon match, it sends a piece of information to the external system (line 9). This piece of information is an event, being comprised of: the time-stamp of the occurrence; the id of the event, specified in the configuration (lines 7-8). The numTick value (line 4) is incremented at every system clock tick, and used as the event time-stamp.

5 Evaluation: analysis and discussion

One relevant metric for code complexity is its size, in lines of source code. The standardized accounting method one employ is the logical source lines of code (logical SLOC) metric of the Unified CodeCount tool [11]. The C implementation of fundamental AIR components, such as the AIR Partition Scheduler and Dispatcher, is assessed in Figure 6, which shows its logical SLOC count along with the entity instantiating the component, and implicitly, the instantiation frequency. The data show a reduction of code complexity.

	Logical SLOC	Instantiation
Software-based AIR Partition Scheduler (specified an analysed in [4, 10])	13	POS-level clock tick
Software-based AIR Dispatcher (specified an analysed in [4, 10])	10	POS-level clock tick
Hardware-assisted AIR RV Partition Scheduler (specified in Algorithm 1)	12	partition preemption point
Hardware-assisted AIR RV Dispatcher (specified in Algorithm 2)	8	partition preemption point

Figure 6: Logical SLOC metrics and instantiation entities for fundamental AIR software components

With respect timing issues, comparing the normalised processing time overheads of AIR Partition Scheduler and Dispatcher (T_{SD}), in the software-based and hardware-assisted approaches, along a full normalised MTF period (T_{MTF}):

$$V \approx \frac{T_{SD\_soft}}{T_{sys\_tick}} - \frac{T_{SD\_Hard}}{T_{MTF}} \cdot n_{ppp} \qquad (1)$$

where, n_{ppp} is the number of partition preemption points in the MTF and T_{sys_tick} is the normalised POS-level clock tick. The normalisation of timing parameters in Figure 5 take the experimental values $T_{SD\_Soft} = 150\ ns$ and $T_{sys\_tick} = 1\ ms$ as references, making $T_{SD\_Hard} \approx T_{SD\_Soft}$ for hardware assisted and $T_{SD\_Hard} = 0$ for a full hardware implementation of the AIR Partition Scheduler/Dispatcher [12].

To exemplify the use of AIR AOM hardware in the observation/monitoring of several events, consider the Attitude and Orbit Control Subsystem (AOCS) function of a Low Earth Orbit (LEO) satellite. The Cartesian coordinates are used to evaluate the satellite position:

$$(x - u_x)^2 + (y - u_y)^2 + (z - u_z)^2 \leq (\delta_d)^2 \qquad (2)$$

where, (x, y, z) are the real position of the satellite and (u_x, u_y, u_z) are the specified satellite position; the value δ_d defines a specified maximum distance deviation.

The real position of the satellite is read and compared with the specified position. This difference should be kept below a given and specified threshold. If a violation occurs, such an event will be signalled to the AIR Health Monitor.

The synthesis of a monitor can be ensured with TeSSLa [13], a Temporal Stream-based Specification Language, which is specially designed for specifying correct program executions.

6 Related Work

Approaches to flexible scheduling in TSP systems are restricted to the mode-based scheduling of the commercial Wind River VxWorks 653 product [14]. Alternatives to TSP/IMA are compared in [15], which includes recommendations for adaptation of IMA-like solutions. Emergence of non-intrusive runtime verification techniques for embedded systems in general is addressed in [16, 17], while its applicability to complex safety-critical systems is presented in [18]. However, no previous work have applied such techniques to the realm of TSP systems.

7 Conclusion

This paper addressed how mechanisms providing support to the AIR architecture for time- and space-partitioned systems can be designed and engineered. The usage of a non-intrusive AIR Observer and Monitor allows not only the monitoring of fundamental AIR components but also of generic events. Non-intrusive runtime verification is a relevant contribution with respect to verification, validation and certification efforts of TSP systems that will be extended in future research.

References

[1] AEEC (Airlines Electronic Engineering Committee) (2006), *Avionics Application Software Standard Interface, Part 1 - Required Services*.

[2] TSP Working Group (2009), *Avionics time and space partitioning user needs*, Technical Note TEC-SW/09-247/JW, ESA.

[3] J. Rushby (1999), *Partitioning in avionics architectures: Requirements, mechanisms and assurance*, Tech. Rep. NASA CR-1999-209347, SRI International.

[4] J. Rufino, J. Craveiro, and P. Verissimo (2010), *Architecting robustness and timeliness in a new generation of aerospace systems*, in Architecting Dependable Systems VII, vol. 6420 of LNCS, Springer.

[5] M. Leucker and C. Schallhart (2009), *A brief account of runtime verification*, The Journal of Logic and Algebric Programming, vol. 78, pp. 293–303.

[6] R. C. Pinto and J. Rufino (2014), *Towards non-invasive runtime verification of real-time systems*, in 26th Euromicro Conf. on Real-Time Systems - WIP Session, (Madrid, Spain), pp. 25–28.

[7] S. Santos, J. Rufino, T. Schoofs, C. Tatibana, and J. Windsor (2008), *A portable ARINC 653 standard interface*, in Proc. 27th Digital Avionics Systems Conf., (St. Paul, MN, USA).

[8] J. Rosa, J. P. Craveiro, and J. Rufino (2011), *Safe online reconfiguration of time- and space-partitioned systems*, in Proc. 9th IEEE Int. Conf. on Industrial Informatics (INDIN 2011), (Caparica, Lisbon, Portugal).

[9] AEEC (Airlines Electronic Engineering Committee) (2008), *Avionics Application Software Standard Interface, Part 2 - Extended Services*.

[10] J. P. Craveiro and J. Rufino (2010), *Adaptability support in time- and space-partitioned aerospace systems*, in Proc. 2nd Int. Conf. on Adaptive and Self-adaptive Systems and Applications, (Lisbon, Portugal).

[11] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. Boehm (2007), *A SLOC counting standard*, in The 22nd Int. Ann. Forum on COCOMO and Systems/Software Cost Modelling, (Los Angeles, USA).

[12] J. Rufino (2016), *Towards integration of adaptability and nonintrusive runtime verification in avionic systems*, ACM SIGBED Review, vol. 13.

[13] M. Leucker, C. Sánchez, T. Scheffel, M. Schmitz, and A. Schramm (2018), *TeSSLa: Runtime verification of nonsynchronized real-time streams*, in ACM Symp. on Applied Computing (SAC), (Pau, France), ACM.

[14] Wind River (2015), *Wind River VxWorks 653 Platform 2.4 and 2.5*.

[15] B. Ford, P. Bull, A. Grigg, L. Guan, and I. Phillips (2009), *Adaptive architectures for future highly dependable, real-time systems*, in Proc. 7th Ann. Conf. on Systems Engineering Research, (Loughborough, UK).

[16] C. Watterson and D. Heffernan (2007), *Runtime verification and monitoring of embedded systems*, Software, IET, vol. 1, pp. 172–179.

[17] T. Reinbacher, M. Fugger, and J. Brauer (2014), *Runtime verification of embedded real-time systems*, Formal Methods in System Design, vol. 24, no. 3, pp. 203–239.

[18] A. Kane (2015), *Runtime Monitoring for Safety-Critical Embedded Systems*, PhD thesis, Carnegie Mellon University, USA.

In memoriam: José Rufino

José Rufino passed away this summer. This summer, we lost someone on whom we could always count, we could talk to and be heard by, we could receive back nice words and incentives. José had a strong voice, but we never heard him raising his voice to anyone. We also lost his deep technical knowledge in the area of real-time and embedded systems, his ideas and persistence in pursuing them, his scientific honesty and strict rigor and his great collaborative skills. We will miss you.

José devoted perhaps most of his time to his students. He cared to conveying knowledge, always seeking to improve the course materials, exercises, investigating new ways to make them learn more and better. He was considered a very demanding teacher, a characteristic that many students are only able to appreciate years later. He was also extremely dedicated and competent teacher. We believe that there will hardly be a student who will not remember José in a positive way. The students will miss you.

José was one of the kindest and educated persons we ever met. He was a very reserved person, his kindness was too great and he didn't like to bother with his problems. Despite so many years knowing him, there are only very few things we knew about his personal life, what he liked and what he used to do when being away from the university. He loved an old car that belonged to his father, which he kept in good shape as he knew a lot about automotive mechanics and he could do the repairs by himself. But he didn't drive this car (at least not always), as he preferred to use public transportation to avoid traffic jams when going to work. He was taking care of his relatives who lived in Alentejo (far from Lisbon), so he went there very often and, from time to time, did some maintenance work on the house there. That's how he was. He lived his life thinking more about the others than about himself.

We will miss you and we know well how much we lost by not having you with us any longer.

Antonio Casimiro, who met José in 1989 and worked together for almost 30 years.

Frank Singhoff, Laurent Lemarchand, Stéphane Rubini, Nam Tran Hai, Jalil Boukhobza, your friends from Brest.

José Rufino was part of a crazy dream from the very start --- to create an internationally successful research group in the Portugal of the mid-eighties. That's how the Navigators group --- then at INESC and TU Lisboa --- was created, and José was there from the start. José fulfilled his part of the dream, becoming one of the most renowned and respected researchers on safety and reliability of distributed real-time and embedded systems. His achievements in this area are highly cited, and I can only thank him for having let me be his advisor and later, colleague, doing so many things together during more than 30 years.

He allied an ultra-soft and polite temper (never heard him raise his voice), with an indestructible stubbornness (which the wise call persistence), and that was a great asset to the group, especially in times of doubt or uncertainty. The Navigators "never failed a demo" (literally, in over 30 years, and more than 40 projects), and a huge part of that success was due to José's meticulousness, leaving no detail behind. His sophisticated sense of humor would be present even in writing papers, like when I was obliged to make a graphical explanation of the "Columbus Egg" story at a PC dinner, after he named a paper after the fable (José, we should have patented that one…).

It is impossible to be concise enumerating his human qualities, as they were so many, but his generosity was immense, and despite worrying about his friends and always being there to help, we knew very little about his own ordeals. Being a research-oriented professor, his dedication to teaching and to his students should be an example to us all.

He left us too early, but he will always be remembered.

Paulo Esteves-Veríssimo

(Head of the Navigators Group, 1985-2014)

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
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
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