ADA USER JOURNAL

Volume 37
Number 3
September 2016

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

In this issue of the Ada User Journal, we continue the publication of contributions from two important Ada events, which took place this year.

First, we include a set of papers from the industrial track of the Ada-Europe 2016 conference, last June in Pisa, Italy. As usual, the industrial track of the conference is an important component of the program, allowing the community to know how Ada and reliable software technologies are being used in industrial settings. The first paper of the issue, by J-P. Rosen, of Adalog and J-C. Van-Den-Hende of ALSTOM Transport, France, discusses how Ada's visibility rules have been used to help in the process of guaranteeing the required component segregation in systems with mixed criticality. Then, we have two papers presenting results from the CONCERTO European project. Silvia Mazzini, Stefano Puri and Andrea Russino, from Intecs, Italy, present how the CHESS modelling approach fits within the development of AUTOSAR systems, and Wenceslas Godard, from Airbus Group Innovations, France, and Geoffrey Nelissen from CISTER, Portugal, discuss how model-based design can be used to help in the integration of several components in the same platform, whilst guaranteeing their timing requirements.

The second part of the issue is dedicated to the results of the International Real-Time Ada Workshop, last April in Benicàssim, Spain. After publishing an overview of the workshop in the last June issue of the Journal, in this issue we republish the summaries of the technical sessions. These summaries have been originally published in the June issue of Ada Letters (together with the workshop position papers), and provide both the discussion as well as conclusions of the workshops six sessions: Parallel and Multicore Systems, Deadline Floor Protocol, Language Issues, Ada Language Profiles, Experience and Time Vulnerabilities.

In the remaining of the issue, a special note to the information provided in the News, Calendar and Forthcoming Events sections. In particular, the latter provides the call for papers for the 22nd International Conference on Reliable Software Technologies – Ada-Europe 2017, to take place June 2017 in Vienna, Austria. The deadline for contributions is January 15th (it seems to be far away, but in reality it is around the corner). After the events section, the reader will also find a call for contributions to the Ada User Journal. It is important that the community supports both the conference and the journal, both by participating and reading, as well as by contributing!

And since we are talking about support, I would like to share with our readers a small, but relevant, news. For the first time Ada-Europe counts with 20 sponsoring companies (you can find them in the inside back cover of the journal). This is indeed something to be happy as it shows both a vibrant community and interest in the activities of the organization. Thank you all for the support.

Luís Miguel Pinho
Porto
September 2016
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation. Email: jacob@jacob-sparre.dk

Contents

Ada-related Organisations

ACM SIGAda Award

From: ACM SIGAda
Date: Tue Jul 19 2016
Subject: ACM SIGAda Award for Ada Community Contributions Named for Late Ada Luminary Robert Dewar
URL: http://www.sigada.org/SIGAdaAwardRenaming.pdf

NYU Professor Emeritus / AdaCore founder played key role in Ada language design and implementation.

NEW YORK, July 19, 2016 - ACM's Special Interest Group on Ada (SIGAda) today announced that its annual award for "broad, lasting contributions to Ada technology and usage" has been named the "Robert Dewar Award for Outstanding Ada Community Contributions". Dr. Dewar, who passed away in June 2015, received this award himself in 1995 – it was then known as SIGAda's Ada Community Contributions Award – in recognition of his innovative technological achievements surrounding the Ada language. Other past recipients of this award include Jean Ichbiah, the head of the design team for the original Ada language; and Tucker Taft, the head of the Ada 95 revision team. This year's recipient(s) of the Robert Dewar Award for Outstanding Ada Community Contributions will be announced at SIGAda's High-Integrity Language Technology (HILT) workshop during Embedded Systems Week in Pittsburgh in October.

"It is hard to overestimate how Robert Dewar shaped the Ada landscape throughout his professional career," said Dr. David Cook, SIGAda Chair. "He was an innovator, and an inspiration to many. I personally compiled my first Ada program in 1986 during a tutorial and workshop taught by Robert. On behalf of SIGAda, it is our honor to have our award named after him.

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. —sparre]

Highlights from Ada-Europe 2016

From: Ian Broster
Date: Wed 6 Jul 2016
Subject: Highlights from Ada Europe 2016
URL: https://www.rapitasystems.com/blog/highlights-ada-europe-2016

I write this on the plane home from the Ada Europe 2016 conference in sunny Pisa, Italy. It's been a good week. Here are a few highlights.

<u>Ada</u>

Firstly, it was good to see the key industrial sponsors like AdaCore pushing and supporting the Ada language itself. PCT, Ansys and others seem to have stepped up their Ada support too.

At Rapita we like Ada - it's a great way of writing reliable software. Many of our aerospace customers use Ada for that reason: projects using Ada work, are more cost effective and seem to come in on-time and budget. Ada is a very valuable language for now and the future, not just for aerospace; the general embedded industry could save a lot of effort by its use. We hear story after story of how the inappropriate use of C#, C, C++ has lead to big problems in medical devices, automotive and industrial automation.

Most of Rapita's software is written in Ada using GNAT Pro, so we really understand the language - this is one of the reasons we provide the best and most complete Ada language support in RVS for coverage, timing and unit-test.

AdaCore have been instrumental in supporting Ada, maintaining the GPL and PRO compilers, books, education, academic programs and the various initiatives including the Make with Ada competition launched this week. Do take a look at this!

The Ada Pilot initiative is also a good promotion of Ada - I finally met Jarno Puff who is the key person pushing this project forwards. The project aims to build an open source Ada flight system for drones, which people can use on their own hobby projects or in commercial situations. It promises to be a great way to get the message out. Rapita is pleased to support this initiative.

<u>Multi-core and Many-core</u>

Multi-core and parallel computation in Ada was a recurring theme this week. There are various language extensions and parallel programming libraries proposed, courtesy of Tucker Taft (AdaCore) and Brad Moor (General Dynamics). They have been looking at ways to efficiently target Ada at many-core platforms for high performance computing. There were also a number of papers presented on this topic.

Of course Ada already supports multi-core processors through its language-level concurrency, yet there is more exciting work to come with fine-grained parallelism opportunities on many-cores. For example, the ability to automatically parallelize iterations of a 'for loop' to different processors.

<u>The Ada Community</u>

It was fantastic to meet new industrial people at the conference this year - key companies who use Ada such as Airbus, BAE, Altran (and more) were represented and it was good to meet top engineers who really have experience of large reliable systems development. However, there were many industries that were not represented, which is a missed opportunity.

Speaking for the UK, I hope that we can encourage more UK companies to get involved through the formation of the new Ada UK organization - thanks to Dene Brown for setting this up - there are lots of opportunities for community building, collaboration and supporting a technology that we all rely on. I encourage you to join Ada UK - whether as a personal member or as a corporate body. Everyone who uses Ada should be a member - but we need to transform this from a "charity donation" mentality to something that gives real benefit to its members: what would you like to see Ada UK do? What can Ada UK do for you? Answers via LinkedIn.

I would also encourage the conference organizers to think hard how to get more engineers to the conference - perhaps that means restructuring the conference a bit to make it practical for industrial people to continue to come - this of course needs a joined-up approach, from targeting the call for papers, understanding what industrial people would get out of it, and making sure that the logistics work.

Testing, Unit testing and Verification

Finally, it was a pleasure to receive the prize for the best presentation for the paper "Automated Testing of SPARK Ada Contracts (AUTOSAC)" - this is a nice piece of work, based on our RapiTest Framework tool, in collaboration with Altran and the University of Oxford and I thank those who did the hard work in preparing the technical work.

So, thanks to the organizers - great job again - hope to see you in Vienna for next year's conference.

Courses in Paris

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Mon, 18 Jul 2016 17:37:10 +0200
Subject: [Ann] Programme des formations Adalog / 2ème semestre 2016
Newsgroups: comp.lang.ada

Formation "Ada cours complet" (6 jours):

Cette formation couvre tout Ada83+95, et présente les points les plus importants d'Ada 2005/2012.
Elle s'adresse à tous ceux qui sont amenés à développer ou à s'occuper de projets en Ada.

21 - 23 et 26 - 28 septembre 2016

05 - 07 et 12 - 14 décembre 2016

Seule la connaissance d'un autre langage de programmation est requise.

[Introduction to Ada for experienced programmers.]

Formation Ada 2005 et 2012 (3 jours):

Cette formation s'adresse à ceux qui pratiquent déjà Ada95 et veulent apprendre les nouvelles possibilités offertes par la dernière version du langage.

Du 15 au 17 novembre 2016

[Ada 2005 and 2012 course for Ada 95 programmers.]

Pour plus d'information, merci d'écrire à info@adalog.fr, ou de visiter http://www.adalog.fr/adaf1.html

FOSDEM 2017

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Fri, 9 Sep 2016 07:40:18 +0200
Subject: Ada at FOSDEM 2017 - proposal submitted
To: ADAFOSDEM@LS.KULEUVEN.BE

As planned, I have submitted yesterday evening a proposal for an Ada Developer Room at FOSDEM 2017, similar to the ones of past years.

The deadline was earlier this year (today 9 September) to give developer room organisers more time to plan their own schedules. One requirement for FOSDEM 2017 is that accepted DevRooms must enter a complete schedule into their conference system by 11 December, which is also earlier than in the past.

All DevRoom proposals will now be reviewed and we will be informed whether ours is accepted or not via email by 21 September.

Pending that decision, you might want to keep the FOSDEM weekend free in your agenda: Sat 4 - Sun 5 February 2017.

I will keep you all informed.

[...]

Ada-related Resources

Ada on Social Media

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed Sep 14 2016
Subject: Ada on Social Media

Ada groups on various social media:

- LinkedIn:	2_481 members	[1]
- Reddit:	889 readers	[2]
- Google+:	686 members	[3]
- StackOverflow:	585 followers	[4]
- Freenode	74 participants	[5]
- Twitter:	8 tweeters	[6]

[1] https://www.linkedin.com/groups?gid=114211

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

[3] https://plus.google.com/communities/102688015980369378804

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

[5] #Ada on irc.freenode.net

[6] https://twitter.com/search?f=realtime&q=%23AdaProgramming

[See also “Ada on Social Media”, AUJ 37-2, p. 70. —sparre]

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed Sep 14 2016
Subject: Repositories of Open Source software

GitHub:	1_424 repositories	[1]
	281 developers	[1]
	1_152 issues	[1]
Rosetta Code:	627 examples	[2]
	30 developers	[3]
	1 issue	[4]
Sourceforge:	252 repositories	[5]
BlackDuck OpenHUB:	211 projects	[6]
Bitbucket:	88 repositories	[7]
OpenDO Forge:	24 projects	[8]
	494 developers	[8]
Codelabs:	19 repositories	[9]
AdaForge:	8 repositories	[10]

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

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

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

[4] http://rosettacode.org/wiki/Category:Ada_examples_needing_attention

[5] http://sourceforge.net/directory/language%3Aada/

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

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

[8] https://forge.open-do.org/

[9] http://git.codelabs.ch/

[10] http://forge.ada-ru.org/adaforge

[See also “Repositories of Open Source Software”, AUJ 37-2, p. 70. —sparre]

Ada-related Tools

INI File Manager

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sun 8 May 2016
Subject: Ini file manager
URL: https://sourceforge.net/projects/ini-files/

Config is an Ada package for parsing configuration files (.ini, .inf, .cfg, ...) and retrieving keys of various types. New values for single keys, or entire sections, can be set. Standalone and unconditionally portable code.

Features

- Pure Ada 95 (nothing compiler/system specific)
- Standalone (no dependency on other packages)
- Object oriented

GLOBE_3D

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Tue, 5 Jul 2016 18:54:38 -0700
Subject: Ann: GLOBE_3D Release 2016-07-05 - "Blender edition"
Newsgroups: comp.lang.ada

GLOBE_3D is a GL Object Based 3D engine realized with the Ada programming language.

URL: http://globe3d.sf.net

Latest additions:

- Use of Generic Image Decoder (GID) in GL.IO; now most image formats are supported for textures and other bitmaps to be used with GLOBE_3D (or any GL app)
- New Wavefront format (.obj / .mtl) importer
- Doom 3 / Quake 4 map importer more complete
- Unified GNAT project file (.gpr), allowing to selected the target Operating System (Windows, Linux, Mac) and compilation mode (fast, debug, small) for demos, tools, etc.
- Project file for ObjectAda 9.1+ updated

The first two points facilitate the import of 3D models from software such as Blender.

Here is an example: http://globe3d.sf.net/g3d/futj.jpg

Coincidentally, the Wavefront file format so simple that you can also write 3D models "by hand" in that format.

An example made in an Excel sheet is provided along with the importer, in the ./tools/wavefront directory.

[See also “GLOBE_3D”, AUJ 37-2, p. 76. —sparre]

Gnoga

From: Pascal Pignard <p.p11@orange.fr>
Date: Wed, 13 Jul 2016 21:23:13 +0200
Subject: GNOGA 1.2 beta.
Newsgroups: gmane.comp.lang.ada.gnoga

No major changes in Gnoga since a while, so Gnoga 1.2 state changes from alpha to beta today in last SF commit today.

Volunteers are welcome to test it on their own configuration. Mine is MacOS 10.11, GNAT GPL 2016, Safari and Firefox. Some testing on Windows and Linux configuration will be appreciated.

Just get last today commit on https://sourceforge.net/p/gnoga and do:

```
$ make gnoga
$ make demo
$ make tutorials
```

and for courageous:

```
$ make test
$ cd bin
```

and test.

Feel free to report detailed issue on this list or create tickets on SF.

[See also “Gnoga”, AUJ 37-1, p. 7. —sparre]

From: Pascal Pignard <p.p11@orange.fr>
Date: Sat, 9 Jul 2016 10:48:07 +0200
Subject: Tip of the day.
Newsgroups: gmane.comp.lang.ada.gnoga

If you want to browse through the Gnoga API, generate them with gnatdoc:

```
$ make rm-docs
$ open docs/html/gnoga_rm/index.html
```

libsodium

From: John Marino <dragonlace.cla@marino.st>
Date: Sun, 17 Jul 2016 17:50:13 -0700
Subject: ANN: Thick bindings for libsodium
Newsgroups: comp.lang.ada

I was unable to find any bindings for libsodium (https://github.com/jedisct1/libsodium) so I created my own:

https://github.com/jrmarino/libsodium-ada

I split them out of a private project I'm working on because I found libsodium to be highly useful and I suspect other Ada users will feel the same way.

These bindings are thick and I've committed the test cases I was using to serve as examples. They cover the most of the functionality of libsodium, but some are missing the thick counterparts (e.g. most of the detached versions have no thick counterpart as I stuck with the recommended "combined" variants).

However, feel free to improve what I have via the github pull request mechanism.

Simple Components (et al.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 25 Jul 2016 08:15:33 +0200
Subject: AICWL, GtkAda contributions, Fuzzy sets, Units GtkAda 3.14 update
Newsgroups: comp.lang.ada

I upgraded all packages relying in part on GtkAda to the version 3.14 as distributed with GNAT GPL 2016. Here is the full list:

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

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

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

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

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

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

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

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

Packaged GtkAda GPL 3.14.2 for Debian and Fedora is here:

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

[See also “Simple Components”, AUJ 37-2, p. 70. —sparre]

Emacs Ada Mode

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 27 Jul 2016 14:56:16 -0500
Subject: ada-mode 5.2.0 released
Newsgroups: gmane.comp.lang.ada.emacs
To: emacs-ada-mode-3GsT/cKSLGBCWfS9sVZFbQ@public.gmane.org

ada-mode 5.2.0 is now available in Gnu ELPA, and at http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html

I'm still working on getting the Savannah project set up.

[See also “Emacs Ada Mode”, AUJ 37-1, p. 8. —sparre]

ASN.1

From: Edward R. Fish <onewingedshark@gmail.com>
Date: Mon, 1 Aug 2016 01:15:21 -0700
Subject: ANN: Ada/SPARK ASN.1 implementation version 0.0.01
Newsgroups: comp.lang.ada

I'm making public my ASN.1 project which aims to be a verified implementation of ASN.1, which is used in security-certificates, which is hopefully the first step in a verified-TLS/-TLS -- the project also aims to be [directly] usable in DSA projects.

As of 0.0.01 the only portion implemented is a pure big-number package, and another currently shared-passive unit for usability.

I would certainly appropriate comments, criticism, and most especially contributions

https://github.com/OneWingedShark/ASN.1/

From: Thanassis Tsiodras <ttsiodras@gmail.com>
Date: Tue, 2 Aug 2016 02:01:18 -0700
Subject: Re: ANN: Ada/SPARK ASN.1 implementation version 0.0.01
Newsgroups: comp.lang.ada

> [...]

I am not sure if it can be used with the TLS ASN.1 grammar - but I think it's worth checking out our own open-source ASN.1 compiler, targeting both C and Spark/Ada (developed under the auspices of the European Space Agency, so targeting the same kind of safety-critical targets you probably have in mind).

The compiler is here:

https://github.com/ttsiodras/asn1scc

And a crash-course in using it is here:

https://www.thanassis.space/asn1.html

Note also that a new project has just started that will add support for Spark 2014.

From: Thanassis Tsiodras <ttsiodras@gmail.com>
Date: Wed, 3 Aug 2016 00:13:56 -0700
Subject: Re: ANN: Ada/SPARK ASN.1 implementation version 0.0.01
Newsgroups: comp.lang.ada

The project contract has just been signed, and will kick-off after the summer vacations. The expected duration of the work is 12 months, with development done in the open (in a branch on the GitHub repo).

Based on past experiences, we anticipare working versions (i.e. with SPARK 2014 support) a lot sooner than 12 months.

[See also "ASN.1", AUJ 34-2, p. 69. —sparre]

Zstd

From: John Marino <dragonlace.cla@marino.st>
Date: Tue, 2 Aug 2016 07:14:49 -0700
Subject: ANN: Thick Ada bindings for Zstd (Fast real-time compression algorithm)
Newsgroups: comp.lang.ada

The Zstandard compression algorithm just reaching version 0.8.0 and it's nearly stable. It has a wide range range of fast vs high compression levels. At the fast levels, it's much faster than gzip with the same compression and it can approach xz -6 levels of compression in a fraction of the time. It's very nice and exactly what I was looking for personally.

http://www.zstd.net

I spent some time creating some thicking bindings for Ada:

https://github.com/jrmarino/zstd-ada

Those bindings cover the stable API. What it does not cover is streaming compression. That API is not yet stable and not even available in the shared library (only the static one). When the streaming compression is stablized, I'll probably update the bindings. As usual, contributions via github are always welcome.

ZanyBlue

From: Michael Rohan <michael@zanyblue.com>
Date: Sat, 6 Aug 2016 19:46:30 -0700
Subject: ANN: ZanyBlue - 1.3.0b available
Newsgroups: comp.lang.ada

The ZanyBlue library and utils version 1.3.0b is available for download at

http://zanyblue.sourceforge.net

or directly from the download area:

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

The changes since the last release are:

- Added a new utility zbinfo to query built-in data. This was released as an example previously (the dumplocale example which has been dropped).
- Added encoding support to convert Wide_String values to String based on an encoding schema, e.g., UTF-8, ISO8859-2, CP932, SHIFT_JIS, etc. To fully use this functionality, narrow accessors should be used which, when printing, use Stream_IO to avoid interaction between the Text_IO and encoded values. The list of supported encodings is available via zbinfo --list-encodings.
- The default locale is now en_US.UTF-8 if no other locale can be determined from the environment.
- Updated the documentation (and website) to use the Sphinx documentation system.
- Updated and expanded the documentation. Additional documentation is, however, needed. Switch to gnatdoc from gnathtml to generate the source code based documentation.
- Restricted the usage of the -gnatW8 compilation option to just the source files containing UTF-8 encoded strings: the message pool file generated by the zbmcompile utility.
- Added option to the zbmcompile utility to generate ASCII only source files (-A option).
- Added option to the zbmcompile utility to define handling of non-Ada message keys when generating accessors (the -X option).
- Updated the build to use gprbuild instead of gnatmake.
- Updated the build to use -gnat2012 in all gpr files.
- Switched from AUnit to Ahven for unit testing.
- Minor source code changes based on stricter checks with GNAT 2016.

[See also "ZanyBlue", AUJ 36-4, p. 202. —sparre]

Zip-Ada

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Fri, 26 Aug 2016 23:15:33 -0700
Subject: Ann: Zip-Ada v.51
Newsgroups: comp.lang.ada
URL: https://sf.net/projects/unzip-ada/

Changes in '51', 27-Aug-2016:

- LZMA.Encoding has been added; it is a standalone compressor, see lzma_enc.adb for an example of use.
- Zip.Compress provides now LZMA_1, LZMA_2 methods. In other words, you can use the LZMA compression with Zip.Create.
- Zip.Compress has also a "Preselection" method that selects a compression method depending on hints like the uncompressed size.
- Zip.Compress.Deflate: Deflate_1 .. Deflate_3 compression is slightly better.

The LZMA format, new in Zip-Ada, is especially good for compressing database data - be it in binary or text forms. Don't be surprised if the resulting archive represent only a few percents of the original data...

[See also "Zip-Ada", AUJ 37-2, p. 71. —sparre]

PragmAda Reusable Components

From: PragmAda Software Engineering <pragmada@pragmada.x10hosting.com>
Date: Mon, 5 Sep 2016 12:04:30 -0700
Subject: Updated PragmAda Reusable Components
Newsgroups: comp.lang.ada

There are some new components in the beta version of the PragmARCs for ISO/IEC 8652:2007: PragmARC.Concurrent_Pipeline and PragmARC.Holders.

Concurrent_Pipeline was inspired by discussions of the Rx approach to concurrency, such as RxJava, in which a sequence of operations are chained together from a source to a sink, the operations being able to proceed in parallel but only one execution of a given operation at a time. If one ignores the Rx syntax and concentrates on providing the functionality in a way that's natural for Ada, it becomes fairly simple: Concurrent_Pipeline is 45 Ada terminator semicolons.

I haven't looked at the Rx approach in detail, so there may be differences between it and Concurrent_Pipeline.

It's not clear that Concurrent_Pipeline pipeline is needed; it seems the same functionality could be achieved with PragmARC.Job_Pools. It may be a more natural approach for some problems, though.

Holders provides variables for indefinite types; something like it is needed to allow the operations in a Concurrent_Pipeline to proceed in parallel. It was also not strictly needed, as the same functionality could be obtained with an indefinite container that is only used to store a single value.

The PragmARCs may be obtained from Github

https://github.com/jrcarter/PragmARC

or from the web site.

[See also "PragmAda Reusable Components", AUJ 37-2, p. 76. —sparre]

SPARK 2014 Tools

From: Claire Dross, AdaCore
Date: Wed 14 Sep 2016
Subject: P909-030 duplicate checks on split scalar types
URL: https://github.com/AdaCore/spark2014

SPARK 2014 is the new version of SPARK, a software development technology specifically designed for engineering high-reliability applications.

[...]

This repository contains the source code for the SPARK 2014 project. SPARK is a software development technology specifically designed for engineering high-reliability applications. It consists of a programming language, a verification toolset and a design method which, taken together, ensure that ultra-low defect software can be deployed in application domains where high-reliability must be assured, for example where safety and security are key requirements.

[See also "GNAT GPL and SPARK GPL", AUJ 37-2, p. 75. —sparre]

A-CUPS

From: Per Sandberg <per.s.sandberg@bahnhof.se>
Date: Thu 15 Sep 2016
Subject: Make the generated code private.
URL: https://github.com/persan/a-cups

An Ada binding to the CUPS printing subsystem.

GNATColl

From: Emmanuel Briot <briot@adacore.com>
Date: Thu 15 Sep 2016
Subject: The GNAT Collection Library.
URL: https://github.com/AdaCore/gnatcoll

The GNAT Components Collection

See the documentation in the docs/ directory for instructions on how to build, install and use gnatcoll.

The documentation is either available as a precompiled HTML file, or in the file building.rst.

To test GNATCOLL itself, you should run "make test" from the current directory, after compiling and installing GNATCOLL.

[See also "GNATColl.JSON Support Packages", AUJ 37-2, p. 75. —sparre]

Ada and Operating Systems

Windows: GNAVI: GNU Ada Visual Interface

From: Gautier de Montmollin <gautier.de.montmollin@gmail.com>
Date: Sat 21 May 2016
Subject: GNAVI: GNU Ada Visual Interface
URL: https://sourceforge.net/projects/gnavi/

[...] The durable Open Source answer to Delphi and VB.

Features

- Complete Windows framework
- Pure Ada code, standalone
- Object-Oriented
- Code generator (GWenerator)

[See also "GWindows Setup", AUJ 35-3, p. 153. —sparre]

Debian and Fedora: GtkAda

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 24 Jul 2016 09:28:18 +0200
Subject: GtkAda 3.14 GPL packages for Debian and Fedora
Newsgroups: comp.lang.ada

I packaged GtkAda 3.14 from GNAT GPL for Debian and Fedora

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

You can use them until GtkAda maintainers catch up.

P.S. There are some changes coming with 3.14 that will break legacy code, e.g. Signal_Name is no more a subtype of String. I also found that some operations do not work anymore, e.g. Load_Icon, though there are variants of that do. OK, if you are using GTK, you know what to expect (:-()

[See also "Debian and Fedora: GtkAda", AUJ 36-3, p. 127. —sparre]

Mac OS X: XNAdaLib

From: Pascal Pignard <p.p11@orange.fr>
Date: Fri, 2 Sep 2016 21:46:30 +0200
Subject: [ANN] XNAdaLib 2016 binaries for El Capitan including GTKAda and more.
Newsgroups: gmane.comp.lang.ada.macosx

This is XNAdaLib 2016 built on MacOS X 10.11 El Capitan for Native Quartz including:

- GTK Ada GPL 2016 with GTK+ 3.20.3 complete,
- Glade 3.18.3,
- GnatColl GPL 2016,
- Florist GPL 2016,
- AdaCurses 20110404,
- Gate 3-05-b,- Components 4.15,
- AICWL 3.15,
- Zanyblue 1.3.0b,
- PragmARC 07-2016-08,
- GNOGA 1.2-beta,
- AdaControl 1.18b4,
- Adadep 1.3r3

and as side libraries:

- Template Parser,
- gtksourceview 3.14.3,
- GNUTLS 3.3.12,
- ASIS GPL 2016.

to be installed (mandatory) at /usr/local:

```
$ cd /usr/local
$ sudo tar xzf xnadalib-gpl-2016-quartz-x86_64-apple-darwin14.5.0-bin.tgz
```

Update your PATH to include gtkada-config, glade, gate3.sh and other executables in it:

```
$ PATH=/usr/local/xnadalib-2016/bin:$PATH
```

Update your GPR_PROJECT_PATH to include gtkada.gpr, adacurses.gpr, florist.gpr, gnatcoll.gpr, gtkada_aicwl.gpr, gnoga.gpr and other projects in it:

```
$ export GPR_PROJECT_PATH=/usr/local/xnadalib-2016/lib/gnat:/usr/local/xnadalib-2016/share/gpr:$GPR_PROJECT_PATH
```

Set XDG_DATA_DIRS for GNOME apps:

```
$ export XDG_DATA_DIRS=/usr/local/xnadalib-2016/share
```

Glade and GPS applications in apps directory must stay in this directory unless you modify the script inside apps.

Then see documentation and examples in share directory and enjoy.

Here are the instructions I used to build XNAdaLib on MacOS (in French): http://blady.pagesperso-orange.fr/telechargements/gtkada/Install-GTKAda-Quartz_wf.pdf

Here are the modifications I made: http://blady.pagesperso-orange.fr/telechargements/gtkada/xadalib-2016-diff.tgz

XNAdaLib binaries have been posted on SourceForge: https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2016-el-capitan/

From: Pascal Pignard <p.p11@orange.fr>
Date: Sun, 11 Sep 2016 13:26:01 +0200
Subject: Re: [ANN] XNAdaLib 2016 binaries for El Capitan including GTKAda and more.
Newsgroups: gmane.comp.lang.ada.macosx

I built XNAdaLib 2016 again taking care to dependences with old versions.

The new archive xnadalib-gpl-2016a-quartz-x86_64-apple-darwin14.5.0-bin.tgz is on SF:

https://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2016-el-capitan

By the way PragmARC has been updated.

My apologies for troubles if any, please let have a try and keep me informed.

[See also "Mac OS X: XNAdaLib", AUJ 36-4, p. 204. —sparre]

References to Publications

Booklet on EN 50128

From: AdaCore Press Center
Date: Tue 28 Jun 2016
Subject: Free Booklet Shows How AdaCore's Technologies Can Help Railway Software Developers Meet EN 50128 Objectives
URL: http://www.adacore.com/press/en50128-booklet-help-railway-developers/

New 70-page report explains how to reduce safety certification effort through Ada language and qualified AdaCore tools

RSRR 2016, PARIS, June 28, 2016 - AdaCore today announced the publication of AdaCore Technologies for CENELEC EN 50128:2011. Authored by AdaCore expert Quentin Ochem and CERTIFER safety assessor Jean-Louis Boulanger, this booklet summarizes the EN 50128 railway software standard and explains how the Ada programming language and AdaCore's products can be used to meet its requirements throughout the software life cycle.

The new booklet presents the following technologies in the context of EN 50128:

- The Ada 2012 language, including its contract-based programming features
- The SPARK 2014 language (a formally verifiable subset of Ada 2012) and its supporting tools, which allow mathematical demonstration of program properties such as absence of run-time errors
- The GNAT Ada compiler, including run-time libraries that have been certified at the SIL 3 and SIL 4 levels
- Static analysis tools
  - o CodePeer, an advanced static analysis tool for code review and verification
  - o GNATmetric, a metric computation tool
  - o GNATcheck, a coding standard checker
  - o GNATdashboard, a platform for integrating and managing information from various analysis tools
- Dynamic analysis tools
  - o GNATtest, a unit testing framework generator
  - o GNATemulator, a host-resident target processor emulator
  - o GNATcoverage, a structural code coverage analyzer
- The QGen model-based development and verification tool, which translates from Simulink® and Stateflow® models into SPARK or MISRA-C

These tools fit into the "V" software life cycle [...]

The booklet has the following contents:

- A summary of the CENELEC EN 50128 standard
- An overview of the relevant AdaCore tools and technologies
- An explanation of AdaCore's contributions towards the Software Quality Assurance Plan
- A technology usage guide keyed to various requirements in EN 50128, such as Analyzable Programs and Boundary Value Analysis, showing how these requirements are met by AdaCore tools and technologies
- A technology annex, summarizing the qualification status of the various tools and showing how they relate to specific Annex D sections

"Certification requirements are getting more and more complex," said Quentin Ochem, Lead of Business Development and Technical Account Management at AdaCore. "In order to stay competitive, it is crucial yet difficult for railway system developers to fully understand how and where software tools can help. AdaCore's new booklet answers this need and provides a clear mapping between technology capabilities and railway safety requirements."

AdaCore products have been used in a variety of safety-critical railway systems and, as explained in the booklet, various tools have been qualified as T2 or T3 tools, and several run-time libraries have been certified at SIL-3 / SIL-4 levels, under EN 50128.

Availability

The AdaCore Technologies for CENELEC EN 50128:2011 booklet is available now, at no cost. To obtain a copy please contact info@adacore.com; it is also available for download from http://adacore.com/en-50128.

Ada Inside

AdaGate

From: FastRgv Development
Date: Sun Mar 20 2016
Subject: AdaGate
URL: https://directory.fsf.org/wiki/AdaGate

A combination of Portal and Sokoban, AdaGate is a great example of OpenGL programming using the Ada language. It is Open Source, of course.

While exploring a remote south-seas island you make a startling historical discovery. But before you can report your findings, an operational stargate transports you into a curious sequence of dungeons. Your escape will require the logical rearrangement of weird power cells, called Zero Point Modules (ZPMs), that can roll in only two directions.

[...]

[Download from https://github.com/fastrgv/AdaGate/blob/master/adagate-src-only-snapshot.tar.gz —sparre]

ExoMars

From: AdaCore Press Center
Date: Tue 19 Jul 2016
Subject: Ada on Board: GNAT Pro Helps ExoMars Get to the Red Planet
URL: http://www.adacore.com/press/ada-on-board-gnat-pro-helps-exomars-get-to-the-red-planet/

Thales Alenia Space implements critical spacecraft software in Ada

CANNES, France & TORINO, Italy & PARIS & NEW YORK, July 19, 2016 - AdaCore today announced that its GNAT Pro Ada cross compilation environment has been successfully used by Thales Alenia Space to develop and verify the critical software embedded on the ExoMars program. Thales Alenia Space implemented in Ada two ExoMars On-Board Software (OBSW) components: one for the Trace Gas Orbiter (TGO) on an ERC32 target, and one for the Entry, Descent and Landing Demonstrator Module (EDM) on a LEON2 target.

Both TGO and EDM OBSW are hosted on their respective centralized hardware platforms. Each OBSW component is a self-contained piece of software handling all Platform applications and performing Payload interface management functions, which are necessary to fulfill the mission objectives under the satellite-specific operational conditions. The OBSW has been developed following a layered and modular architecture to facilitate an incremental development and verification process, to improve the management of multi-team development, to maximize reuse and to ease maintenance.

Thales Alenia Space also uses Ada for its internal real-time kernel product used to ensure a higher safety-critical level with a small memory footprint and good performance.

The development process has been performed according to the European Space Agency (ESA) ECSS standards, reaching compliance with criticality level B.

In addition to the compilation tools, several AdaCore tools were successfully used, including the GNAT Programming Studio (GPS) Integrated Development Environment (IDE) and the static analysis tool GNATstack.

The ExoMars program marks a continuation of GNAT Pro and Ada's long and successful history in space applications, helping software developers to achieve the high reliability required in that domain.

“We are very pleased to have helped Thales Alenia Space go to the Red Planet thanks to our high-integrity tools for the Green Language,” said Cyrille Comar, AdaCore President. “Long-term space projects such as ExoMars gain particular benefits -- higher assurance with lower development and verification cost -- from our open-source tools and libraries that have been adapted to meet the most stringent certification requirements.”

“AdaCore has a longstanding business relationship with Thales Alenia Space,” said Laurent Scarfo, Thales Alenia Space OBSW project manager. “We started working together in 2007 with OBSW embedded in a satellites constellation. Now, AdaCore is flying successfully to Mars!”

DAB Decoder

From: Jan van Katwijk <j.vankatwijk@gmail.com>
Date: Thu, 18 Aug 2016 02:00:00 -0700
Subject: feedback asked on dab-decoder software in Ada
Newsgroups: comp.lang.ada

Last years I did some programming in C++ on SDR-type software. One of the programs is a decoder for DAB(+) signals. This summer I wanted to learn Ada (again, after a period of well over 20 years) and I made a reimplementation of the DAB software in Ada.

The resulting Ada program is limited compared to the C++ one in that it is not possible to change dynamically device and DAB mode, and the GUI - I used GtkAda - is pretty limited. It does work however fine. Supported input devices are the common dabsticks, the SDRplay device and the airspy. Output is currently using a "default" output channel on the PC.

I would like to get some feedback on the use of the Ada language. I am using bindings to C (some libraries are in C), callbacks from C libraries and quite some tasking.

The sources are available on github:

```
git clone
https://github.com/JvanKatwijk/ada-dab
```

Any feedback and suggestions for improvement (it definitely runs slower than the C++ version) is welcome

SparForte

From: Ken O. Burtch <koburtch@gmail.com>
Date: Sat 3 Sep 2016
Subject: sparforte 2.0 release candidate
URL: https://github.com/kburtch/SparForte

Shell, web engine, scripting language mission-critical, scalable projects.

[...]

Ken O. Burtch is the author of "Linux Shell Scripting with Bash" and former IT Architect with the "Webkinz" brand websites. With nearly 20 years experience in the IT business, including many years with multi-billion dollar companies, Ken was concerned over hard to scale, hard to maintain scripting languages, he created SparForte as a tool to solve real business problems. Based on a ISO standard proven effective for large, mission-critical projects, SparForte is designed for fast development of large projects while, at the same time, providing easier maintenance and bug removal.

[See also “SparForte”, AUJ 35-4, p. 220. —sparre]

Steam Sky

From: Bartek Jasicki <thindil@laeran.pl>
Date: Thu 15 Sep 2016
Subject: show only tradeable items on list
URL: https://github.com/thindil/steamsky

Roguelike in sky with steampunk theme

General Info

Steam Sky is an Open Source roguelike steampunk game. Your role is to command flying ship with crew in the sky, traveling between floating bases, fighting with enemies, trade in goods, etc. The game is in a very early stage of development, so at this moment most functions are not implemented yet. For now the game has only been tested 64-bit Linux systems.

Build game

To build it, you need:

- Any Ada language compiler, for example GCC with enabled Ada support or GNAT: http://libre.adacore.com/download/
- ncurses Ada binding (should be available in most distributions or with ncurses package or as standalone package). If not, you can download it from: http://invisible-island.net/ncurses/ncurses-Ada95.html
- optional, but highly recommended: gprbuild program - should be available in most distributions, if not, download from: http://libre.adacore.com/download/

If you have all, in main source code directory type:

- if you don't have gprbuild: gnatmake -j0 -P steamsky.gpr for debug build or for release version: gnatmake -j0 -P steamsky.gpr -XMode=release
- if you have gprbuild: gprbuild -j0 steamsky.gpr for debug mode build or for release mode: gprbuild -j0 steamsky.gpr -XMode=release

Running game

To run game need only ncurses library, available in all Linux distribution. Enter bin directory (if you build game from sources) or in main game directory (if you use released binary) and type ./steamsky. Game works only in terminal.

Note: If you build game from source, copy license file COPYING to bin directory.

That's all for now, as usual, probably I forgot about something important ;)

x_Cleaner

From: George J <ivanov_george@list.ru>
Date: Tue, 23 Aug 2016 01:18:52 -0700
Subject: x_Cleaner v1.2 available
Newsgroups: comp.lang.ada

It's my first opensource project, so I don't fully understand, how to do it right) and I want only say that it's x_Cleaner, realized with Ada using Win32Ada and GWindows modules, licensed with GPL v3.

x_Cleaner is Win32 based app. It can erase data from storage with next algorithms (at this moment):

1. Britain HMG IS5 - Base (one pass writing 0x0)
2. Britain HMG IS5 - Extended (1-st pass 0x0, 2-nd pass 0x1 and 3-d pass - random values)
3. Russian GOST R-50739-95 (1-st pass 0x0, 2-nd pass-random values)
4. Bruce Schneier's algorithm (1-st pass 0x1, 2-nd pass 0x0, 3-7 passes - random values)

Download path:

https://sourceforge.net/projects/x-cleaner/

BTW I want to thank especially Dmitry A. Kazakov, Gautier de Montmollin, Aurele, Simon Wright, Remy Lebeau, Mark Hall, @andlabs!

And thanks to all for the help! Hope it will be useful to somebody!

From: George J <ivanov_george@list.ru>
Date: Tue, 30 Aug 2016 02:14:35 -0700
Subject: x_Cleaner v 1.2.2 available
Newsgroups: comp.lang.ada

[...]

From: George J <ivanov_george@list.ru>
Date: Fri, 9 Sep 2016 01:30:27 -0700
Subject: x_Cleaner v 1.2.3 available.
Newsgroups: comp.lang.ada

x_Cleaner erases stored data using the following algorithms*:

1. Britain HMG IS5 - Base (one pass of writing 0x0)
2. Britain HMG IS5 - Enhanced (1-st pass 0x0, 2-nd pass 0x1 and 3-d pass - random values)
3. Russian GOST R-50739-95 (1-st pass 0x0, 2-nd pass-random values)
4. US DoD 5220.22-M(E) (1-st pass 0x0, 2-nd pass 0x1 and 3-d pass - random values)
5. Bruce Schneier's algorithm (1-st pass 0x1, 2-nd pass 0x0, 3-7 passes - random values)

x_Cleaner works for Windows XP, Vista,7 and later versions.

Important!

Run x_Cleaner with administrative rights in order to clean non-remo

Download path: https://sourceforge.net/projects/x-cleaner/

Changelog:

Version 1.2.1 :

1. Improved writing methods (inlining in task).

Version 1.2.2 :

1. Improved progress bars and added support to Windows Vista,Seven and newer with using UXTHEME.DLL for progress bar themes.

Version 1.2.2.1 :

1. Improved progress bars not to appear if 0 % value.

Version 1.2.3 :

1. Released US DoD 5220.22-M(E) data sanitization method.
2. Corrected algorithm name "HMG IS5 (Enhanced)".

Ada in Context

A Universal_Integer Trick

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Tue, 19 Jul 2016 18:04:15 +0200
Subject: Re: Generic formals and Aspects
Newsgroups: comp.lang.ada

```
>     Long_Long_Integer (Destination_Type'First)
>         >= Long_Long_Integer (Source_Type'First)
>     and Long_Long_Integer (Destination_Type'Last)
>         <= Long_Long_Integer (Source_Type'Last);
```

Tip: you don't need to convert to Long_Long_Integer here (non portable, etc...). Use:

```
Destination_Type'Pos
          (Destination_Type'First)
   >= Source_Type'Pos
          (Source_Type'First)
   and Destination_Type'Pos
          (Destination_Type'Last)
   <= Source_Type'Pos
          (Source_Type'Last);
```

Since 'Pos returns Universal_Integer, you can compare values of different types. Generic Formals and Aspects

From: Oliver Kellogg <olivermkellogg@gmail.com>
Date: Tue, 19 Jul 2016 08:49:30 -0700
Subject: Re: Generic formals and Aspects
Newsgroups: comp.lang.ada

[...]

Here is my use case:

```
with Interfaces;
generic
  type Discrete_Type_16_or_32_or_64 is
          (<>);  -- CANDIDATE

package Big_Endian_Integer_Buffer is

  function Get return
          Discrete_Type_16_or_32_or_64;
  procedure Set (Value :
          Discrete_Type_16_or_32_or_64);

  Size_In_Bytes : constant Positive :=
    Discrete_Type_16_or_32_or_64'Size / 8;

  type Buffer_Type is array
          (1 .. Size_In_Bytes) of
          Interfaces.Unsigned_8;
  for Buffer_Type'Component_Size use 8;

  Buffer : aliased Buffer_Type :=
          (others => 0);

end Big_Endian_Integer_Buffer;
```

At the line marked CANDIDATE, I would have liked to write something like

```
  type Discrete_Type_16_or_32_or_64 is
          (<>)
   with Static_Predicate =>
        Discrete_Type_16_or_32_or_64'Size
          in 16 | 32 | 64;
```

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 19 Jul 2016 14:23:31 -0500
Subject: Re: Generic formals and Aspects
Newsgroups: comp.lang.ada

>> [...]

>> type Internal is new Discrete_Type

>> with Static_Predicate => Internal'Size in 16 | 32 | 64;

You could use a subtype here, if you don't want a new type:

```
subtype Internal is Discrete_Type
   with Dynamic_Predicate => Internal'Size
          in 16 | 32 | 64;
```

BUT:

'Size shouldn't be allowed in either of these predicates, because "Internal" is a value (the value of the "current instance" of the subtype), while Size is the attribute of a subtype or object. (See AI12-0068-1.) This is necessary so that the properties of the object can't be queried in a predicate; that wasn't the purpose of predicates and it would allow some truly bizarre uses. (See "Zoofable" in the question of that AI.) Specifically, 8.6(17.1/4) says:

Within an aspect_specification for a type or subtype, the current instance represents a value of the type; it is not an object. The nominal subtype of this value is given by the subtype itself (the first subtype in the case of a type_declaration), prior to applying any predicate specified directly on the type or subtype. If the type or subtype is by-reference, the associated object with the value is the object associated (see 6.2) with the execution of the usage name.

AARM Ramification: For the purposes of Legality Rules, the current instance acts as a value within an aspect_specification. It might really be an object (and has to be for a by-reference type), but that isn't discoverable by direct use of the name of the current instance.

Looks like an ACATS test is needed.

[...]

> However, it runs without failure. (I would have expected a failure on the Fail instantiation.)

Did you remember to enable assertions? GNAT has the Assertion_Policy as Ignore by default. (This is implementation-defined in the Standard, mainly because we didn't have enough votes to make GNAT change.) If you're depending on assertions (like a predicate), you always need to appropriately place a Assertion_Policy pragma (or the equivalent command-line option, whatever it is).

A Bit of History

From: Robert I. Eachus <rieachus@comcast.net>
Date: Mon, 22 Aug 2016 16:14:50 -0700
Subject: Re: Could you write a BSD like os in ADA?
Newsgroups: comp.lang.ada

[...]

What is really relevant to this group is that Ada would not have existed without Multics (and e-mail). The Ada language was developed on several Multics machines, and as a result all Ada developers knew the Multics way of thinking.

The most important thing (IMHO) that Ada got from Multics was the idea that the OS or compiler should do all the work of maintaining a consistent source tree. I never got used to linkers, writing make files, etc. (Multics did not require linking. If you ran an executable, and it called another unit it would dynamically link it in. If it didn't exist? Multics would tell you and allow you to write it, compile it, resume your program and it would use the unit you just created.)

Why wasn't there an Ada compiler for Multics? At Honeywell Small Systems (and other names) where I worked we were tracking the development of Ada with a compiler that ran on Multics and generated code for DPS6 and other small systems. The problem was that our compiler was intended as a systems development tool. As a result it could ride roughshod over the OS. Not only did it "know" how to access the OS internals, it was used to develop things like the e-mail system. All OS internal calls could be made from within Ada/SIL (for systems implementation language).

We arranged for Dansk Datamatik to port their compiler to our system, and we validated it. Large systems decided to also port the DDC compiler, but to GCOS-8, not Multics. I never found out if they finished, but it was years late...

Address Overlays

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 31 Aug 2016 06:01:36 -0700
Subject: for X'Address use - and Volatile
Newsgroups: comp.lang.ada

Consider:

```
X : Integer;
Y : Integer;
for Y'Address use X'Address;
```

The above is a simple overlay, typically used for under-the-table type conversions.

AARM 13.3 says:

"If the Address of an object is specified [...], then the implementation should not perform optimizations based on assumptions of no aliases."

Interestingly, in the above example there are two objects involved in the overlay, yet only one (Y) is affected by this rule (because Address is *specified* only for Y, not for X).

Let's assume that this is an omission and that the intent is that both object (X and Y) should be excluded from such optimizations, otherwise it will not work.

The question is - do we need pragma Volatile on these objects as well?

C.6 (16c/3):

"If for a shared variable X, a read of X occurs sequentially after an update of X, then the read will return the updated value if X is volatile or atomic, but may or may not return the updated value if X is nonvolatile."

My understanding is that Volatile is *not* needed to ensure proper working of this overlay, even though C.6 seems to imply otherwise. My feeling is that C.6 focuses on data sharing between tasks only and in the case of overlays, the lack of non-aliasing optimizations is enough.

The question comes from analyzing of the code which contains such an overlay together with pragma Volatile. This is a single-tasking program.

My feeling is that Volatile is superfluous - unless there are other reasons for it, for example related to I/O register mapping, etc.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 31 Aug 2016 14:36:24 -0500
Subject: Re: for X'Address use - and Volatile
Newsgroups: comp.lang.ada

> [...]

Ada does not now, nor ever has, officially supported overlays. Such code might work on a particular implementation, but it's not portable (even to another version of the same compiler).

Indeed, address clauses ought to be avoided for all but their intended purpose (mapping to hardware registers); for other purposes, better solutions exist (Unchecked_Conversion, Address_to_Access_Conversions, etc.)

In addition, since you didn't declare these objects aliased, the compiler is allowed to optimize them completely away.

> [...]

[...] You ignored the important rule, 13.3(13/3):

If an Address is specified, it is the programmer's responsibility to ensure that the address is valid and appropriate for the entity and its use; otherwise, program execution is erroneous.

The associated AARM note says that "Appropriate for the entity and its use" covers cases like "addresses which would force objects that are supposed to be independently addressable to not be". Since X and Y are independently addressable and there is no way to avoid that, this case *always* will cause erroneous execution.

Compilers (obviously) don't have to protect against that, so any optimization on X is allowed.

Indeed, since X isn't aliased, it's not even required that X'Address is meaningful (it could be in a machine register).

These things *might* work on a particular implementation, but no guarantees.

The correct way to do this is something like:

```
package A2A is new
System.Address_to_Access_Conversions
(Integer);

X : aliased Integer;
Y : A2A.Object_Pointer :=
        A2A.To_Pointer (X'Address);
```

or better still, avoid Address altogther:

```
type Acc_Int is access all Integer;

X : aliased Integer;
Y : Acc_Int := X'Access; -- Or
-- 'Unchecked_Access if accessibility
-- is an issue.
```

(You need the former if the types are different, the latter if not. But Ada doesn't really allow the case with the types being different to be portable in any case - Unchecked_Conversion is needed, and even that isn't certain to be portable depending on the types involved.)

[...]

P.S. Yes, you hit two of my pet peeves about the way some people use Ada. Most compilers (but not Janus/Ada 95) try to support the overlay case because it was common in Ada 83 code -- both of the better alternatives didn't exist until Ada 95. Similarly with the use of Aliased on any stand-alone object that you're planning to take the 'Address of.

Trying to support these things makes Ada optimization many times more complex and substantially less effective than it otherwise could be. I suppose GNAT gets away with it since C has these issues many times worse and thus there already is support for that in the GCC backend. People not using a C backend have no such (dis?)advantage. Grrrrr.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 1 Sep 2016 10:12:28 +0200
Subject: Re: for X'Address use - and Volatile
Newsgroups: comp.lang.ada

> [...] Indeed, address clauses ought to be avoided for all but their intended purpose [...]

Shameless plug:

and this can be detected in AdaControl:

check representation_clauses (overlay);

More Danish Ada Developers

From: Jacob Sparre Andersen
<jacob@jacob-sparre.dk>
Date: Wed Sep 7 2016
IRC-channel: #Ada
IRC-network: Freenode

* sparre has been teaching Ada all day. :-)

< joakim> sparre: great! :-)

< sparre> Yes.

< sparre> Three new engineers at my customer.

< sparre> All with Delphi experience.

< charlie5> nice

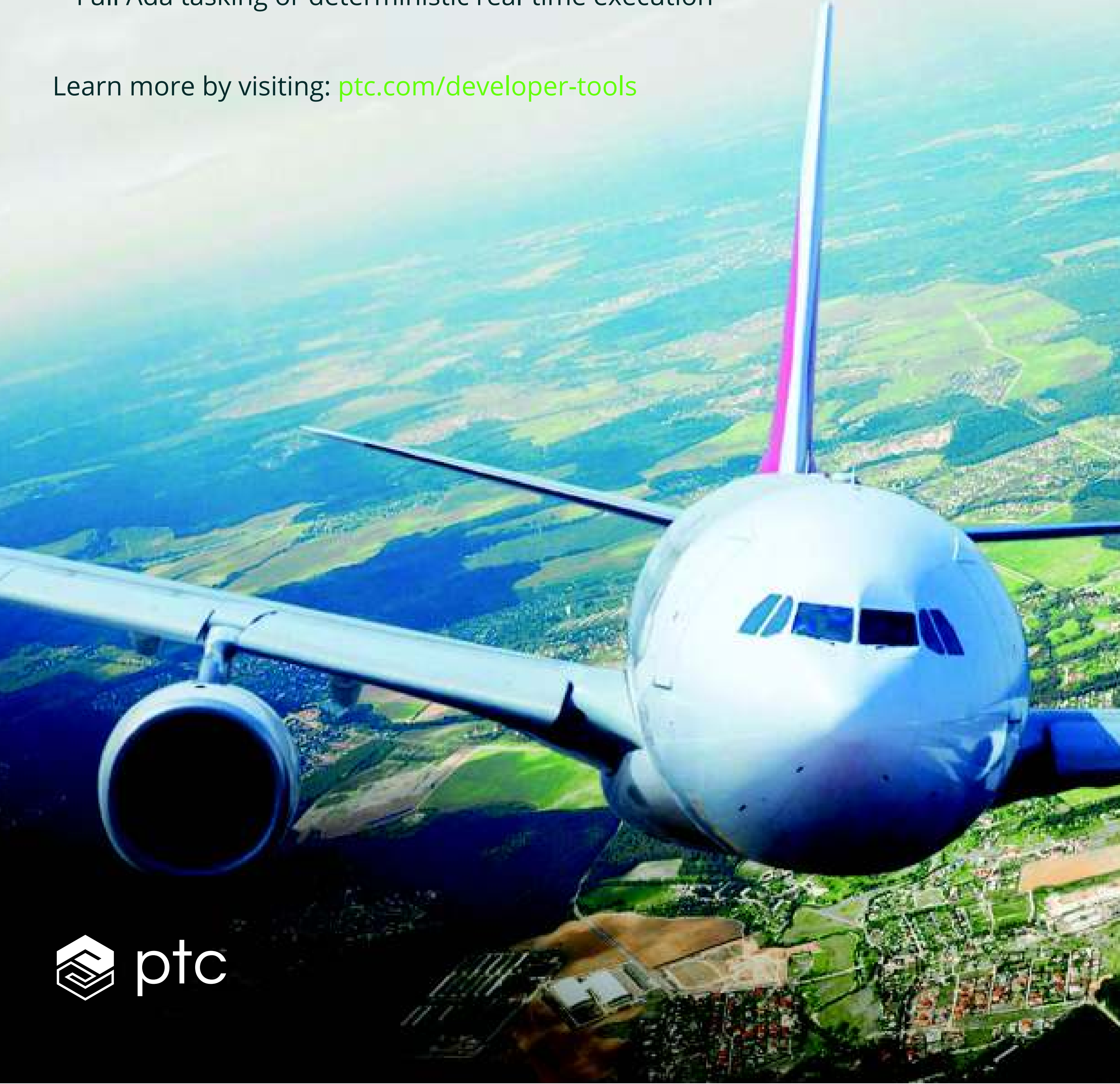
< sparre> We started out with "hello world" in parallel. ;-)

[See http://www.consafelogistics.com/our-offer/warehousing/sattstore-wms for information about the product they are going to work on. —sparre] 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

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.

2016

October 02-10	32nd IEEE **International Conference on Software Maintenance and Evolution** (ICSME'2016), Raleigh, North Carolina, USA. Topics include: reverse engineering and re-engineering, software refactoring and restructuring, software migration and renovation, software and system comprehension, software repository analysis and mining, software testing, maintenance and evolution processes, software quality assessment, continuous integration/deployment, etc.
October 03-06	16th **International Conference on Formal Methods in Computer-Aided Design** (FMCAD'2016), Mountain View, California, USA. Topics include: theory and applications of formal methods in hardware and system verification; synthesis and compilation for computer system descriptions, modeling, specification, and implementation languages, model-based design, correct-by-construction methods; experience with the application of formal and semi-formal methods to industrial-scale designs; etc.
October 06	6th **International Workshop on Design, Modeling and Evaluation of Cyber Physical Systems** (CyPhy'2016), Pittsburgh, Pennsylvania, USA. In conjunction with ESWEEK 2016. Topics include: development of industrial or research-oriented cyber-physical systems in domains such as robotics, smart systems (homes, vehicles, buildings), medical and healthcare devices, future generation networks; evaluation of novel research tools; comparisons of state of the art tools in industrial practice; etc.
☺ October 06-07	ACM SIGAda's **High Integrity Language Technology International Workshop on Model-Based Development and Contract-Based Programming** (HILT'2016), Pittsburgh, Pennsylvania, USA. Sponsored by ACM SIGAda. Co-located with EMSOFT 2016 (ACM SIGBED's International Conference on Embedded Software), part of ESWEEK 2016 (Embedded Systems Week). Topics include: automated analysis and code generation targeting verification-oriented tools and/or programming language subsets (such as SPARK/Ada, ...); contributions linking modeling and contracts to the topics associated with the co-located EMSOFT conference (such as model- and component-based software design and analysis, software technologies for safety-critical and mixed-critical systems, robust implementation of control systems, ...); etc.
October 17-20	14th **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2016), Chiba, Japan. Topics include: program analysis and software verification; analytical techniques for safety, security, and dependability; testing and runtime analysis based on verification technology; analysis and verification of parallel and concurrent hardware/software systems; verification in industrial practice; applications and case studies; etc.
October 19-21	24th **International Conference on Real-Time Networks and Systems** (RTNS'2016), Brest, France. Topics include: real-time system design and analysis: task and message scheduling, modelling, verification, evaluation, model-driven development, timing analysis, worst-case execution time estimation, distributed systems, fault tolerance, quality of service, security; software technologies for real-time systems: compilers, programming languages, middleware and component-based technologies, operating systems, databases; etc.
October 23-27	27th IEEE **International Symposium on Software Reliability Engineering** (ISSRE'2016), Ottawa, Canada. Topics include: reliability, availability and safety of software systems; validation and

verification; software quality and productivity; software security; dependability, survivability, fault tolerance and resilience of software systems; systems (hardware + software) reliability engineering; supporting tools and automation; industry best practices; software standards; etc.

October 24-31 13th **International Colloquium on Theoretical Aspects of Computing** (ICTAC'2016), Taipei, Taiwan, Republic of China. Topics include: principles and semantics of programming languages; relationship between software requirements, models and code; program static and dynamic analysis and verification; software specification, refinement, verification and testing; model checking and theorem proving; integration of theories, formal methods and tools for engineering computing systems; models of concurrency, security, and mobility; real-time, embedded, hybrid and cyber-physical systems; case studies, theories, tools and experiments of verified systems; etc.

October 29-30 5th **International Conference on Software Engineering and Applications** (SEAS'2016), Vienna, Austria. Topics include: software engineering practice, quality management, advanced topics in software engineering, software maintenance and testing, languages and formal methods, software engineering decision making, etc.

☺ Oct 30 - Nov 4 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2016), Amsterdam, the Netherlands. Topics include: all aspects of software construction, at the intersection of programming, languages, systems, and software engineering.

☺ October 30 7th **Workshop on Evaluation and Usability of Programming Languages and Tools** (PLATEAU'2016). Topics include: methods, metrics and techniques for evaluating the usability of languages and language tools; making programs easier to read, write, and maintain; allowing programmers to write more flexible and powerful programs; restricting programs to make them more safe and secure; empirical studies of programming languages; methodologies and philosophies behind language and tool evaluation; software design metrics and their relations to the underlying language; user studies of language features and software engineering tools; critical comparisons of programming paradigms; tools to support evaluating programming languages; etc.

October 31 1st **Workshop on ReUsable and Modular Programming Language Ecosystems** (RUMPLE'2016). Topics include: reusable implementation of runtime components; static and dynamic compiler techniques for different languages; multi-language runtimes and mechanisms for cross-language interoperability between different languages; tooling support for different languages (e.g. debugging, profiling, etc.); modular language implementations that use existing frameworks and systems; case studies of existing language implementations, virtual machines, and runtime components; etc.

Oct 31-Nov 1 9th ACM SIGPLAN **International Conference on Software Language Engineering** (SLE'2016). Topics include: the application of systematic, disciplined, and measurable approaches to the development, use, deployment, and maintenance of software languages, including general-purpose programming languages, domain-specific languages, modeling and metamodeling languages, etc.; language design and implementation, language validation, language integration, language maintenance (software language reuse, language evolution, language families and variability), empirical evaluation and experience reports of language engineering tools; etc.

☺ November 01 **High Integrity Software Conference** (HIS'2016), Bristol, UK.

November 06-08 21st **International Conference on Engineering of Complex Computer Systems** (ICECCS'2016), Dubai, United Arab Emirates. Topics include: verification and validation, security and privacy of complex systems, model-driven development, reverse engineering and refactoring, design by contract, agile methods, safety-critical & fault-tolerant architectures, real-time and embedded systems, cyber-physical systems and Internet of Things (IoT), tools and tool integration, past reflections and future outlooks, industrial case studies, etc.

November 07-11 21st **International Symposium on Formal Methods** (FM'2016), Limassol, Cyprus. Topics include: interdisciplinary formal methods (techniques, tools and experiences demonstrating formal methods in interdisciplinary frameworks); formal methods in practice (industrial applications of formal methods, experience with introducing formal methods in industry, tool usage reports, etc); tools for formal methods (advances in automated verification and model-checking, tools integration, environments for formal methods, etc); role of formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, method integration, qualitative or quantitative improvements); theoretical foundations (all aspects of theory related to specification, verification, refinement, and static and dynamic analysis). Deadline for early registration: October 6, 2016.

November 09-11 **Symposium on Dependable Software Engineering: Theories, Tools and Applications** (SETTA'2016), Beijing, China. Topics include: formalisms for modeling, design and implementation; model checking, theorem proving, and decision procedures; scalable approaches to formal system analysis; integration of formal methods into software engineering practice; contract-based engineering of components, systems, and systems of systems; formal and engineering aspects of software evolution and maintenance; parallel and multicore programming; embedded, real-time, hybrid, and cyber-physical systems; mixed-critical applications and systems; safety, reliability, robustness, and fault-tolerance; applications and industrial experience reports; tool integration; etc. Deadline for early registration: October 10, 2016.

November 13-19 24th ACM SIGSOFT **International Symposium on the Foundations of Software Engineering** (FSE'2016), Seattle, Washington, USA. Topics include: architecture and design; components, services, and middleware; debugging; dependability, safety, and reliability; development tools and environments; distributed, parallel, and concurrent software; education; embedded and real-time software; formal methods; model-driven software engineering; policy and ethics; program analysis; programming languages; refactoring; reverse engineering; safety-critical systems; scientific computing; software evolution and maintenance; software product lines; software reuse; specification and verification; etc.

November 14-18 18th **International Conference on Formal Engineering Methods** (ICFEM'2016), Tokyo, Japan. Topics include: abstraction, refinement and evolution; program analysis; formal verification; model checking; formal methods for object-oriented systems, for component-based systems, for concurrent and real-time systems, for cyber-physical systems, for software safety, security, reliability and dependability; tool development, integration and experiments involving verified systems; formal methods used in certifying products under international standards; formal model-based development and code generation; etc.

November 16-17 **Ada-France at Paris Open Source Summit**. Paris, France.

November 21-23 14th **Asian Symposium on Programming Languages and Systems** (APLAS'2016), Hanoi, Vietnam. Topics include: foundational and practical issues in programming languages and systems, such as semantics, design of languages and type systems, domain-specific languages, compilers, interpreters, abstract machines, program analysis, verification, model-checking, software security, concurrency and parallelism, tools and environments for programming and implementation, etc.

November 22-24 17th **International Conference on Product Focused Software Process Improvement** (PROFES'2016), Trondheim, Norway. Topics include: the challenges of improving software development within the different practice areas such as requirements, design, construction, testing, maintenance, process, methods, management, etc.; research papers based on empirical evidence ranging from controlled experiments to case studies and from quantitative to qualitative studies; etc.

Nov 29 - Dec 02 37th IEEE **Real-Time Systems Symposium** (RTSS'2016), Porto, Portugal. Topics include: all aspects of real-time systems theory, design, analysis, implementation, evaluation, and experiences.

☺ Nov 29 4th IEEE **International Workshop on Real-Time Computing and Distributed systems in Emerging Applications** (REACTION'2016). Topics include: integration of real-time computing and distributed systems in the context of reliable software technologies, real-time middleware, system modeling and component technology, technologies for modeling and programming distributed real-time systems and CPS, etc.

Nov 29 4th **International Workshop on Mixed Criticality Systems** (WMC’2016). Topics include: Task and system models for MCS on single-core, multi-core, and many-core platforms; MCS models (Vestal, DAL / IMA, SIL / AUTOSAR, …); Scheduling schemes and analyses for MCS; operating systems, hypervisors, run-time environments; certification issues of MCS on multi-core and many-core platforms; Safety and fault-tolerance mechanisms for real-time MCS systems; etc.

Nov 29 1st **Workshop on Security and Dependability of Critical Embedded Real-Time Systems** (CERTS’2016). Topics include: Security and dependability of cyber-physical and other real-time and embedded systems; vulnerabilities and protective measures of CPS infrastructure; fault and intrusion tolerant distributed real-time systems; system architectures encompassing combinations of distribution, security, dependability and timeliness; etc.

December 06-09 23rd **Asia-Pacific Software Engineering Conference** (APSEC'2016), Hamilton, New Zealand. Topics include: component-based software engineering; debugging, fault localization, and repair; embedded real-time systems; formal methods; model-driven engineering; parallel, distributed, and concurrent systems; product-line software engineering; programming languages and systems; refactoring; reverse engineering; security, reliability, and privacy; software architecture, modelling and design; software engineering environments and tools; software reuse; testing, verification, and validation; tools and environments; etc.

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

December 12-16 17th ACM/IFIP/USENIX **International Middleware Conference** (Middleware'2016), Trento, Italy. Topics include: design, implementation, deployment, and evaluation of distributed system platforms and architectures for computing, storage, and communication environments; reliability and fault tolerance; real-time solutions and quality of service; scalability and performance; methodologies and tools for middleware design, implementation, verification, and evaluation; retrospective reviews of middleware paradigms; etc.

2017

January 12-14 18th IEEE **International Symposium on High Assurance Systems Engineering** (HASE'2017), Singapore. Topics include: model-driven engineering, design languages, formal methods, domain specific languages, evolution and change, verification and validation, security and privacy, reliability and safety, tools for high assurance systems, etc. Systems of interest include: cyber-physical systems, distributed systems, embedded systems, autonomous vehicles, robot swarms, etc.

January 17-20 9th **Software Quality Days Conference** (SWQD'2017), 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.

☺ January 18-20 44th ACM SIGPLAN **Symposium on Principles of Programming Languages** (POPL'2017), Paris, France. Topics include: all aspects of programming languages and programming systems.

January 22-25 22nd IEEE **Pacific Rim International Symposium on Dependable Computing** (PRDC'2017), Christchurch, New Zealand. Topics include: architecture and system design for dependability; dependability issues in parallel and distributed systems; dependability issues in real-time systems; dependability issues in cyber-physical systems; dependability measurement, modeling, evaluation, and tools; software and hardware reliability; safety-critical systems and software; etc.

January 23-25 12th **European Network on High Performance and Embedded Architecture and Compilation conference** (HiPEAC'2017), Stockholm, Sweden. Topics include: parallel, multi-core and heterogeneous systems; architectural support for programming productivity; reliability and real-time support in processors, compilers and run-time systems; architectural and run-time support for programming languages; programming models, frameworks and environments for exploiting parallelism; compiler techniques; etc.

February 01-03 11th **International Workshop on Variability Modelling of Software-intensive Systems** (VaMoS'2017), Eindhoven, the Netherlands. Topics include: variability across the software life cycle; architecture and design of variable software systems; formal verification, testing, and debugging of variable software systems; refactoring and evolution of variable software systems; reverse engineering of variability; formal reasoning and automated analysis on variability; software economic aspects of variability; etc. Deadline for submissions: October 28, 2016 (abstracts), November 4, 2016 (papers).

February 05-06 26th **International Conference on Compiler Construction** (CC'2017), Austin, Texas, USA. Topics include: work on processing programs in the most general sense, such as compilation and interpretation techniques, run-time techniques (memory management, virtual machines, ...), programming tools (refactoring editors, checkers, verifiers, compilers, debuggers, profilers), techniques for specific domains (secure, parallel, distributed, embedded, ... environments), design and implementation of novel language constructs and programming models, etc. Deadline for submissions: November 1, 2016 (abstracts), November 8, 2016 (papers).

February 19-21 5th **International Conference on Model-Driven Engineering and Software Development** (MODELSWARD'2017), Porto, Portugal. Topics include: domain-specific modeling, general-purpose modeling languages and standards, syntax and semantics of modeling languages, model-based testing and validation, model execution and simulation, model quality, component-based software engineering, software factories and software product lines, etc. Deadline for submissions: October 7, 2016 (regular papers), November 10, 2016 (workshops), November 11, 2016 (position papers), November 28, 2016 (special session), December 14, 2016 (doctoral consortium), January 3, 2017 (tutorials, demos, panels).

February 20-24 24th IEEE **International Conference on Software Analysis, Evolution, and Reengineering** (SANER'2017), Klagenfurt, Austria. 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; experience reports; education; tools and methods; etc. Deadline for submissions: October 12, 2016 (research abstracts), October 17, 2016 (research papers), November 21, 2016 (industrial abstracts), November 26, 2016 (industrial papers), November 28, 2016 (early research achievements abstracts, tool abstracts), December 2, 2016 (early research achievements papers, tool papers), January 8, 2017 (posters).

March 13-18 10th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2017), Tokyo, Japan. Topics include: formal verification and testing, such as model checking; software reliability, security, safety, and trustworthiness; embedded software testing; testing concurrent software; testing large-scale distributed systems; testing real-time systems; testing in multi-core environments; security testing; conformance and interoperability testing; static analysis, code reviews and inspections; testing of open source and third-party software; testing and analysis tools; quality assurance; experience reports; etc. Deadline for submissions: October 7, 2016 (workshops), December 1, 2016 (testing tools papers), December 2, 2016 (doctoral symposium), December 23, 2016 (industry track papers), January 12, 2017 (tool demos), January 29, 2017 (posters).

☺ April 03-06 **The Art, Science, and Engineering of Programming Conference** (Programming'2016), Brussels, Belgium. A new conference, with an associated gold open access journal, created with the goal of placing the art of programming in the map of scholarly works. Topics include: The Art (knowledge and technical skills acquired through practice and personal experiences; examples include libraries, frameworks, languages, APIs, programming models and styles, programming pearls, and essays about programming); Science - empirical (knowledge and technical skills acquired through experiments and systematic observations; examples include user studies and programming-related data mining); Science - theoretical (knowledge and technical skills acquired through mathematical formalisms; examples include formal programming models and proofs); Engineering (knowledge and technical skills acquired through designing and building large systems and through calculated application of principles in building those systems; examples include measurements of artifacts' properties, development processes and tools, and quality assurance methods). Areas include: general-purpose programming, distributed systems programming, parallel and multi-core programming, security programming, interpreters, virtual machines and compilers, modeling and modularity, testing and debugging, program verification, programming education, programming environments, etc. Deadline for submissions: December 1, 2016.

April 03-07 32nd ACM **Symposium on Applied Computing** (SAC'2017), Marrakech, Morocco.

> ☺ April 03-07 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2017). Topics include: aspects and components; code generation, and optimization; distribution and concurrency; formal verification; integration with other paradigms; interoperability, versioning and software evolution and adaptation; language design and implementation; modular and generic programming; runtime verification; secure and dependable software; static analysis; testing and debugging; type systems; virtual machines; etc.
>
> ☺ April 03-07 **Track on Programming Languages** (PL'2017). Topics include: 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, programming languages from all paradigms, etc.
>
> April 03-07 **Track on Software Verification and Testing** (SVT'2017). 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, static and run-time

	analysis, analysis methods for dependable systems, software certification and proof carrying code, real world applications and case studies applying software verification, etc.
April 05-07	IEEE **International Conference on Software Architecture** (ICSA'2017), Gothenburg, Sweden. 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: November 14, 2016 (workshops), December 2, 2016 (tutorials), January 5, 2017 (technical paper abstracts), January 10, 2017 (technical papers), February 18, 2017 (abstracts for industry track, tool papers, New and Emerging Ideas, and Young Researchers Forum), February 23, 2017 (industry track, tool papers, New and Emerging Ideas, Young Researchers Forum, workshop papers).
April 18-21	23rd IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS), Pittsburgh, USA. Topics include: applications, tools, and run-time software for real-time systems; methodologies, algorithms, and analyses that are applied to real systems to solve specific problems; hardware/software co-design, integration methodologies, design-time tools and architectures for modern embedded systems for real-time applications; etc. Deadline for submissions: October 13, 2016 (strict).
April 18-21	8th ACM/IEEE **International Conference on Cyber-Physical Systems** (ICCPS'2017), Pittsburgh, USA. In conjunction with CPSWEEK 2016. Topics include: mechanism design for CPS; model-based design and verification of CPS; etc. Deadline for submissions: October 6, 2016 (abstracts), October 13, 2016 (papers).
April 22-29	20th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2017), Uppsala, Sweden. 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), SV-COMP (Competition on Software Verification). Deadline for submissions: October 14, 2016 (abstracts), October 21, 2016 (full papers).
April 28-29	12th **International Conference on Evaluation of Novel Approaches to Software Engineering** (ENASE'2017), Porto, Portugal. Topics include: application integration technologies, architectural design and frameworks, component-based software engineering, formal methods, model-driven engineering, reverse software engineering, software and system complexity, software and systems development methodologies, software and system quality management, software patterns and refactoring, software product line engineering, software process improvement, etc. Deadline for submissions: November 30, 2016 (regular papers), January 12, 2017 (workshops), January 19, 2017 (position papers), January 30, 2017 (special sessions), February 13, 2017 (special session papers), March 1, 2017 (doctoral consortium papers, open communications papers), March 6, 2017 (tutorials, demos, panels).
May 16-18	9th **NASA Formal Methods Symposium** (NFM'2017), Moffett Field, California, USA. Topics include: identify challenges and provide solutions for achieving assurance for critical systems; model checking; static analysis; model-based development; software and system testing; safety assurance; fault tolerance; compositional verification; design for verification and correct-by-design techniques; applications of formal methods in the development of autonomous systems, cyber-physical, embedded, and hybrid systems, ...; use of formal methods in assurance cases, automated testing and verification, ...; etc. Deadline for submissions: November 28, 2016 (abstracts), December 5, 2016 (papers).
☺ May 20-28	39th **International Conference on Software Engineering** (ICSE'2017), Buenos Aires, Argentina. Deadline for submissions: October 7, 2016 (workshop proposals); October 26, 2016 (Software Engineering in Practice, Software Engineering Education & Training, New Ideas and Emerging Results, Software Engineering in Society); November 18, 2016 (formal demonstrations, technical briefings, Doctoral Symposium); December 28, 2016 (Student Research Competition); January 9, 2017 (posters).
May 29 - Jun 02	31st IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2017), Orlando, Florida, USA.

♦ June 12-16	22nd **International Conference on Reliable Software Technologies - Ada-Europe'2017**. Vienna, Austria. Topics include but are not limited to: Real-Time and Embedded Systems, Mixed Criticality Systems, Theory and Practice of High-Integrity Systems, Software Architectures, Methods and Techniques for Software Development and Maintenance, Formal Methods, Ada Language and Technologies, Software Quality, Mainstream and Emerging Applications, Experience Reports in Reliable System Development, Experiences with Ada. Sponsored by Ada-Europe. This edition of Ada-Europe also features a focused Special Session on Reliable and Safe Robotics. Deadline for submissions: January 15, 2017 (papers, tutorials, workshops, industrial presentations).
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

22nd International Conference on Reliable Software Technologies
Ada-Europe 2017
12-16 June 2017, Vienna, Austria
Copyright: Schaub-Walzer / PID

Conference Chair

Wolfgang Kastner
TU Vienna, Austria

Program Co-Chairs

Johann Blieberger
blieb@auto.tuwien.ac.at
TU Vienna, Austria

Tullio Vardanega
tullio.vardanega@math.unipd.it
Università di Padova, Italy

Special Session Chair

Markus Bader
markus.bader@tuwien.ac.at
TU Vienna, Austria

Tutorial and Workshop Chair

Ben Brosgol
brosgol@adacore.com
AdaCore

Industrial Chair

Jacob Sparre Andersen
jacob@jacob-sparre.dk
JSA Research & innovation, Denmark

Exhibition Chair

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

Publicity Chair

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

Local Chair

Markus Bader
markus.bader@tuwien.ac.at
TU Vienna, Austria

General Information

The **22nd International Conference on Reliable Software Technologies – Ada-Europe 2017** will take place in Vienna, Austria. Following its traditional style, the conference will span a full week, including a three-day technical program and vendor exhibition from Tuesday to Thursday, along with parallel tutorials and workshops on Monday and Friday.

Schedule

15 January 2017	Submission of papers, industrial presentation, tutorial and workshop proposals.
26 February 2017	Notification of acceptance to all authors
19 March 2017	Camera-ready version of papers required
30 April 2017	Industrial presentations, tutorial, and workshop material required

Topics

The conference has over the years become 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 representing industry, academia and government organizations active in the promotion and development of reliable software technologies.

This edition of Ada-Europe features a focused **Special Session on Reliable and Safe Robotics**. Following the increasing trend of robotic systems in industrial and public environments it is more and more important to address software systems to control autonomous vehicles. This special topic discusses issues regarding challenging problems in the field of autonomous navigation and sensor fusion. Topics include (but are not limited to): **Frameworks for robotics, planning and system modelling, as well as multi-agent and logistics applications**.

For the **general track of the conference**, topics of interest include but are not limited to (full list in the website): Real-Time and Embedded Systems, Mixed-Criticality Systems, Theory and Practice of High-Integrity Systems, Software Architectures, Methods and Techniques for Software Development and Maintenance, Formal Methods, Ada Language and Technologies, Software Quality, Mainstream and Emerging Applications, Experience Reports in Reliable System Development, Experiences with Ada.

http://www.ada-europe.org/conference2017

Call for Regular and Special Session Papers

Authors of papers which are to undergo peer review for acceptance are invited to submit original contributions by 15 January 2017. Paper submissions shall not exceed 14 LNCS-style pages in length. Authors for both the general track and the special session shall submit their work via EasyChair at https://easychair.org/conferences/?conf=adaeurope2017. The format for submission is solely PDF.

The International Conference on Reliable Software Technologies is ranked class A in the CORE ranking and Microsoft Academic Search has it in the top third for conferences on programming languages. The conference is listed in DBLP, SCOPUS and Web of Science Conference Proceedings Citation index, among others.

Proceedings

Conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the conference. Camera-ready accepted papers are due strictly by 19 March 2017 (format guidelines available at in the conference site). Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings

Call for Industrial Presentations

The conference seeks industrial presentations which deliver value and insight but may not fit the selection process for regular papers. Authors are invited to submit a presentation outline of exactly 1 page in length by 15 January 2017. Submissions shall be made via EasyChair following the link https://easychair.org/conferences/?conf=adaeurope2017. The format for submission is solely PDF.

The Industrial Committee will review the submissions and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by 30 April 2017, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the *Ada User Journal* (http://www.ada-europe.org/auj/), which will host 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 Tutorials

Tutorials should address subjects that fall within the scope of the conference and may be proposed as either half- or full-day events. Proposals should include a title, an abstract, a description of the topic, a detailed outline of the presentation, a description of the presenter's lecturing expertise in general and with the proposed topic in particular, the proposed duration (half day or full day), the intended level of the tutorial (introductory, intermediate, or advanced), the recommended audience experience and background, and a statement of the reasons for attending. Proposals should be submitted by e-mail to the Tutorial Chair. The authors of accepted full-day tutorials will receive a complimentary conference registration as well as a fee for every paying participant in excess of 5; for half-day tutorials, these benefits will be accordingly halved. The *Ada User Journal* (http://www.ada-europe.org/auj/) will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes that fall 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 should be submitted to the Workshop Chair. The workshop organizer shall also commit to preparing proceedings for timely publication in the *Ada User Journal* (http://www.ada-europe.org/auj/).

Call for Exhibitors

The commercial exhibition will span the three days of the main conference. Vendors and providers of software products and services should contact the Exhibition Chair for information and for allowing suitable planning of the exhibition space and time.

Grants for Reduced Student Fees

A limited number of sponsored grants for reduced fees is expected to be available for students who would like to attend the conference or tutorials. Contact the Conference Chair for details.

Venue

The conference will take place at Palais Eschenbach (see images below), in the heart of Vienna, Austria.

Ada User Journal

The journal for the international Ada community

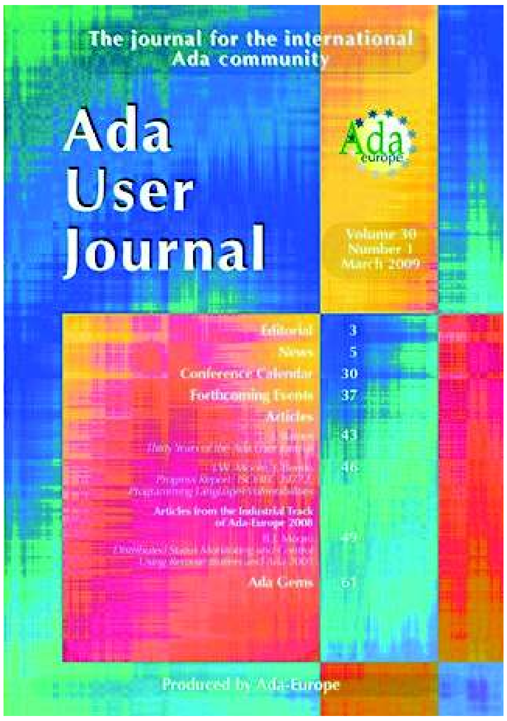
Call for Contributions

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.

Aims

The ***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 in Europe and other parts of the world. The language of the journal is English.

Although the title of the Journal refers to the Ada language, any related topics are welcome. In particular papers in areas related to reliable software technologies.



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.
- ✓ **Information** regarding standards concerning Ada.
- ✓ **Reviews** of publications in the field of software engineering.

Submission Guidelines

All material for publication should be sent to the Editor, preferably in electronic format. Prospective authors are encouraged to contact the Editor (Luís Miguel Pinho <AUJ_EDITOR@ADA-EUROPE.ORG>) to determine the best format for submission.

http://www.ada-europe.org/auj/home

Online Archive available

Proven Test Solutions for Reliable Embedded Software

VECTOR software

"VectorCAST is unique in that it provides us with the ability to increase the reliability and quality of our flight software."

-- Honeywell

vectorcast.com

VectorCAST is a TÜV SÜD Certified Software Tool for Safety Related Development

Vector Software, Inc.
Golden Cross House | 8 Duncannon Street | London WC2N 4JF UK | +44 203 603 0120 | sales@vectorcast.com

Using Ada's Visibility Rules and Static Analysis to Enforce Segregation of Safety Critical Components

Jean-Pierre Rosen

Adalog, 2 rue du Docteur Lombard, 92441 Issy les Moulineaux CEDEX, France;
email: rosen@adalog.fr

Jean-Christophe Van-Den-Hende

ALSTOM Transport, 48, rue Albert Dhalenne, 93482 SAINT-OUEN CEDEX, France;
email: jean-christophe.van-den-hende@transport.alstom.com

Abstract

Segregation of components is required in mixed criticality systems, where different safety integrity levels apply to various components. This paper presents a solution where appropriate organization of the project into child units and proper usage of Ada's visibility rules complemented with simple static analysis are sufficient to ensure that all violations of segregation rules will be rejected at compile time.

This paper provides some explanations about the Ada mechanisms used to that effect, in order to make it understandable by those who are not familiar with the Ada language.

1 The need for segregation

ALSTOM Transport is a leading provider of ground and embedded railway systems. In order to minimize costs as well as to maximize safety, it is developing a new, components based, architecture in Ada that would maximize the possibility of reusing components between various systems.

Railway safety is highly dependent on software; although it is true that a train can stop in an emergency situation (unlike planes), stopping a high speed train (such as the French TGV) with emergency breaking requires three minutes and 3300 meter distance. This is far too much to avoid an accident that would be caused by a software failure, and no manual action of the driver can compensate for a software fault. Therefore, railway systems are subject to very strict rules ensuring correctness of the software.

Railway software is governed by the safety standard EN-50128 [1], which defines five *Safety Integrity Levels* (SIL), ranging from SIL0 (lowest criticality) to SIL4 (highest criticality). This is similar to the "levels" E to A of DO178C [2] for avionics systems. As can be expected, the cost of developing, checking, and certifying SIL4 software is much higher than the one of lower SILs. The necessity of reducing development costs implies that only truly critical parts be subject to the highest criticality checks.

1.1 Mixed criticality systems

In a complex system such as those that ensure safety and correct operation of trains, only a relatively small subset of the functions (and hence associated components) is of a SIL4 level. However, the lower criticality components (considered SIL0 for short) run on the same computer and are part of the same main program as the SIL4 components.

Such systems where components with different safety requirements are running together are called *mixed criticality systems*, whether the components are several applications running on the same computer, or a single application that mixes various software components.

Of course, the difficulty with mixed criticality systems is that a defect in a SIL0 component could adversely affect the behaviour of a SIL4 component. The traditional approach to addressing this issue is to submit all components to the same safety process as required by the highest criticality component in the system - in practice the SIL4 process. While this has the benefit of ensuring the highest confidence in the system as a whole, it has an enormous cost, since the vast majority of components must suffer a costly validation and certification process that goes far beyond what is required for their own criticality.

1.2 Segregation

This cost can be dramatically reduced through *segregation*, i.e. if it can be proven that SIL0 components are independent from SIL4 ones, and that the behaviour of no SIL4 component depends on a SIL0 component. Such a segregation can be achieved through hardware or software control.

For example, in avionics systems (which have similar issues), the ARINC-653 [3] standard has been designed to ensure hardware segregation of components of different levels: the standard ensures that components of different criticalities have different address spaces, and a MMU ensures that each component can access only its own address space. Communications between components are performed through a dedicated bus, etc. Note however that hardware segregation prevents corruption by an incorrect low criticality component at execution time, but does not ensure that the software is free from such errors.

On the other hand, software proofs and other static verification techniques can be used to demonstrate that by design, no low criticality component performs dangerous or incorrect actions that could jeopardize the safety of high criticality components. Of course, to be effective and economical, such proof systems have to be much cheaper than the usual SIL4 validation process.

2 The study and its requirements

ALSTOM wanted to evaluate various solutions to ensure segregation of components, and asked Novasys [4] (part of the Pacte-Novation group) to conduct two studies on solutions using hardware and software segregation respectively. The hardware solution was studied directly by Novasys, while the software solution, which is the purpose of this paper, was conducted by Adalog [5], a subsidiary of Novasys specialized in Ada consultancy, expertise, and training.

2.1 Requirements

A SIL4 component is one which is responsible for actions that can compromise safety, like setting the speed of the train, controlling the opening of the doors, etc. Such components must not only be checked for their own correctness; it is also important to check that they do not use unsafe operations, that their provided operations are not called in an incorrect manner and that they do not operate on incorrect data.

Therefore, the following rules were established as a basis for the software segregation study:

- Data passed from SIL0 to SIL4 components are deemed unreliable; it is up to the SIL4 component to assess the validity of the data.
- Except for the dedicated zones for data exchange, no SIL0 component is allowed to access SIL4 data.
- Some utility components that do not perform any safety critical function can be called by SIL0 as well as SIL4 components; however, since they are used by SIL4 components, they are classified as SIL4.
- If a SIL0 component needs to be called by a SIL4 component, this can be done only through a dedicated SIL4 component that will perform all required checking.
- Except for the special cases above, no SIL4 component or functionality can be used by a SIL0 component.

In addition, low level features of Ada, unchecked programming, and removal of language checks are not allowed in SIL0 components, in order to guarantee memory integrity of the system (see below).

3 A software architecture for statically checking segregation rules

The software study goal was to find a convincing (and economical) way of enforcing the above rules. The study proposed an architecture of the software that would allow checking of the segregation rules by the compiler. In other words, a program that would not obey by the rules would simply not compile. This was made possible by using Ada's visibility rules related to packages and child packages.

3.1 Ada packages and visibility rules

In Ada a *package* is a logical module that gathers a set of logically related elements (types, constants, subprograms…). Like all Ada units, a package has a *specification* and a *body*. The specification exposes the elements that are usable outside of the package, while the body contains the implementation of the services announced in the specification. The specification is furthermore divided into a *visible part* and a *private part*; actually, only elements from the visible part are made available to the outside units. This part can contain *private types* that are announced without revealing their internal structure. The private part of the package serves to give the compiler the full declaration of these types, without making it visible to the users. This allows the definition of *abstract data types*, where only the type name and its operations are made visible, all implementation details being hidden in the private part and in the body. Of course, the body of a package sees the private part, including the full declaration of abstract data types.

The typical structure of a package is shown in the following example:

```
package Example is -- specification
  type T is private;    -- a private type
  procedure P (X : T);  -- operation
private     -- beginning of private part
  type T is -- full declaration of T
    record
      Compo: Compo_Type;   -- Components…
    end record;
end Example;
package body Example is    -- body
  procedure P (X : T) is  -- body of P
    …
  end P;
end Example;
```

Figure 1 Structure of a package

Packages can be organized as a hierarchy of parent/child units. A child package is simply a package whose name is prefixed by the name of its parent. A child package can be either public or private.

- A public child can be accessed normally by the rest of the system; however its visible part has only access to the visible part of its parent[1]. For implementation purposes, its own private part and its body see the private part of the parent.

[1] Consequently, a public child cannot reveal declarations hidden in the private part of its parent.

- A private child is available only to the bodies of its parent and siblings (and descendants). A parent, together with its private children, defines a subsystem, where only the parent interface is available outside the subsystem.

The following example illustrates the declaration (specification) of public and private child packages:

```
-- public child package
package Parent.Pack1 is
  …
end Parent.Pack1;

-- private child package
private package Parent.Pack2 is
  …
end Parent.Pack2;
```

Figure 2 Child packages

3.2 The architecture

As exposed above, Ada features a sophisticated system for controlling visibilities, and therefore the allowed calls between separately compiled modules. The idea of the study was to use these features to provide compile-time enforcement of the segregation rules.

The proposed structure followed the overall general framework exemplified by the following figure:

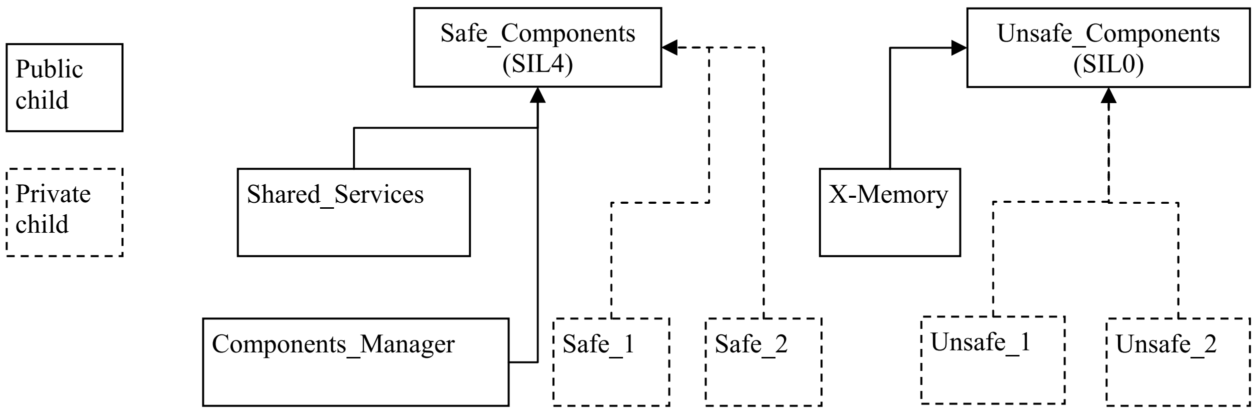


Figure 3 Architecture of the application

In this example, "Safe_Components" and "Unsafe_Components" are empty packages that serve as roots to the SIL4 and SIL0 hierarchies, respectively. "Shared_Services" and "Components_Manager", which are callable from SIL0 components, are public children of "Safe_Components" (thus visible and callable by all components), while SIL4 components are private children (therefore visible and callable only from within the SIL4 hierarchy): with this structure, it is impossible for SIL0 components to call SIL4 components, except for the dedicated and easily identifiable shared components.

Similarly, SIL0 components are private children of "Unsafe_Components", thus preventing them from being called by SIL4 components. On the other hand, the dedicated area for exchange of data ("X-Memory"), which is classified as SIL0 but usable from SIL4 components, is declared as a public child of "Unsafe_Components".

In the few cases where a SIL0 component would need to call a functionality from a SIL4 component, it would do so through an exported service of "Shared_Services", that would either perform the required validation of data, or, if there is no safety issue, simply be a renaming of the underlying (hidden) SIL4 service that remains private.

As far as data are concerned, except for the exchange area ("X-Memory"), no SIL0 variable should be accessible from SIL4 components, and conversely. This is easily obtained by forbidding the declaration of any variable in the visible part of packages (which is, in addition, a generally accepted coding rule, independently of any segregation issue). Possible data shared between components of the same level are placed in private children of "Safe_Components" and "Unsafe_Components".

3.3 Tracing the integrity level of components

In a mixed criticality system, it is important to trace the integrity level of each element, in order to perform checks appropriate to each level. This requires generally extra documentation, check lists, special comments, etc.

Another benefit of this structure is that the classification (SIL4 or SIL0) of components shows directly from the structure of the software; there is no need of maintaining manually a list of components with their assigned safety level. The level of the component appears directly from its Ada name; for example, the full name of the "Safe_1" component, the one given in its declaration, would be "Safe_Components.Safe_1", thus immediately showing that it is a SIL4 component. The list of SIL0 components is simply obtained by filtering all components whose name start with "Unsafe_Components.".[2]

[2] A common convention is to name a file containing a unit with the name of the unit (with some substitutions, like replacing "." with "-"). Some popular compilers enforce this convention. In such a case, obtaining the list of *files* containing SIL0 units is as simple as using the Unix command "ls unsafe_components-*".

Conversely, the simple fact that a component's name starts with "Safe_Components." or "Unsafe_Components." will automatically enforce the corresponding segregation rules.

3.4 Alternative possible architectures

The above described architecture was optimized according to the requirements of Alstom. But many variations on this basic principle of architecture are possible, depending on the constraints of the project. For example, shared component could constitute a hierarchy of their own rather than being under the "Safe_Components" tree[3].

In summary, the basic principles used for achieving segregation, and that Ada rules can enforce, are:

- Every segregated subsystem constitutes a single tree, with an empty root and where every module (except for communication modules) are private child units.
- Communication between modules of different criticality is achieved through public child units. Every communication module needs to be certified at the highest integrity level among its own level and the level of all possible callers.

4 Other necessary checks

Because it is sometimes necessary to escape from common programming rules, often in connection with low level programming such as direct management of hardware, Ada provides so-called *unsafe programming* features. These features include special packages to overcome normal type checking and provide direct access to memory, and *pragmas* for the removal of mandatory compiler checks (such as array overflow control). Malicious use of these features could be used to defeat the controls provided by the above structure, therefore their use is not allowed in SIL0 components[4].

In a safety critical system, it is not sufficient to have a programming standard that forbids such features; it must be *proven* that they are effectively not used. In Ada, any compilation unit that requires the use of a package must name it in a special clause (a *with clause*), therefore ensuring that any dependency between units is explicitly stated – and this applies to predefined packages as well. Removal of language checks requires the use of special pragmas. Therefore, it is sufficient to make sure that there is no **with** clause naming one of the unsafe programming packages and no use of the special pragmas to ensure that the safety features of the language are effective.

Checking these rules is easily achieved with static analysis tools. One of these tools is Adalog's AdaControl tool [6][7][8], a free static rule checking tool whose rich set of rules covers all the necessary restrictions.

Finally, some constructs that are normally allowed by the language were forbidden by the constraints of the project, such as the declaration of variables in the visible part of packages. This can be checked by manual inspection; however AdaControl is also able to check these automatically, which is always preferable to human (and therefore fallible) inspection.

In addition, the study analyzed (existing) ALSTOM's coding standard to determine which SIL4 rules were applicable to SIL0 components in order to allow cohabitation, and all applicable rules were also found checkable with AdaControl.

Conclusion

In conclusion, the appropriate use of visibility rules related to public and private children allowed the definition of a structure where segregation rules are enforced by the compiler.

The remaining safety constraints were checked automatically by a static analysis tool (AdaControl), thus allowing cohabitation of SIL4 and SIL0 components without loss of safety, and with a considerable economic gain compared to solutions that involve hardware segregation, or full certification at SIL4 level of SIL0 components.

As an additional benefit, the structure allows easy tracing of the integrity level of each component.

References

[1] CENELEC (2011), *EN50128:2011 Railway Applications -Communications, signaling and processing systems.*

[2] DO-178B: Software Considerations in Airborne Systems and Equipment Certification (1992).

[3] ARINC 653 - Avionics Application Software Standard Interface (2010).

[4] http://www.novasys-ingenierie.com/

[5] http://www.adalog.fr/en/

[6] J-P. Rosen (2005), *On the benefits for industrials of sponsoring free software development*, Ada User Journal, Volume 26, n° 4.

[7] J-P. Rosen (2006), *AdaControl: a free ASIS based tool*, presentation at FOSDEM, Brussels, Belgium.

[8] M. Jemli and J-P. Rosen (2010), *A Methodology for Avoiding Known Compiler Problems Using Static Analysis*, in Proc. of the ACM SIGAda Annual International Conference (SIGAda 2010), ACM Press, ACM order number 825100, Fairfax, USA.

[3] This possibility was not retained because Alstom wanted to have all units requiring SIL4 verification under the same root.

[4] They are allowed in SIL4 components, since those are subject to extensive reviews to make sure that the features are used only appropriately.

Fitting the CHESS Approach to the AUTOSAR Development Flow

Silvia Mazzini, Stefano Puri, Andrea Russino

Intecs, via Umberto Forti, 5 Montacchiello – I-56121 Pisa, Italy;
email: silvia.mazzini@intecs.it, andrea.russino@intecs.it, stefano.puri@intecs.it

Abstract

Enabling early validation of extra functional properties can play an important role in the development process of reliable and safe cyber physical systems (CPS); this is particularly relevant in the automotive industry where the ever-increasing need to reduce electronic control unit (ECU) production development cost, enhance quality, and shorten the development cycle time is a crucial aspect. Building on the results of the recently ended CONCERTO project, CHESS pursues this objective by proposing a methodology and technology which fits into the well-defined AUTOSAR workflow.

Keywords: AUTOSAR, model-based, real-time, safety, component-based, correctness-by-construction, separation of concerns, CPS, schedulability analysis, multi-core.

1 Introduction

Being able to develop critical dependable software systems in the face of the speedup of technological progress and of time to market in the automotive domain is a quite complex task.

The fulfilment of extra-functional properties of the software architecture is a very important goal in the automotive domain, where for instance execution time of control loops or end-to-end scenarios must be bounded in order to guarantee the safety properties of the system.

The approach of ensuring extra-functional constraints through testing and corrections is resource-consuming and not exhaustive, and may easily become impractical with the increase of systems complexity and heterogeneity.

Schedulability analysis for example is a common practice which makes it possible to statically estimate worst case response time of the software applications running on a given execution platform. Schedulability analysis can be applied at different abstraction levels; in particular, the application of schedulability analysis in the early phase of the design makes it possible to avoid late discovery of bad software and architectural design choices and thus to avoid high costs of redesign and delay of the product availability.

Early analysis support fits well in the context of the model driven engineering (MDE) approach, the latter providing formal and semantically grounded support for the design of the system, capable of capturing the overall characteristics as well as detailed properties of all its composing parts.

One of the main challenges in MDE is to ensure that the analysis model can be automatically derived starting from the modelled system; moreover the approach should guarantee that the assumptions made by the analyzed model at a given step of the design are preserved during automatic model refinements, thus implementing a correct-by-construction development process.

AUTOSAR (AUTomotive Open System ARchitecture) [1] is an open and standardized software architecture and methodology for automotive applications, jointly developed by automobile manufacturers, suppliers and tool developers with the following aspects in mind: scalability to different vehicle and platform variants, integration of functional modules from multiple suppliers, maintainability, and transferability of functions throughout network.

CHESS is a methodology and supporting toolset which is the principal result of several R&D projects, starting from the original CHESS (Composition with Guarantees for High-integrity Embedded Software Components Assembly) ARTEMIS JU Call 2008 project [2], to provide a model-based solution to address the challenges of developing critical real-time and embedded software systems, by adopting a component-based approach, across several domains of interest.

A distinct and publicly acknowledged limitation of the AUTOSAR workflow is the treatment of extra-functional requirements [2][4].

Interestingly, CHESS extends Model-Driven Engineering practices and technologies specifically to address extra-functional concerns, and it does so in a manner that promotes a correct-by construction approach to software production.

The above observations prompted the exploration of whether and how the CHESS development method and technology would fit into AUTOSAR, and provide benefits to it.

1.2 Outline

In this paper first we introduce the relevant aspects of the AUTOSAR and CHESS approaches. Then we discuss commonalities and differences between the two modelling languages, with the purpose of defining a suitable integration between the two workflows. The feasibility of our investigation is then evaluated through a case study.

2 Background

2.1 AUTOSAR

All the overarching AUTOSAR objectives are met with a well-defined workflow, which provides, in a stepwise fashion, the appropriate level of abstraction to system construction, starting from the definition of the software architecture, proceeding with the definition and configuration of the specific system architecture, the implementation, and finally the generation of the software executables. The decoupling of the application architecture from the target hardware, and the flexibility to integrate multiple applications (possibly produced by different suppliers) on one and the same electronic control unit (ECU) are central aspects of AUTOSAR.

The AUTOSAR methodology comprises the following steps:

Application Configuration: During this phase, the application is specified in terms of the software architecture: software components, interfaces, ports and connectors. The platform is specified in terms of hardware resources: electronic control units and their interconnection topology, i.e. physical ECUs interconnection through buses or dedicated links, peripherals, sensors and actuators. The mapping of software components on ECUs is not done during this phase, but constraints on this mapping can be specified at this level. The Application Configuration models are exchanged through an XML artefact called System Configuration Input, which actually serves as input for the following phase.

System Configuration: During this phase the mapping of the software architecture into the hardware architecture is performed. Software components are mapped into ECUs, and application messages are mapped into bus frames. The artefact to be produced at the end of this phase is called System Configuration Description, which serves as input for the following phase.

ECU specific information extraction: During this phase, information specific to each ECU is automatically extracted, and a first layer of run time environment (RTE) is automatically generated. The artefact to be produced at the end of this phase is called Extract of System Configuration Description, which serves as input for the following phase.

ECU configuration: During this phase, the basic services of the platform are configured on each ECU. The most important step lies in the specification of the mapping of runnable entities into operating system (OS) tasks. The artefact to be produced at the end of this phase is the ECU Configuration Description. This artefact is used for the generation of binary code.

2.2 CHESS

CHESS is a cross-domain model-based methodology and toolset for developing, analysis and implementation of critical real-time and dependable embedded software systems [13]. The correctness-by-construction principle is one of the key foundations of the CHESS methodology [14]. This principle allows for early assertion and verification (predictability) of system non-functional properties, like timing, at model level, assuring also fully automated code generation with guarantees for property preservation and monitoring of these asserted properties at run time.

The modelling language of CHESS [17] is implemented as a UML [12], MARTE [8] and SysML [9] profile. The profile comes with the set of constraints which makes it possible to support the CHESS model driven methodology. Few stereotypes extending UML entities have been introduced to support the CHESS component model, while a couple of stereotypes (CHRtSpecification and CHRtFeature) extending MARTE entities (RtSpecification and RtFeature) have been defined to allow deployment of timing contracts on UML provided ports of the component instances. Regarding dependability, a dedicated profile has been defined in CHESS as an extension of UML entities to basically allow modelling of failure modes for components, their propagation inter and intra components and their qualitative and quantitative aspects.

The modelling language makes it possible to define: the user model, which corresponds to a platform independent model, PIM, in the jargon of the model-driven architecture initiative; the description of the hardware platform; and the platform specific model, PSM, which represents the model-level description of the implementation of the PIM on the given platform. The design space supports the creation of the user model through several design views and with the aid of model validation.

The software is modelled by using a specific component model, built around the concepts of *components*, *containers* and *connectors,* that supports the separation of concerns principle, strictly separating the functional aspects of a component from the non-functional ones. According to this model, a component represents a purely functional unit, whereas the non-functional aspects regarding dependability and timing concerns are in charge of the component's infrastructure and delegated to the container, while connectors are responsible for the communication between containers.

From the interaction perspective, components are considered as black boxes that only expose their provided and required interfaces. Non-functional attributes are specified by decorating the provided UML ports (i.e. interfaces) of the components instances with non-functional properties (e.g., for real-time concerns, a real-time activation pattern for an operation).

The declarative specification of non-functional attributes of a component, together with its communication concerns, declared by the user at the PIM level, are used in CHESS for the automated generation of the containers and connectors that embody the system's infrastructure, at the PSM level. The design space supports also automated model transformations from the PSM to the model-based analysis tools, and the code generation to create a property-preserving implementation of the system and the properties asserted at design level.

One of the aspects of the CHESS methodology is that the PSM is read-only and conforms to the same modelling language of the PIM: the user is allowed to explore it but not to modify the properties defined at PIM level so as to jeopardize the property preservation principle that is one of the pillars of the CHESS methodology.

The user then can trigger the execution of various model-based analyses, in particular dependability and timing analysis, at PIM or PSM level. Depending on the analysis, the model is transformed through a series of transformations into a proper input for the analysis tool of choice. The results of the analysis tool are then propagated back to the PIM user model by means of back-propagation transformations that use traces generated by the source transformations.

For schedulability analysis, the PSM includes the schedulability analysis model (SAM). The analysis is executed by using the open source tool MAST, a scheduling and timing analysis tool developed and maintained by the Universidad de Cantabria [5]; in particular an extension of MAST has been implemented in CONCERTO to allow analysis of software allocated on multi-core architectures. When the user requests timing analysis, automatic model transformation from CHESS SAM to the semantically equivalent MAST analysis model and the execution of the MAST analysis are automatically performed.

With respect to dependability analysis, CHESS supports quantitative state based analysis, performed via integration with the DEEM server [15], and qualitative failure logic analysis [16].

Once the results of the analyses are satisfactory, the user can trigger the code generation from the PSM for a specific platform and industrial domain. The code generation follows the component model approach: components address only functional and algorithmic concerns and originate from the specification at design level; containers address extra-functional concerns, in particular those related to concurrency; the connectors that implement interactions among components through a mediated communication among containers; containers and connectors pertain only to the implementation space. Containers expose the same provided and required interfaces of the enclosed components, through an interface promotion that creates delegation and subsume relationships from the operations of the component to the equivalent operations on the container. This approach promotes the isolation of components so they need to know only the required interfaces to interact with other components. Moreover, extra-functional concerns are dealt with only by containers, connectors and the execution platform (via containers). Code generation produces infrastructural code and/or functional code: the infrastructural code contains all but the functional implementation of components. The infrastructural code can later be complemented with handwritten functional code.

CHESS has been recently improved in the CONCERTO[1] project (ARTEMIS JU Call 2012) [7] which has consolidated and extended the CHESS component-based language, methodology and related tool support for the modelling and development of high-integrity multi-core systems. In particular CONCERTO investigated and realized specific support for petroleum, telecare, avionics and automotive domains, the latter discussed in this paper.

CHESS and CONCERTO results are currently hosted in the Eclipse PolarSys open source ecosystem [6].

3 CHESS and AUTOSAR comparison

In order to investigate similarities and differences between CHESS and AUTOSAR, we first defined a mapping between the entities defined in the CHESS component model and modelling language to semantically equivalent entities available in AUTOSAR.

This allowed us to identify feasible integration between CHESS and AUTOSAR workflows.

3.1 Software Components

CHESS supports a domain-neutral component model, which enables hierarchical composition of components, and interaction among them via ports that implement interfaces. Analogous concepts exist in the AUTOSAR component model, although with slightly different and (obviously) highly domain-specific interpretations.

In both CHESS and AUTOSAR approaches, the description of a software-component is the sum of different but inter-related parts: hierarchical structure, ports and interfaces, internal behaviour, implementation (object code or source code).

AUTOSAR defines the concept of composite component: as it holds in CHESS, a composite component can only act as container of internal (composite or atomic) components. In other words, the composite component has to delegate all the declared implementation to the internals. The concept of composite component of AUTOSAR is covered by the CHESS component entity; in fact, the CHESS modelling language does not provide explicit constructs to represent atomic or composite components.

[1] "Guaranteed Component Assembly with Round Trip Analysis for Energy Efficient High-integrity Multi-core Systems"

As in CHESS, AUTOSAR defines assembly and delegation connectors; in both CHESS and AUTOSAR, connectors must connect component ports.

AUTOSAR software components, whether atomic or composite, can only communicate through ports independently of their physical allocation (i.e. whether they are on the same ECU or on different ECUs), as it holds in CHESS; in AUTOSAR this is allowed thanks to the notion of the Virtual Functional Bus (VFB) provided by the run-time environment. The run-time environment has the responsibility to provide a uniform environment to AUTOSAR software components to make the implementation of the software components independent from communication mechanisms: the notion of run-time environment is captured in CHESS under the umbrella term PSM, which encompasses all implementation artefacts and concerns, including the run-time support.

In summary, we may conclude that even if different support is available in AUTOSAR and CHESS to represent components, their definition, implementation and instantiation, the CHESS component entities, at type and instance level, provide sufficient coverage of the semantics of the AUTOSAR software component.

3.2 Software components interactions

AUTOSAR and CHESS share the same concept of component ports; in fact, in both languages a port belongs to exactly one software component and represents a point of interaction of that component.

In AUTOSAR, the kind of interaction that can occur through a given port is defined by the AUTOSAR interface that is declared to be provided or required by the port itself. The AUTOSAR interface can be client-server, sender-receiver or mode-switch; in the UML/MARTE implementation of CHESS, the data ports and operations ports are distinct constructs available in the modelling language.

Sender-receive interactions in AUTOSAR represent data flow based interactions.

CHESS does not allow for data flow between components to be realized through flow ports. Data flow ports are allowed for a given component only to represent the sending/receiving of events to/from the run-time environment. This is however a minor point, as inter-component data flow can be realized through operation calls, although with a slightly different behaviour[2].

Client-server ports in AUTOSAR correspond to CHESS interface ports. AUTOSAR allows both synchronous and asynchronous communication between client and server, while in CHESS, concerning threaded operations, only asynchronous invocation of operations is allowed; this is an important constraint to allow the application of the theories allowing timing analysis.

AUTOSAR and CHESS equally support the concept of operational modes. In AUTOSAR, a mode switch interface represents an interface through which mode switch requests can occur. Mode switch interactions are available in CHESS by reusing MARTE support, in particular by using data flow or operations ports; in CHESS, the arrival of events to data flow ports (event receiver ports), or the arrival of messages to interface ports can be used to trigger changes in operational modes. Hence, the (tiny) difference between CHESS and AUTOSAR in this regard is that in CHESS the kind of port through which requests for mode switches can occur is not defined at language level.

3.3 Component Internal Behaviour

An AUTOSAR application component (ApplicationSWComponentType) comes with one or more internal behaviours (SwcInternalBehavior). An internal behaviour represents an implementation of a given atomic (i.e. not decomposed) software component; however, the internal behaviour does not describe the detailed functional behaviour of the component.

The internal behaviour describes the dynamic functionalities of an application software component, in the functional and timing dimension; as further elaborated in the following subsections, we can state that AUTOSAR SwcInternalBehavior is sufficiently covered by the CHESS component.

3.3.1Runnables

AUTOSAR runnable is the smallest unit that can be scheduled and executed by the OS; it corresponds to a component operation in CHESS.

4.3.2 Run-time environment (RTE) events

Run-time environment (RTE) events represent conditions to start or resume the execution of a runnable. The RTE event must have:

- A type (data received, data receive error, data send completed, operation invoked, asynchronous server call returns, mode switch, an ack to a mode, timing);
- An associated runnable (only one);
- Time period, data element, event message, operation (only one) depending on the type of the RTE event.

In CHESS, the analogous information discussed here for RTE events is embedded in timing decoration of the provided operations. CONCERTO has extended CHESS to allow decorating private operations as protected or unprotected. Unlike AUTOSAR, CHESS does not allow private operations to be decorated periodic or sporadic. The rationale for this constraint has a methodological nature. By using the CHESS methodology, in fact, the “main” operations of the software functional design must be first defined at interface level, and then realized at

[2] In case of sender-receiver ports, the runnable in charge to receive the data is always triggered by the run-time environment, while in CHESS this holds only when the service called in the target component has been tagged as sporadic.

component level. In this way, the resulting component-based design approach increases the ability to reuse the corresponding functionality. The main operations appear in the provided interface of the component, whereby their visibility has to be public. At instance level, the functional operations can be decorated with timing attributes. For the software architectures of interest to CHESS, the main operations are periodic or sporadic, which makes those two attributes distinct features of main operation. Allowing them to be used for private operations would violate the CHESS methodology principle that associates the component behaviour to the attribute set to its public operations.

3.3.3 Data access

While modelling internal behaviour AUTOSAR allows specifying that a Runnable needs read-access or write-access to the data elements of a required or provided sender/receive port. CHESS does not allow having data flow between components realized through flow ports.

3.3.4 Interrunnable variables

AUTOSAR allows the user to model variables shared by runnables defined in the same component. In particular, AUTOSAR allows specifying how runnables can access this shared state in a manner that makes it thread-safe. CHESS currently does not support this feature, although adaptations to this end have been experimented with in a study funded by the European Space Agency. The solution explored in that side study provided access to the shared data through get/set provided operations, decorated with protected or unprotected access. This would be a viable solution in CHESS, whose implementation is left as a future extension.

3.3.5 Exclusive Areas

In AUTOSAR, runnable entity can be declared to have the ability to enter an exclusive area; therefore the runtime environment has to ensure synchronized access to it. The modelling of exclusive areas are available in CHESS for operation invocation, in particular through protected decoration of operation calls.

3.3.6 Wait points

In AUTOSAR, the runtime environment provides "wait-points" that allow a runnable to block until an event in a set of events occurs; in CHESS these wait points are automatically derived starting from the timing decoration of the provided operations. In CHESS, a given threaded operation can actually wait on a single "WaitPoint" for being scheduled.

3.4 Execution platform

CHESS allows the modelling of the target executing platform by importing the MARTE support for processing resources modelling; only information useful to perform timing analysis is actually of interest in CHESS and thus information about multi-core processors and their interconnections. AUTOSAR allows specifying the ECUs used in the system together with their connection properties; in particular strong support is available in AUTOSAR to model details of the communication between the ECUs, so regarding FlexRay and CAN bus configuration; this support is not available in CHESS.

3.5 Software to hardware allocation

CHESS allows the modeller to define the mapping of PIM component operations decorated as threaded (i.e. cyclic or sporadic) to cores; the allocation of component operations to tasks and then the allocation of tasks to cores is automatically performed in the PSM. Equivalent modelling features, but with different methodological and automatic support, are available in AUTOSAR to map Runnable Entities to tasks and then to map tasks to ECUs.

4 The approach

According to the mapping presented in the previous sections, and so related to the CHESS support for software components modelling, hardware processing resources modelling and software to hardware allocation, we can state that the CHESS methodology and toolset sufficiently cover the initial steps of the AUTOSAR methodology, while adding the possibility to perform early validation of the software applicative level; in particular CHESS supports the Application, System and ECU configuration steps in the AUTOSAR development process.

Schedulability analysis for single and multicore and end-to-end response can be applied, with back propagation of analysis results in the user modelling space, allowing early validation of safety software requirements. Timing analysis could be refined taking into account delays originating from the CAN bus connecting the two ECU’s.

CHESS toolset can be used for validation of timing properties, for instance to calculate the constraints to be applied to threads and tasks based on the end-to-end response time analysis on the modelled SW and HW.

To enforce the applicability of the integration between the AUTOSAR and CHESS workflows, we were able to devise sound model transformations from CHESS to AUTOSAR by using the CHESS and AUTOSAR component models mapping. The corresponding AUTOSAR representation of the information modelled in CHESS can be automatically represented in the AUTOSAR exchange format, i.e., the ARXML (AUTOSAR XML) file is automatically derived by model transformation.

The automatic generation of the AUTOSAR model starting from the CHESS model allows easy integration with external AUTOSAR tools, where the latter can then be used for automatic generation of the AUTOSAR run-time environment to be executed on top of the target AUTOSAR platform, together with the application layer.

The opposite direction from AUTOSAR to CHESS was a problem, however, as the AUTOSAR component model is considerably richer in constructs and in “modelling freedom” than CHESS. The net consequence is that not all AUTOSAR models find correspondence in a legal CHESS model, for syntax and for semantics. This taught us that AUTOSAR pays less attention than CHESS in fostering correctness-by-construction (CbyC) by means of constraints placed on what to model, how to model it, and when in the development flow. One particular exemplar of this difference in intent is worth recalling here. CHESS sets restrictions on the component model (directly in the modelling language and modelling actions availed to the user) to ensure that the chosen forms of feasibility analysis can always be performed soundly on the user model that is decorated with sufficient information attributes. This is necessary to ensure that the model transformation that uses the user model to feed the analysis can be proven correct by construction, i.e., such that the semantic meaning of each analysis artefact and analysis operation corresponds to the semantic meaning of the modelling artefact and decoration attribute in the user model. For instance, in CHESS, a provided operation that is attached to a thread at run time, can only receive release events from a single source (a clock, an external interrupt, another thread, etc.). This restriction causes the run-time semantics of that operation to conform to the abstraction of thread in feasibility analysis. AUTOSAR lifts that restriction, so that the run-time semantics of operations specified in the user model is not guaranteed, by construction, to be statically analysable for feasibility.

The bottom line of the experiment is that a complete (for process coverage and for automation) bi-directional integration between CHESS and AUTOSAR is presently not possible without unsatisfactory compromises. This is a product of the confrontation between the rigidity of seeking adhesion to the CbyC principles (manifest in CHESS), and the permissiveness of wanting to assist without imposing too much perceived burden on the user (manifest in AUTOSAR).

5 Case Study

In the context of the CONCERTO project, the CHESS support for AUTOSAR workflow integration has been evaluated; a cruise control software application has been modelled in CHESS first. Timing analysis has been applied to investigate the benefit of using a multi-core processor with respect to a single-core one. It is worth saying here that CHESS supports an explicit design step that provides the user with guidance on recommended task-to-core allocations, which achieve adequate system utilization; this feature makes it possible to avoid inefficient utilization of the system resources that can arise when performing manual allocation of tasks to cores.

Once the design of the functional component and the evaluation of the timing requirements reached a satisfactory level, the CHESS to AUTOSAR model transformation has been applied.

In the case study the MentorGraphics VSx tool has been used to import the ARXML model produced by CHESS. Because of some limits in the expressivity of CHESS with respect to the richer AUTOSAR concepts, the imported model needed some refinements before being furthermore processed in the AUTOSAR workflow, as for example the specification of inter-runnable variables. Moreover, the models developed within CHESS are more strictly constrained: for example, the usage of flow ports is not allowed among internal software components. The possibility to automatically export tasks definition in the ARXML representation is currently missing and had to be reported by hand in the AUTOSAR model; this is an important step that should be covered in the future in order to be sure that the properties assumed for the timing analysis are preserved while moving to the AUTOSAR process.

6 Related Work

The possibility to allow schedulability analysis in the AUTOSAR context has already been the subject of previous interesting researches.

Anssi et al [10] present a comparison of AUTOSAR and MARTE; in particular they discuss some crucial specification capabilities that need to be satisfied by modelling languages to enable timing analysis in automotive applications. While both languages seem to be expressive enough to enable schedulability analysis, the authors note the fact that more expressive languages require additional effort to define methodological frameworks and tools well suited to allow analysability, which is the focus of our objective.

Anssi *et al.* [11] shows that it is possible to perform scheduling analysis implemented in common open source tools for AUTOSAR systems, but without focusing on the definition of a proper methodology that helps the designer to assure the analysability of the designed system.

7 Conclusions and Future Work

The study performed in CONCERTO has shown that the AUTOSAR and the CHESS component models have significant commonalities, which enables a sound integration of the two respective methodologies. In its intent to pursue correctness by construction, however, the CHESS component model enacts more constraints on the way components can be built and how they can interact. In the same vein, the CHESS methodology disallows some modelling capabilities that are available in AUTOSAR (e.g. the decoration of private operations as periodic of sporadic), presumably in the intent of favouring more liberal reuse. Closer and fuller adherence to AUTOSAR – hence reaching up to methodological considerations – would require:

1. Numerous minor extensions to the attribute and feature set provided for by the CHESS component model, which would be strictly specific to AUTOSAR and not relevant to other domains.
2. The implementation of important "gateway" like instruments to allow the user to transition seamlessly from and to CHESS.

The CHESS response time analysis and back propagation, when fully connected with current commercial AUTOSAR tools, allow elevating the level of abstraction in the design for both single and multicore platforms, keeping trace of timing requirements across all the development cycles with early validation and “correct-by-construction” approach, in contrast to the current and costly practice that is “correct-by-correction”. Non-compliances with respect to these requirements are usually discovered with testing, late simulations or even when deployed in costly prototyping stages. Our feedback from the automotive industry says that the positive impact would result in reducing the related reworking time and cost by an estimated 20% and the development cycles by an estimated 15%.

As our future work, we plan to extend the CHESS component model to achieve stricter compliance with AUTOSAR, e.g. regarding inter-runnable variables support. The CHESS model should be extended to allow modelling of the AUTOSAR basic software entities in order to allow schedulability analysis at a lower abstraction layer. Also ECU’s bus, i.e. their relevant properties, should be modelled in CHESS and so considered during the analysis step to allow better estimation of the end-to-end response time of the application software.

Regarding the integration with the AUTOSAR workflow, CHESS tool support will be extended to allow automatic extraction of task definition from CHESS to AUTOSAR model.

Acknowledgements

The authors are grateful to Tullio Vardanega from the University of Padua for his support for the definition of the CHESS to AUTOSAR component models mapping.

This work is based on the results of the CONCERTO ARTEMIS-JU project.

References

[1] AUTOSAR Partnership, http://www.autosar.org

[2] CHESS ARTEMIS project website, www.chess-project.org/

[3] M. Peraldi-Frati, H. Blom, D. Karlsson, S. Kuntz (2012), *Timing modeling with AUTOSAR: current state and future directions, Proceedings of the Conference on Design*, Automation and Test in Europe, pp. 805-809 .

[4] O. Scheickl, C. Ainhauser, and P. Gliwa (2012), *Tool Support for Seamless System Development based on AUTOSAR Timing Extensions*, Proceedings of Embedded Real-Time Software Congress (ERTS).

[5] Universidad de Cantabria, *Mast: Modeling and Analysis Suite for Real-Time Applications*, http://mast.unican.es/

[6] CHESS Polarsys website, https://www.polarsys.org/chess/

[7] CONCERTO ARTEMIS project website, http://www.concerto-project.org/

[8] MARTE website, http://www.omgmarte.org

[9] SysML website, http://www.omgsysml.org

[10] S. Anssi, S. Gérard, S. Kuntz, and F. Terrier (2011), *AUTOSAR vs. MARTE for Enabling Timing Analysis of Automotive Applications*, I. Ober and I. Ober (Eds.): SDL 2011, LNCS 7083, pp. 262–275.

[11] S. Anssi, S. Tucci-Piergiovanni, S. Kuntz, S. Gérard, François Terrier (2011), *Enabling Scheduling Analysis for AUTOSAR Systems*, 14th IEEE International Symposium on Object/Component/ Service-Oriented Real-Time Distributed Computing

[12] UML website, www.omg.org/spec/UML/

[13] S. Mazzini, S. Puri G. Veran, T. Vardanega, M. Panunzio, C. Santamaria, A. Zovi (2011), *Model-Driven and Component-Based Engineering with the CHESS Methodology*, Proc. of DASIA Conference.

[14] L. Baracchi L., S. Mazzini, S. Puri, T. Vardanega (2016), *Lessons Learned in a Journey toward Correct-by-Construction Model-Based Development*, Proc of 21st International Conference on Reliable Software Technologies- Ada-Europe 2016, Pisa.

[15] *DEEM: DEpendability Modeling and Evaluation of Multiple Phased Systems* [Online], Available: http://rcl.dsi.unifi.it/projects/tools [Accessed: May 14, 2015].

[16] B. Gallina and E. Sefer (2014), *Towards Safety Risk Assessment of Socio-technical Systems via Failure Logic Analysis*, submitted to RISK 2014.

[17] *CHESS Modelling Language*, https://www.polarsys.org/chess/publis/ CHESSMLprofile.pdf.

Model-Based Design and Schedulability Analysis for Avionic Applications on Multicore Platforms

Wenceslas Godard
Airbus Group Innovations, Toulouse; email: Wenceslas.godard@airbus.com

Geoffrey Nelissen
CISTER/INESC TEC, ISEP, Polytechnic Institute of Porto, Portugal; email; grrpn@isep.ipp.pt

Abstract

This paper presents a component-based approach tailored for the modelling of avionic systems. The system is defined as a set of applications developed following time partitioning principle. The toolset provides means to help the designer configure the system and compute a partition schedule for each available processing unit. The model and the generated schedule can then be used as inputs for a response-time analysis engine that calculates the worst-case response time of each task and therefore, assesses the overall system schedulability.

Keywords: Schedulability analysis, Integrated Modular Avionics, IMA, ARINC-653, Component-based, Multicore.

1 Introduction

For aircraft manufacturers, the growing number of software applications to embed on-board along with their increasing size, results in a more and more challenging problem. Their goal is to minimize the weight of the embedded hardware, by maximizing the number of applications sharing a same resource processing, while guaranteeing that no unexpected behaviour will ever occur. The problem complexity partly originates from the hierarchical aspect of Integrated Modular Avionics (IMA) schedulers (see Figure 1) as defined in the ARINC-653 standard [1]. Partitions are scheduled according to statically allocated time windows, and processes are assigned to partitions and scheduled according to a fixed priority scheduling policy inside each partition. This hierarchical structure necessitates the definition of a partition schedule stating when partitions start executing and for how long, as well as setting the relative priorities between the different processes. Operating systems support these parameters, but it is the duty of the system integrator to set them up, in a most efficient way.

In an attempt to reduce costs, aircraft manufacturers have deployed increasing efforts during the last years to integrate the latest processor technologies in their products. The potential benefits are especially evident when considering modern multicore processors. For a same amount of workload, the power consumption as well as the number of hardware components can be reduced in comparison to older technologies. Furthermore, integrating the latest technologies in new aircraft helps limit the costly problem of hardware obsolescence. Indeed, the current regulation imposes to manufacturers to buy and stock large amounts of older and obsolete components to ensure the aircraft maintenance for several decades. Yet, safety and security remain the most important concerns for avionic systems. Therefore, embedding new hardware technologies is acceptable only if correct run-time behaviour can be demonstrated.

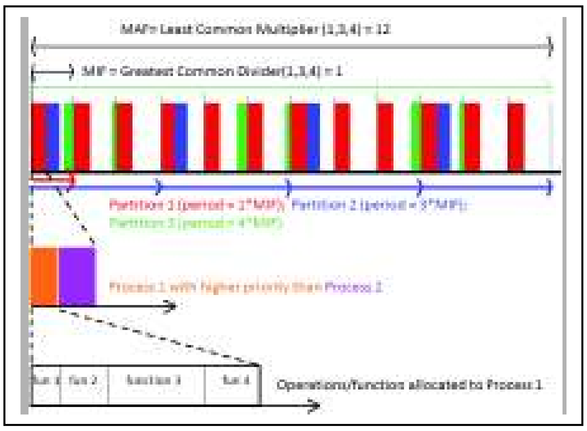

MAF= Least Common Multiplier [1,3,4] = 12
MIF = Greatest Common Divider[1,3,4] = 1
Partition 1 (period = 1*MIF); Partition 2 (period = 3*MIF);
Partition 3 (period = 4*MIF)
Process 1 with higher priority than Process 2
fun 1
fun 2
function 3
fun 4
Operations/function allocated to Process 1

Figure 1: The IMA hierarchical scheduling policy

In that context, this paper presents a framework tailored to facilitate the integration process of several applications on the same computing resource. It is important to note here that the parameters that are generated to configure the system are sound-by-construction, meaning that they are verified at the same time that they are produced, so as to ensure that the timing properties are respected. This framework supports not only the modelling of software application, but also the target processor, including multicore processors. The presented work has been conducted in the frame of an ARTEMIS project named CONCERTO [2][3]. It has been integrated in the CHESS [4] modelling environment and is being released under the PolarSys initiative [5].

2 State of Practice

Avionics producers traditionally follow the Integrated Modular Avionics (IMA) approach to improve the reliability of the system and ease the development and integration of several applications on a same execution platform. In IMA systems, multiple applications can share a same processing resource (e.g., a single-core processor). Applications are assigned to partitions and partitions are allocated to specific time slots during which they can execute at run-time (see Figure 1). This time slot allocation, also referred to as the *partition scheduling table*, is performed offline by the system integrator. This segregation between applications is very important because it makes possible to perform verification of each application separately. One can then prove the correctness of the overall system by simply ensuring the robustness of the partitioning mechanisms. In order to determine a schedule that makes an efficient usage of the processing resource, many system properties and constraints have to be taken into account. Applications are composed of processes, and processes comprise operations. These operations, also referred to as functions, are defined by their own period which is either equal to the period of their belonging process, or a multiple of it. Each operation has an execution time, which can be estimated by experimentations or formally computed by a static analysis tool such as AiT [6], OTAWA [7], or RapiTime [8]. Operations may also be involved in precedence relationships that designate groups of operations to be executed sequentially. As for the processes, they come with their priority, period or minimum inter-arrival time, and deadline.

Efficiently using this information, the integrator may assign offsets to functions in order to distribute the resource usage evenly over time. The integrator must also generate the partition scheduling table so as to ensure that all processes will always complete their execution by their deadline. Currently these operations are done by hand and rely mainly on the experience of the system integrator.

The approach presented in this paper uses an abstract model of the overall system in order to semi-automate the generation of the data required (1) to configure the different functions and (2) to produce the partition scheduling table. It enables an early validation and performance estimation for new projects. It is also suitable whenever an existing project, has to be modified, updated or redeployed. Indeed, only information from the modified element has to be spread into the CHESS model. A new system configuration can then be automatically generated. In the particular case of a re-deployment, a very useful feature of CHESS is the support of multicore processor targets. Details about the advantages and limitations related to the multicore support are given in section 4.

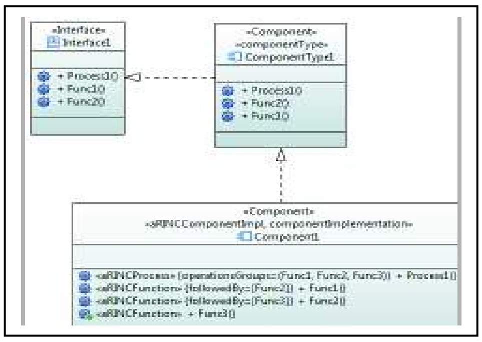

«Interface»
Interface1
+ Process1()
+ Func1()
+ Func2()
«Component»
«componentType»
ComponentType1
+ Process1()
+ Func2()
+ Func3()
«Component»
«aRINCComponentImpl, componentImplementation»
Component1
«aRINCProcess» {operationsGroups=[Func1, Func2, Func3]} + Process1()
«aRINCFunction» {followedBy=[Func2]} + Func1()
«aRINCFunction» {followedBy=[Func3]} + Func2()
«aRINCFunction» + Func3()

Figure 2: Class diagram in the component view

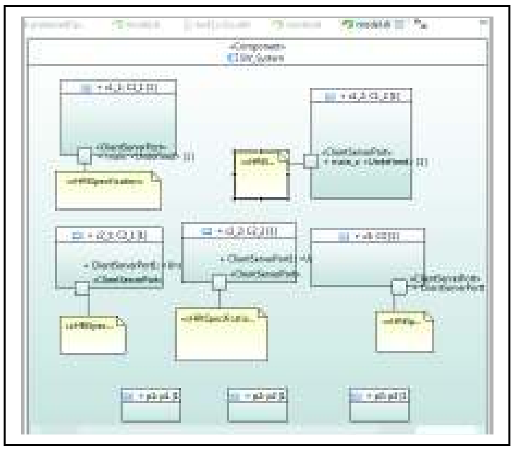


Figure 3: Composite Structure diagram

3 Toolset Description

The toolset discussed in this paper has been integrated in the CHESS environment [4]. CHESS is a component-based design methodology and language articulated around multiple views treating of different aspects of the system design (e.g., functional against non-functional properties, and application versus execution environment) [9]. CHESS proposes domain specific features tailored for a given application domain. In its latest version, CHESS was extended to support the design of avionic systems. New meta-model entities modelling different elements of a typical IMA architecture as well as semi-automated configuration and analysis tools are now available.

Starting from higher level requirements such as periodicity, deadline and priority, and information about ARINC-653 processes and their implementation, the proposed toolset can be used to deploy, configure and analyse the schedulability (i.e., the capability for all functions composing the system to always respect their deadlines) of the overall system in a mostly automatic way.

The CHESS methodology for avionic systems consists of the following four successive steps:

- Modelling the ARINC-653 processes, the logical partitions and the execution platform;
- Partitioning (i.e., assigning processes to partitions and partitions to processor cores);
- Configuring the system (i.e., priority and offset assignment, generation of the partition scheduling table);
- Analysing the worst-case response time of each function and the overall system schedulability.

3.1 System Model

The CHESS platform is composed of different views that are used to model the software (using the "Component View"), model the hardware (using the "Deployment View") and perform the analysis (using the "RealTimeAnalysis View"). CHESS imports the UML support for component-based design with some limitations; in particular, limitations are applied to avoid the UML-way of modelling real-time information. The latter being better supported by the UML profile for *Modelling and Analysis of Real-Time Embedded Systems* (MARTE) [10], adopted and integrated in CHESS.

In the "Component View", CHESS uses Class diagrams to model the processes and their allocated functions (see Figure 2), and Composite Structure diagrams to model the functional interactions between instances of those components (see Figure 3). Information on the run-time behaviour and timing properties of each process and function can also be associated with the components using an "Instance View". As for the "Deployment View", it provides support for the modelling of the information related to the deployment of software components on the hardware platform. Using this view, the user is able to represent the hardware entities and their relevant properties, in particular by using the MARTE specific stereotypes defined in the *Hardware Resource Modelling* (HRM) sub-profile. Finally, the "RealTimeAnalysis View" is used to store the entities derived by model transformations which are needed to perform the system schedulability analysis.

3.1.1 ARINC-653 processes modelling

Each CHESS component implementation modelling an ARINC-653 component is associated with a specific stereotype called ARINCComponentImpl (see Figure 2). It allows modelling ARINC-653 processes and their functional and extra-functional properties. Each ARINCComponentImpl is assigned one public operation modelling an ARINCProcess, and a collection of private ARINCFunctions modelling the different functions composing the process associated with that component (see Figure 2).

Using the "Instance View", extra-functional properties (i.e., priority, deadline, release offset and release period or minimum inter-arrival time) can be added to the process specification (see Figure 4).

The WCET of each function can be specified in a similar manner.

Finally, so as to accurately model the run-time behaviour of the processes, the system designer can specify "rate dividers", precedence and exclusion constraints over functions of a same process. Rate dividers allow the system designer to model the fact that not all functions must be executed at the same period than their corresponding process. Assigning a rate divider of value r (where r is a natural number larger than 0) to a function f associated to a process p, means that f executes once every r activations of p.

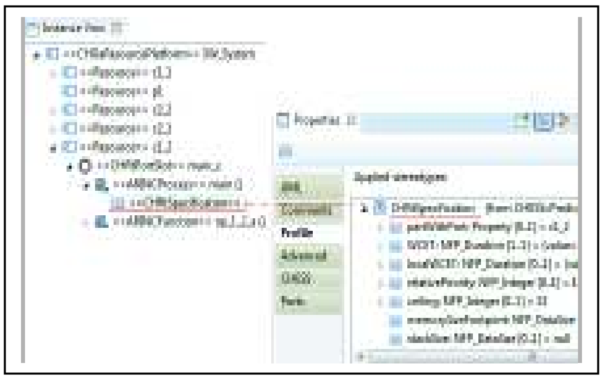


Figure 4: Instance view

Precedence constraints between functions of a same process allow to define an order of execution between those functions. Precedence constraints are specified using the "FollowedBy" attribute of each function (see Figure 2).

Exclusion constraints are used to define groups of operations that can or cannot compete for processing resources. If G_1 and G_2 are two groups comprised of different functions, an exclusion constraints between G_1 and G_2 means that all the functions in G_1 must have completed their execution when those of G_2 start running, and inversely, all the functions of G_2 must have completed before those of G_1 start executing. Such grouping mechanism may be used, for example, to better distribute the execution load over time, or to avoid functions competing between themselves (otherwise using a locking protocol) when they access the same resources. In CHESS, function groups and their exclusion constraints are specified using the "OperationGroups" attribute of the appropriate process (see Figure 2).

3.1.2 Partition modelling

IMA partitions are modelled as "functional partitions" in CHESS. Instances of ARINC-653 components can be manually assigned to those partitions. Processes linked to a partition P are isolated in the time and space domain from all processes that are not assigned to P, meaning that a timing misbehaviour or a data corruption caused by a process that is not in P cannot propagate to those in partition P.

Partitioning is of key importance for the IMA methodology as it allows for the independent development and verification of applications. Furthermore, such a design approach also permits to integrate applications of different criticalities on a same execution platform.

3.1.3 Execution platform modelling

The processing platform is modelled in the "Deployment View". CHESS supports multicore processors where each core is identical in capabilities and performances to the other cores.

Functional partitions can be manually (using a graphical interface as shown on Figure 5 or automatically (using the tools discussed in Section 3.2) assigned to those cores. Note that a process is assigned to at most one partition, and a partition to at most one core. A partition P can execute

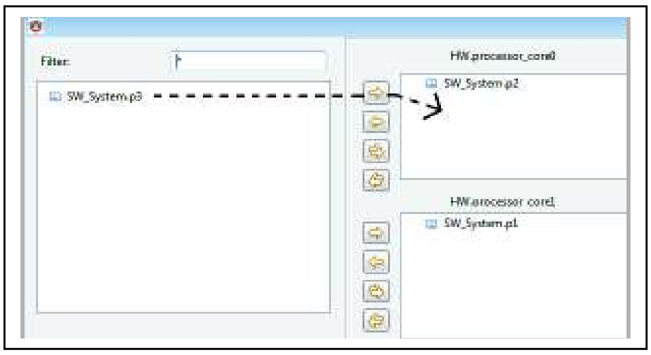


Figure 5: Partition to core assignment

only on the core to which it is assigned, that is, the processes allocated to *P* cannot migrate or execute in parallel on different processor cores.

3.2 Configuration Tools

During the CONCERTO project, CHESS has been enhanced with tools automatically or semi-automatically configuring IMA compliant avionic systems. After the modelling of the system architecture, properties and run-time behaviour using the modelling features described in the previous section, CHESS automatically dimensions the partitions, proposes a partition-to-core mapping, and computes a partition scheduling table for each core. It also assigns relative priorities and release offsets to functions so as to enforce the precedence and exclusion constraints specified in the model, and attempt to better distribute the execution resources accesses over time. The automatic system configuration process is made of four successive steps:

1. First, the precedence and exclusion constraints together with the timing properties of each function are extracted from the model. Based on that information, a priority, an activation period, release offset and deadline is computed for each function, so as to respect the model specifications and distribute the execution load over time.

2. Then, the period of activation and execution budget of each partition is calculated so as to ensure that all the functions assigned to that partition will respect their timing constraints.

3. Once properly dimensioned, the partitions are automatically assigned to processor cores. Two rules are used to guide the partition-to-core mapping phase. First, any mapping preference specified in the model by the tool user (see Section 3.1.3) is respected. Second, the execution load is balanced as much as possible between the available cores.

4. Finally, the partition schedule is generated. The Minor Frame (MIF), Major Frame (MAF), and execution windows of each partition are computed for each processor core. The generated partition schedule ensures that all the functions and processes will respect their timing constraints.

Every solution proposed by the toolset is back-propagated to the model and can be adapted by the user. Those highly critical and time consuming development tasks are therefore drastically simplified and accelerated for the system integrator.

3.3 Analysis Tools

Whenever a partition schedule has been computed for all cores, a model transformation is triggered so as to perform a response time analysis. The analysis is performed using an extension of the MAST analysis tool [11] developed during the CONCERTO project. This extension is able to compute the exact worst-case response time of each function executed in the generated IMA partition schedule and thus provides useful performance indicators to the system integrator with respect to the system implementation and configuration.

The timing information provided by the timing analysis tool are back-propagated to the CHESS modelling environment and can be used during an early design phase to better dimension the execution platform or the amount of execution resources associated to each process. It can also serve as indicators to help the system integrator refine its system deployment decisions in a later phase of the project.

4 Support for Multicore

The emergence on the market of multicore processors is expected to come along with many advantages for the avionics industry, since embedding more applications on a single chip means less weight, less power consumption and fewer parts to stock for maintenance purposes. IMA was defined at a time where multicore processors were not foreseen as a viable solution for avionic systems. There is therefore a need to propose new software development processes adapted to safety critical applications targeting a multicore execution environment. Supporting multicore deployment was consequently expressed as a requirement of the CONCERTO project. Although results of the project are not numerous enough to precisely assess the performance gains that can be brought by the latest processors, our work contributes to solving challenges that are presented in deeper details in the rest of this section.

First, with respect to the deployment strategy, it is important to note that the choice to only support a static one-to-one partition-to-core mapping has been made. In other words, a partition is allocated to one and only one core by the schedule generation step, and no migration is allowed. Of course the allocation can be achieved manually by the user; but if no such information is provided then the toolset automatically assigns a feasible allocation of partitions to cores. Therefore, at the end of the system configuration process, each core gets a set of partitions allocated to it, and a partition scheduling table. For this purpose, a new engine for the schedule generation had to be designed, with an extension to the schedulability tool plugged in the toolset. This extension of schedulability analysis tool able to return response times for all functions of a two-level scheduler system deployed on a multicore target has been a major outcome of our work. However, some limitations do exist in the current status of the presented approach, especially in order to take into account extra delays that are generated by the interferences introduced by multicore processors. Such interferences are introduced by hardware resources such as memories, buses and caches, shared by tasks running on different cores [12].

As a reminder, in CHESS, the worst case execution times are assumed to be provided as inputs to the model and are used in order to compute the partition schedule. Whereas in the single-core case the WCET can be fairly accurately computed with static analysis tools, in the multicore case, several functions can run at the same time on different cores leading to delays that static analysis tools cannot currently soundly estimate without excessive pessimism. Consequently, the timing bounds cannot be guaranteed, which makes the overall process inefficient. Fortunately, some recently published research results are compatible with CHESS' goal and their results could later be integrated in the toolset. Among those solutions, the one presented in [13] offers the basis for a theoretical framework for building a solution. Besides, a description of an implementation of it, completed with experimental results, can be found in [14].

5 Conclusion

In this paper, we have presented a toolset dedicated to the development of avionic systems. A meta-model has been defined for the modelling of avionic applications and IMA partitions. The proposed toolset helps the integration process of several applications on the same computing platform by semi-automatically configuring the system and generating a partition schedule. The toolset allows precedence and exclusion constraints to be annotated to the application model. Those constraints are then considered by the toolset in order to automatically compute a system configuration enforcing the constraints at runtime. Another piece of automatically generated information is the table that defines the time windows regulating the execution of each partition. In the case of multicore, the configuration engine also generates a mapping of partition to cores. In this case, each core has a completely independent table for the time windows. Further extensions related to multicore are already being considered and have been identified so as to make the tool able to deliver results that can be considered as formal proofs and thus be quoted as such for the certification process. The release of the platform under Polarsys will facilitate the toolset maintenance and updates with new features.

References

[1] ARINC (2005), *Avionics Application Software Standard Interface PART 1 – Required Services, ARINC Specification 653-2.*

[2] Mazzini, S. (2015), *The CONCERTO Project: an Open Source Methodology for Designing, Deploying, and Operating Reliable and Safe CPS Systems,* Ada User Journal, vol 36 no 4, pp 264-267.

[3] CONCERTO project, online: *http://www.concerto-project.org*

[4] CHESS, online: https://www.eclipse.org/proposals/polarsys.chess/

[5] PolarSys, Open Source Solutions for Embedded Systems, online: https://www.polarsys.org/

[6] AbsInt, *aiT WCET Analyzers,* online: https://www.absint.com/ait/index.htm

[7] OTAWA, online: *http://www.otawa.fr/*

[8] RAPITA Systems ltd, *RapitTime*, online: *https://www.rapitasystems.com/products/rapitime*

[9] Cicchetti, A., Ciccozzi, F., Mazzini, S., Puri, S., Panunzio, M., Zovi, A., Vardanega, T. (2012), *CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems*, In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 362-365, IEEE.

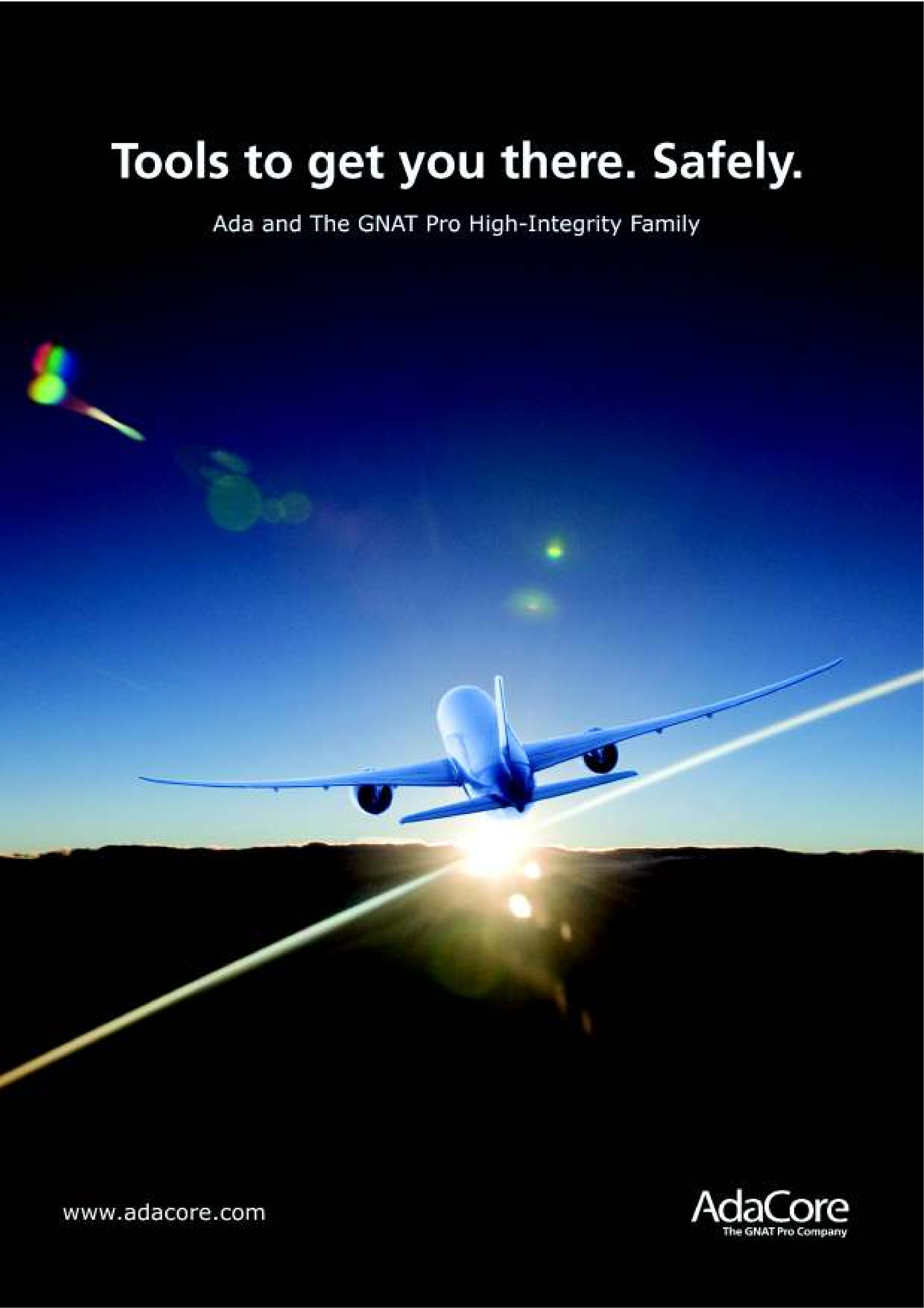
[10] Object Management Group (2011), *MARTE A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, v.1.1,* online: http://www.omg.org/spec/MARTE/1.1

[11] Universidad de Cantabria, *MAST: Modeling and Analysis Suite for Real-Time Applications*, online: http://mast.unican.es/

[12] Dasari, D., Akesson, B., Nelis, V., Awan, M. A., Petters, S. M. (2013), *Identifying the sources of unpredictability in COTS-based multicore systems,* In 8th IEEE International Symposium on Industrial Embedded Systems (SIES), pp. 39-48, IEEE.

[13] Kim, H., de Niz, D., Andersson, B., Klein, M., Mutlu, O., Rajkumar, R. (2014), *Bounding memory interference delay in COTS-based multi-core systems*, In IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 145-154, IEEE.

[14] M'sirdi, S., Godard, W., Pantel, M., Stilkerich, S., (2016), *Improved Resource-Efficient Allocation of IMA Applications to Multi-Cores,* In IEEE/AIAA 35th Digital Avionics Systems Conference (DASC).


Tools to get you there. Safely.
Ada and The GNAT Pro High-Integrity Family
www.adacore.com
AdaCore
The GNAT Pro Company

Overview of the 18th International Real-Time Ada Workshop

11-13 April 2016
Benicàssim, Spain

Contents *

Program Committee

Mario Aldea Rivas, John Barnes, Ben Brosgol, Alan Burns, Michael González Harbour, José Javier Gutiérrez, Stephen Michell (Program Chair), Brad Moore, Luís Miguel Pinho, Juan Antonio de la Puente, Jorge Real, José F. Ruiz, Sergio Sáez, Joyce Tokar, Tullio Vardanega, Andy Wellings and Rod White.

Workshop Participants

Mario Aldea Rivas, University of Cantabria, Spain
Alan Burns, University of York, UK
Alfons Crespo, Universitat Politècnica de València, Spain
Jorge Garrido, Technical University of Madrid, Spain
Michael González Harbour, University of Cantabria, Spain
Kristoffer Nyborg Gregertsen, SINTEF ICT Trondheim, Norway
Stephen Michell (Program Chair), Maurya Software, Canada
Luis Miguel Pinho, Polytechnic Institute of Porto, Portugal
Juan Antonio de la Puente, Technical University of Madrid, Spain
Jorge Real (Workshop Chair), Universitat Politècnica de València, Spain
Pat Rogers, AdaCore, USA
Sergio Sáez, Universitat Politècnica de València, Spain
Tullio Vardanega, University of Padua, Italy
Andy Wellings, University of York, UK
Juan Zamorano, Technical University of Madrid, Spain

Sponsors

Ada europe AdaCore The GNAT Pro Company DISCA DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES UNIVERSITAT POLITÈCNICA DE VALÈNCIA

* The Proceedings of the 18th International Real-Time Ada Workshop are published in the June 2016 issue of ACM Ada Letters.

Session Summary: Parallel and Multicore Systems

Chair: Luis Miguel Pinho

Rapporteur: Stephen Michell

Abstract

The first session of the 18th International Real Time Ada Workshop discussed two aspects of parallel programming in real-time systems, the use of executors in parallel systems, and syntax to guide the reduction of parallel computations to return a correct single answer. This report captures the discussions held and the decisions and recommendations of the workshop on these topics.

1 Introduction

The Multicore/Parallel Processing session examined issues associated with the addition of syntax to Ada to effectively manage parallel computation on multicore processors. The papers considered were:

- Michell, Pinho, Moore, Taft, "Constraints on the Use of Executors in Real-time Systems" [4].
- Taft, Moore, Pinho, Michell, "Reduction of parallel computation in the parallel model for Ada"[5].

2 Discussion

2.1 Executors

The use of executors was introduced at IRTAW 2015 in [1] to map the execution of tasklets in a directed acyclic graph (DAG) to the underlying processors of a multicore processor. Many issues were discussed in IRTAW 2015, but some of the fundamental questions were not concluded, and some other were not discussed. This session addressed those issues as follows.

Changing Task Base Priorities or other attributes

Changing a task priority while a task is executing a parallel opportunity could result in

a) multiple tasklets executing the same priority changing (but possibly with different values of base priority); or

b) some tasklets having priority boosted and others not boosted.

It was generally agreed that the results of changing a tasks base priority inside a POP may be ineffective and even problematic, but cannot in general be statically flagged as an error.

It was agreed that although setting a task's priority from within a POP should be disallowed, it could be eventually difficult to check. There was a strong sentiment that such priority setting should be at worst a bounded error, not erroneous. There was also a discussion if it should be a different behaviour when the call to change priority was executed by the tasks code (when executing in a parallel region) or of it was a call from a different task (which was performed while the affected task was in a parallel region).

It was noted that Ada already has rules for the immediacy of Set_Priority for the same task and for other tasks on single-CPU and multiple CPU hardware, as well as the definition of abort-deferred regions. The Ada language already provides a notion of base_priority and active_priority. Changing the Pase_Priority of a task only changes its active priority when the next synchronization point is reached for the task whose priority is being changed.

Due to these issues, the general agreement was that the effects of the call should be deferred. It was proposed and accepted that POP's should be abort-deferred regions, and that the current rules of Set_Priority should apply as currently written.

For setting the priority of a task, if multiple tasklets make the same call, it is not specified which one will succeed.

The discussion also considered if multiple different actions are pending during a POP, what happens when returning from the parallel region. For instance, an exception is raised by code executing in a tasklet whilst another tasklet changes the priority to the task.

The general consensus was that the different actions should be applied, and the same behaviour as per similar situations in the tasking model. In the example, the task priority would be changed before the exception handler being executed.

Timing Events

The next discussion was on the use of timing events from within a POP. In the general model, tasklets can self-suspend in order to communicate with other tasklets, but in the RT model, tasklets should not self- suspend.

To show the requirement for tasklet suspension, Miguel showed the following example:

```
for I in parallel 1..N loop
        Do_something; Barrier;
        Do_Something_Else;
end loop;
```

This could be broken into two POP's, but a significant part of the performance improvement from parallelization derives from the localization of data, and having the same tasklet and executor retain the data that it processed in Do_Something to further process it in Do_Something_Else.

The blocking in question could be a variant of synchronous barriers, or could be entries. Since Ada already supports asynchronous transfer of control, the mechanisms to permit the queuing of code fragments is already in Ada, and can be leveraged for tasklets.

The workshop discussed delaying one or more tasklets on a timer, and decided that calls to delay, delay until or calls to suspend on execution timers should not be permitted within a POP. Since tasklets can call subprograms, in general it is not decidable if a such a call is within a POP, so it must be a bounded error.

It was noted that some subprograms may not be safe to call from within a POP (such as ones that have an internal suspension). There was a strong desire to be able to mark the specification of subprograms as non-blocking, and to have an aspect or a pragma such as H.5's pragma Detect_Blocking. Such a contract would be transitive, in that a subprogram that was marked as non-blocking could only call subprograms that were non-blocking.

It was decided that it would be useful to create parallel regions (likely the region that is marked by the *parallel* keyword). Parallel regions could then be used to forbid certain operations, such as setting event handlers.

A side discussion was held about the expression of concurrency within a program. Some members had a strong sentiment for a construct such as :

```
parallel over I in 1..N
        Do_1
barrier
        Do_2
end_parallel;
```

Where parallel over would be the parallel loop construct. Others pointed out that this would require significant code rewrite to change sequential loops into parallel loops, and would prohibit opportunistic parallel loop creation.

No decision was made on syntax.

Execution Time Timers

The workshop discussed how execution time timers can be used in an environment with tasklets and with executors. Execution time timers are applied at the task level, and execution time timers are used to notify a task when its execution time for the current work package has been exceeded. There may be some benefit in subdividing a tasks execution time budget into smaller budgets to manage execution time at a finer granularity, but there are significant issues with such a subdivision:

c) The executors that execute tasklet code are very simple structures, and tracking their execution time individually increases tasklet complexity and removes some of the benefit of parallelization.

d) Modern processors are complex devices, with "lanes" and pseudo cores. It is not clear that applying execution time concepts to executors processing code in a "lane" or in a hyper-threaded core is meaningful.

e) Tasklets are not individually identifiable as code chunks, hence calls to set an execution timer or waiting on an execution time timer are impractical.

The accounting of all time used by various tasklets and charged to a task is straightforward, but when such time is accounted can be an issue. The workshop accepted that updates to the execution time of a task can happen at the end of a POP, thus actually deferring potential overrun actions. It would also be desirable to account for execution time of each CPU or for each executor, and provide facilities to handle per-CPU or per-executor overruns (to isolate a misbehaving tasklet), but it was considered that a simpler model should first be provided, and then augmented.

Parallelizing inside interrupt/timing event handlers

Disallowing parallelization inside interrupt/timing event handlers had been decided at IRTAW 17 In this workshop it was discussed if interrupt handlers could/should also use a pool of executors in a similar way as tasklets. It was agreed that this would not be forbidden, but would be implementation dependent.

Relation with Set_CPU/Get_CPU

Set_CPU

The Ada function Set_CPU locks a task to the CPU identified as a parameter to the call. The question for the workshop is to determine what effect Set_CPU has on tasklets in a POP. Clearly it makes no sense to have a model where a task calls Set_CPU and then enters a POP, but all of the tasklets are restricted to a single CPU, thereby defeating parallelism.

The workshop discussed the possibility of permitting the tasklets to be executed on CPU's that were within the dispatching domain that contains the task's CPU, or to create a new Set_CPU to specify a set of CPUs within the dispatching domains of the task, where the tasklets would be able to execute.

The agreement was that a new Set_CPU should be provided, to specify the set of CPUs where tasklets of the task could execute. Calling this new function with a single CPU would then force all tasklets to be executed in this CPU (thus defeating parallelism).

The current Set_CPU, if used for a task with tasklets, would pin only the "master" tasklets, i.e., the one which is executed when the task starts. Other tasklets are free to execute in the dispatching domain that holds the parent task. To achieve the behaviour that all tasklets run on the same CPU, allocate the task to a dispatching domain that contains only the single CPU.

Get_CPU

In addition to the current subprogram Get_CPU to return the CPU that is executing the named task, there is a need to determine the set of CPU's that are executing tasklets for that task. Therefore, a new call for Get_CPU is also required with expanded semantics.

Tasklet stealing

In the real-time model for parallel Ada, tasklets run-to-completion in the same executor where they started execution and parent stealing is disallowed, mainly because analysis and predictability are extremely difficult to impossible. It was noted, however, that for some algorithms, parent stealing is significantly more efficient than blocking the parent until all tasklets complete.

It was decided that work stealing or parent stealing should not be forbidden, although it should be made clear that currently there is no analysis for hard real-time systems.

Distinguishing between number of allowed and active executors

For efficiencies sake, the number of executors in a system will almost always be more than the number of cores by at least a factor of two. There are also at least two different notations for the number of executors – the number of global executors available to an application, or the number of active executors (i.e. running or ready-to-run. These two numbers can differ, because executors can be allocated on a per-task basis or globally. Depending upon system configuration, either notion (global executors or active executors) may be needed.

It was initially considered that the specification of the number of active executors would be sufficient. Nevertheless a second view of the problem identified a situation where having both could be interesting. If a task wants to restrict the level of parallelism but still have "spare" executors in case some block, it might need to provide different numbers to the available and allowed simultaneous executors. Therefore the final decision was that an interface to specify both should be provided.

Explicit control of executors

The workshop discussed the feasibility and desirability of dynamically controlling the number and/or behaviour of executors executing the tasklets in a POP. The workshop agreed that, aside from allocating the number of executors to the overall application or task (initial configuration), that explicit control of executors is not desired.

Tasklet minimum execution time

The workshop discussed the cost of parallelism, in that the creation of tasklets, the mapping of the algorithm onto tasklets, reduction and reducing the partial results into a final value set usually costs one or more orders of magnitude processing than executing a single iteration or branch of the algorithm. Hence, work is aggregated into contiguous appropriately-sized *chunks* which are executed in parallel with other chunks to maximize parallelism and minimize work distribution overhead. For general systems also, but in particular for real-time systems, the application developer must be able to control the splitting of data into chunks.

The workshop discussed mechanisms to configure this *chunking*. A compiler option is a possibility, but there was a preference for a language-defined aspect to control chunking, either based upon the type or based on the object. It was agreed that confirming pragmas or aspects would be suitable in most situations.

It was discussed that it is currently impossible to use aspects on loop bodies because of language restrictions, but that this should be extended.

It was decided that, for real-time systems, that ability to control tasklet configurations is a requirement. It was also discussed if the programmer should be given control of the minimum size of tasklets (e.g. instructions). It was considered that this was better left to the compiler.

Relation with simpler runtimes

The subject of a simpler runtime for parallelism was discussed in a separate session (about profiles). There was general consensus that work should exist leading to the proposal of a profile for efficient parallelism.

Other pre-emption models

An interesting model for highly parallel execution is where pre-emptions are deferred to tasklet boundaries (when an executor completes the tasklet it is executing when a pre-emption is requested). This can improve data locality. However, Ada current policies are either pre-emptive or non-pre-emptive.

The workshop discussed the use of a non-pre-emptive policy, together with careful use of Yeld as a potential solution, but it would force non-pre-emptive even when no parallel tasklets are being executed, or potentially introduce large blocking in particular tasklets with large execution time. There is also the issue that the main program is an implicit tasklet, making the concept that tasklets are non-preemptive too strict.

There was no decision taken on this subject.

Applicability to high-reliability hard real-time systems

It was discussed that there is currently insufficient analysis to make parallel systems such as was proposed here and in [1], [2] and [3] suitable for high reliability (critical) hard real-time systems. This is mainly due to the timing interference of highly parallel code, and its reflection in current real-time analysis. At the present time, however, systems that require such parallel processing should be decoupled from the high- reliability system.

2.2 Parallelism Syntax

The session also discussed a second paper, about the reduction of parallel computations in the proposed Ada support for parallelism [5]. Miguel presented the situation and the issues for discussion. Reduction of partial results from multiple tasklets is relatively straightforward if the reduction is done in the same order as the serial computation would have occurred. But as tasklets complete in arbitrary order this might not be the case. Waiting for all tasklets to complete before the reduction can take place may waste time and resources, therefore it is desirable to support reductions in an arbitrary order, and even in parallel. The addition of non-associative or noncommutative reduction operations further complicates how one specifies reductions.

The workshop discussed alternatives for performing reductions, as presented in the paper. The main approaches discussed were the type-based approach and the reduction based on hyper-objects.

This latter approach to reduction uses a set of generic Ada packages, together with the proposed parallel loop syntax. The programmer creates reduction objects, by instantiating the package with the reduction type and operations. However, the user is required also to program explicitly the processing inside the parallel loop.

Pat noted that instead of a generic approach, a type-extension approach could be explored.

In the type-based approach, a special reduction variable is created to be used in the loop. This is an array, which allows each parallel "chunk" to process an independent value, which are then reduced at the end. The special box (<>) notation is used to denote this special behaviour:

```
declare
  type Array_Type is new array(…) of float:
  Arr : Array_Type;
  type Partial_Array_Type is new array (parallel <>)
       of Float with Reducer => "+", Identity => 0.0;

   Partial_Sum : Partial_Array_Type := (others => 0.0);
   Sum : Float := 0.0;

begin
  for I in parallel Arr'Range loop
    Partial_Sum(<>) := Partial_Sum(<>) + Arr(I);
  end loop;

  Sum := Partial_Sum(<>)'Reduced;
  -- value is reduced either here or
  -- already during the parallel loop

  Put_Line("Sum over Arr = " &   Float'Image (Sum));
end;
```

In this example, the array used for reduction (Partial_Sum) is of a type that has reducer functions defined for the array and the operation. The implementation is free to call the reduction operation as tasklets complete their *chunk* of the loop, or to reduce values when the Reduced aspect is used.

There was significant preference at the workshop for syntax-based approaches to parallelism, including reduction. The workshop considered that the proposed approach binds too late the array being processed and the reduction array, and that this binding is weak. It would be necessary to connect both, in principle as soon as the reduction variable is defined. Such as:

```
type Array_Type is new array(…) of float:
Arr : Array_Type;
type Partial_Array_Type is new array (parallel <>) of
Float
with Reducer => "+", Identity => 0.0;

Partial_Sum : Partial_Array_Type := (others => 0.0);

  for Arr'Reducer use Partial_Sum;
```

Or

```
  Partial_Sum : Partial_Array_Type
  with Array => Arr;
```

There was a discussion on why the reduction variable is an array. It could be a single variable (potentially of a controlled type). However, there was a note that it could eventually be interesting to be able to access specific partial values (e.g. accessing the left and right chunks' partial values).

The group also briefly reviewed the ideas for parallel iterators for containers, which were considered to fit well (and to complement) the proposed type-based approach.

There was a general discussion about whether parallelism support should be placed in the core language or placed in specialised annexes. It was noted that parallelism are part of the control flow of the program, and parallels were made to other Ada features, such as tasking and priority, where the model and basic capability is defined in the core of the language and extra syntax, calls and restrictions are placed in specialized annexes for real-time, distribution, and high integrity systems. It was agreed that a similar approach for parallel syntax would be desirable, with the basic syntax placed in the core language and pragmas, aspects, libraries and restrictions placed in the real-time annex for real-time systems.

It was noted, that for real-time systems where the application developer needs complete control over the parallelism and the reduction operations, the programmer may need to explicitly take control of chunking and reduction, which should be allowed by the language. A pragma No_Implicit_Parallelism could be also necessary.

3 Conclusions

The following summarizes the main positions taken by the workshop during this session:

- Changing attributes of tasks should be deferred until being out of any parallel execution. If parallel tasklets perform multiple attribute changes, arbitrarily one is selected;
- If multiple different operations are deferred during the parallel execution (such as an attribute change and an exception) all should be applied in a manner similar to the current model for sequential Ada;
- The notion of parallel regions should be considered, both for the previous behaviours, and also used to forbid certain operations, such as setting handlers;
- Although a more accurate per CPU time accounting of a task parallel execution is desirable, a simpler model is initially proposed, which only provides a per task single values, updated at the end of parallel regions;
- Set_CPU and Get_CPU calls should be also provided to specify the group of CPUs where tasklets of a task may execute;

- Ada is a syntax-based language, and parallelism should also follow the model. Parallelism introduces a significant model extension to Ada which should be specified in the core of the standard;
- For real-time systems, the programmer should be able to take control of the parallelism decomposing and operations in the program.

References

[1] Michell, S., Moore, B., Pinho, L.M (2013), *Tasklettes – a Fine-Grained Parallelism for Ada on Multicores*, Ada Europe 2013, Springer Publishing.

[2] Taft S.T., Moore B.J., Pinho L.M, Michell S.G. (2014), *Safe Parallel Programming in Ada with Language Extensions*, HILT/Sig Ada Conference 2014, ACM New York.

[3] Pinho L.M., Michell S.G., Moore B.J., Taft, S.T. (2015), *Real-Time Fine-Grained Parallelism in Ada*, Proceedings of the 17th International Real Time Ada Workshop, Ada Letters 2015, ACM New York.

[4] Michell S.G., Pinho L.M., Moore B., Taft T. (2016), *Constraints on the Use of Executors in Real-time Systems*, Proceedings of the 18th International Real Time Ada Workshop, Ada Letters 2016, ACM New York.

[5] Taft S.T., Moore B.J., Pinho L.M., Michell S.G. (2016), *Reduction of Parallel Computation in the Parallel Model for Ada*, Proceedings of the 18th International Real Time Ada Workshop, Ada Letters 2016, ACM New York.

Session Summary: Deadline Floor Protocol

Chair: Alan Burns
Rapporteur: Andy Wellings

1 Introduction

At the 2013 IRTAW Workshop [1], it was accepted that the Deadline Floor Protocol (DFP) has many advantages over the Stack Resource Protocol (SRP), and that it should be incorporated into a future version of the language, and that ideally the support for SRP should be deprecated.

The overall goal of this session was to determine if the DFP was now mature enough to be put forward as an AI.

2 Protocol Discussion

The workshop focused on two issues associated with the behaviour of the protocol, which had been previously identified by Michael González Harbour. The first issue was that setting the deadline floor of a protected object to be the minimum relative deadline of the tasks that access it did not take into account release jitter. Release jitter is a property of the platform and hence a program's portability is undermined. However, this is also the situation with the current SRP protocol. Unfortunately, if the release jitter of a task is underestimated then under DFP (on a single processor) mutual exclusion cannot be assured and hence a lock is required. This is not the case with the SRP. Given that a lock is required for both protocols in a multiprocessor environment, the workshop agreed that the DFP should require the lock. Of course, deadlocks can occur if strict order of nested locks is not enforced.

The second issue discussed was that of a task (on a single processor) self suspending within its release and the task overrun its assumed worst case execution time. The following code was considered to show that without a lock the DFP protocol would be compromised. The example consists of two tasks: T1 and T2 that share a critical section and T1 self suspends.

```
task body T1 is
begin
  Next_Time:=Clock
  Set_Deadline(Next_Time+5ms)
  loop
    Action1_almost_5ms
    Trigger D/A conversion
    Wake_Up_Time:=Clock+5ms
    delay_until_and_set_deadline(
       wake_up_time, Next_Time+20ms)
    Action2_5ms
    Critical_Section_1ms
    Action3_1ms
    Next_Time:=Next_Time + . . .
    Delay_Until_And_Set_Deadline(
       Next_Time, Next_Time+5ms)
  end loop;
end T1;
```

```
task body T2 is
begin
  Next_Time:=Clock
  Set_Deadline(Next_Time+30ms)
  loop
    Action1_4ms
    Critical_Section_3ms
    Action2_1ms
    Next_Time:=Next_Time + . . .;
    Delay_Until_And_Set_Deadline(
       Next_Time, Next_Time+30ms)
  end loop;
end T2;
```

Two scenarios were considered. In the first scenario, the task T1 runs to its worst case (but no further). As illustrated in Figure 1.

Here, the relative deadline of the tasks must be set carefully to reflect its relative deadline following the self suspension. If this is done then the DFP behaves as required.

The second scenario, shown in Figure 2 illustrates the behaviour of the protocol if T1 overruns its execution time. In this scenario mutual exclusion is not assured.

The workshop discussed the pros and cons of the DFP and SRP protocols and concluded that for the general case DFP with Locks is still preferable to SR due to its simpler implementation and understandability.

Further, with the DFP it was possible to have a uniform two level scheduling approach. At the top level is fixed priority scheduling. Within priority, FIFO, EDF or round-robin scheduling can be used. This can be contrasted to having EDF across priorities to support SRP. Further consideration is needed as to whether, with a lock, trying to access a PO becomes a dispatching point.

If when a task calls a PO it is suspended, then the task with the lock could inherit the urgency of the suspended task. This was, however, felt not to be a necessary property and the workshop did not go as far as to recommend it.

3 Representation of Deadlines

The workshop discussed where to define the deadline types. Two alternatives to the ones proposed in the Burns and Wellings paper were considered. The first was to deprecate `Ada.Dispatching.EDF` and add the types to `Ada.Real_Time`. The workshop felt that this sends the wrong signal to the community. The second alternative was preferred by the workshop, which was that of adding new types to `Ada.Dispatching.EDF` Hence:

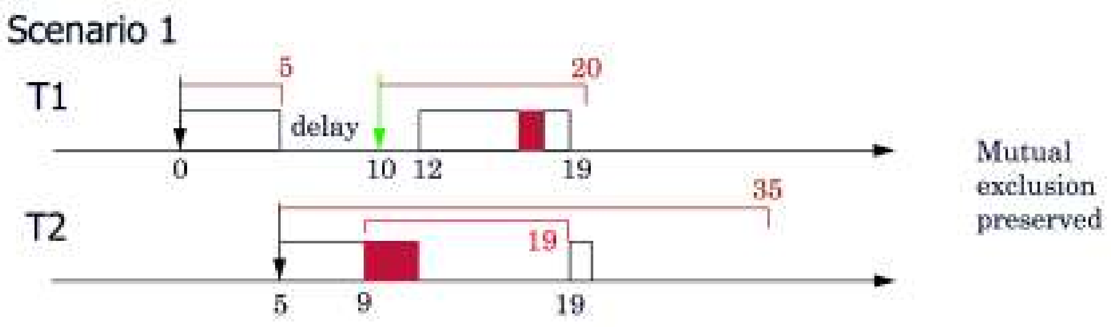


Figure 1: First Scenario.

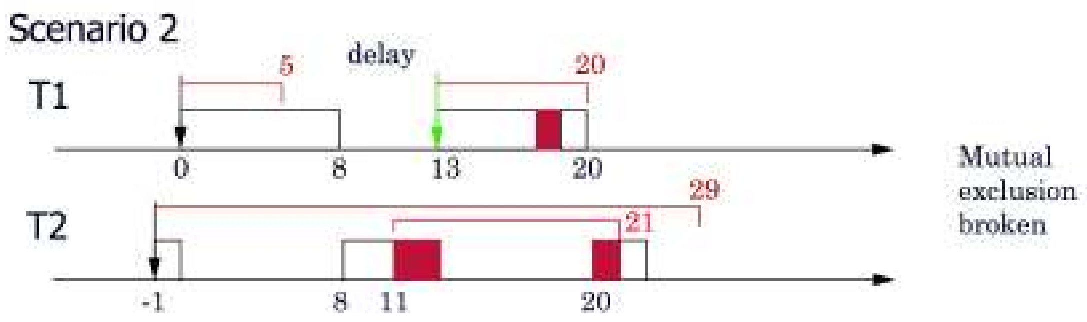


Figure 2: Second Scenario.

```
with Ada.Real_Time;
with Ada.Task_Identification;
package Ada.Dispatching.EDF is
  .....
  subtype Relative_Deadline is Real_Time.Time_Span;
  Default_Relative_Deadline : constant
       Relative_Deadline := Real_Time.Time_Span_Last;
   procedure Set_Relative_Deadline
       (D : in Relative_Deadline;
        T : in Ada.Task_Identification.Task_Id :=
             Ada.Task_Identification.Current_Task);
  function Get_Relative_Deadline
       (T : Ada.Task_Identification.Task_Id :=
               Ada.Task_Identification.Current_Task)
       return Relative_Deadline;
end Ada.Dispatching.EDF.Dynamic_Relative_Deadlines;
```

4 Deadline Aspect for PO

Each PO needs a deadline floor attribute what can be set on the creation of a PO by as aspect. The `relative_deadline` aspect already exists, so it was felt appropriate to reuse this aspect to set the deadline floor. This is identical to the way that the `priority` aspect is used both for task priority and PO ceiling priority.

5 Summary of Workshop position

The Workshop's position can be summarised as follows.

- There was unanimous support for developing an AI for DFP.
- The AI should recommend the deprecation of the SRP protocol.
- The AI should define the use of a mutex lock to obtain mutual exclusion rather the relying on the correction of the application in its setting of deadline floor.

References

[1] A. Wellings (2013), *Session summary: Locking protocols*, ACM SIGAda Ada Letters, Proc. of IRTAW 16, XXXIII(2):123–125.

Session Summary: Language Issues

Chair: Michael González Harbour

Rapporteur: Mario Aldea Rivas

1 Introduction

The goal of this session was to discuss and, if appropriate, generate Ada Interpretations for several language related issues presented to the workshop:

- Extension of the Synchronous Task Control in order to allow the use of Suspension Objects by concurrent tasks.
- Inclusion of Synchronous Barriers in the Ravenscar profile.
- Addition of execution timer and group budget support for interrupt handlers.
- Issues on High-Integrity Dynamic Memory Management.

2 Synchronous Task Control

The session chair summarized the issue presented in [1]. The original motivation of the Suspension Objects defined in the Ada.Synchronous_Task_Control package is the implementation of efficient suspend and resume operations. They are intended to be used on a per-task basis: each Suspension Object is dedicated to having just one task to suspend upon it. In particular, Program_Error is raised upon calling Suspend_Until_True if another task is already waiting on that Suspension Object.

The ARG has asked the IRTAW community to consider the extent to which Synchronous Task Control should be used by concurrent tasks. This extension would allow the use of Suspension Objects as general binary semaphores:

```
Suspend_Until_True(Sem);
   -- critical section
Set_True(Sem);
```

It is important to notice that with the current definition of the Synchronous Task Control in the Reference Manual, this use is not possible even with two tasks because concurrent calls to Suspend_Until_True are not defined to be atomic with respect to each other. However, Program_Error would not be raised in this case as there could only be one task suspended at any moment.

Three possible modifications could be made:

1. Disallow concurrent calls to Suspend_Until_True.
2. Allow concurrent calls, and define them to be atomic (which would allow the use of Suspension Objects as binary semaphores to be used by two tasks at most).
3. Allow concurrent calls, define them to be atomic and remove the restriction as to there being at most one suspended task per Suspension Object.

2.1 Discussions

There was little discussion on this issue since there was a wide agreement on not changing the original semantics of the Suspension Objects as a mechanism to implement efficient suspend and resume operations on a per-task basis. Consequently, modifications 2 and 3 were quickly discarded.

During the discussion an issue was raised on whether or not the Ravenscar profile should include the restriction "No local suspension objects" (in Ravenscar a Suspension Object local to a task would be useless since it would not be visible to any other task in the system). The general agreement was the new restriction was not worthy and a compiler warning will be a coherent response to this situation.

2.2 Conclusions

The main conclusion of this part of the session was to support the modification number 1: maintain the original semantics of the Suspension Objects and define as an bounded error the situation where two tasks make concurrent calls to Suspend_Until_True.

3 Synchronous Barriers

The issue presented in [1] was summarized by the session chair. Synchronous barriers allow synchronously releasing a group of tasks after the number of blocked tasks reaches a specified count value. When synchronous barriers were introduced into the language it was decided not to include them in the Ravenscar profile.

As it is shown in [1] they can be implemented with Ravenscar primitives (using protected objects and suspension objects). But if the underlying platform directly supports a barrier primitive then more efficient code can be generated if they are directly supported by the language

The question for the IRTAW was whether this potential efficiency gain is sufficient to warrant the inclusion of synchronous barriers in Ravenscar.

3.1 Discussions

During the discussion, the workshop did not find objections to including this functionality in Ravenscar but it was noticed that there is no industrial experience on the use of the Ravenscar profile for multiprocessors. AdaCore's first Ravenscar run-time for multiprocessor (for the Leon architecture) is expected for this year.

3.2 Conclusions

The workshop did not have any objections to including this functionality in Ravenscar. However, the general feeling was that more extensive industrial experience on the use of Ravenscar on multiprocessors would be desirable before taking a strong opinion about including this functionality in Ravenscar.

4 Execution time control for interrupt handling

The session chair presented a brief overview of the proposal made in [2]. Support for the separate accounting of the execution time of interrupt handlers was included in the 2012 revision of the Ada language but no mechanism was included to control the execution time dedicated to interrupt handling. In order to fill this gap, the proposal argues for the inclusion of interrupt execution time timers and group budgets.

Kristoffer Nyborg Gregertsen made a detailed description of his proposal that is a refinement of a previous one discussed at a previous IRTAW [3]. Several issues were identified on that previous work [4] which are addressed in the current proposal by defining a new tagged type hierarchy that integrates execution time timers for tasks and interrupts and another hierarchy to integrate group budgets for tasks and interrupts.

These unified hierarchies allow a single implementation of an execution time control policy, such as the deferrable server, to work for both tasks and interrupts.

The main disadvantage of this approach is that the proposed API is not backwards compatible with the current Ada specification due to the modifications in the packages Ada.Execution_Time.Timers and Ada.Execution_Time.Group_Budgets, respectively replaced by Ada.Execution_Time.Timers.Tasks and Ada.Execution_Time.Group_Budgets.Tasks in the proposal.

4.1 Preliminary discussions

Discussions were centered on whether the uniformity provided by the proposed hierarchies compensates the loss of backwards compatibility. The workshop's feeling was that backwards compatibility should be maintained even at the expense of jeopardizing the uniformity of the types.

An alternative approach briefly outlined in the same paper [3] was also discussed. It is based on the definition of the types Root_Timer and Root_Group_Budget in their own packages. With this proposal the names of the packages Ada.Execution_Time.Timers and Ada.Execution_Time.Group_Budgets would be preserved. This approach would require less changes to existing code than the original proposal, but it is not fully backwards compatible either.

Kristoffer Gregertsen was asked to present a revised proposal that would preserve backwards compatibility.

4.2 Revised proposal

Kristoffer Gregertsen presented a revised proposal that does not modify the Ada.Execution_Time.Timers and Ada.Execution_Time.Group_Budgets packages and, in consequence, is fully backwards compatible. The type Interrupt_Timer is defined in a new package Ada.Execution_Time.Interrupts.Timers. This type is identical to the task timers type with exception that its discriminant would be an interrupt ID.

The same approach would be applied to group budgets.

4.3 Final discussions

Once the backwards compatibility problem was overcome, the discussions centered on the usefulness and efficiency of the proposal.

It was pointed out that the possible actions on a misbehaving handler are very limited since the interrupt can be disabled but the running handler cannot be aborted. In consequence, the functionality provided by the interrupt execution time timers could be achieved by the programmer by measuring the interrupt clock at the end of the handler and performing the appropriate actions if an overrun is detected. Although this ad-hoc implementation is feasible, the inclusion of the interrupt execution time timers would ease the programmer's task.

Some concerns were raised about the overhead introduced by the implementation of the mechanism to control the execution time of the interrupts. Kristoffer Gregertsen's experience with his implementation on the Atmel AVR32 architecture showed that the overhead is acceptable, and could be further improved by using a specialized Time Management Unit available for that architecture. Some doubts remained about the possibility of implementing this functionality efficiently on other architectures or in an Ada run-time running on top of an operating system.

A vote was taken on whether the workshop considered the proposal useful, with the result of 5 yes, 1 no and 9 abstentions.

The general opinion of the workshop was that the proposal should integrate possible multiprocessor related issues.

During the discussion an observation was raised pointing out that some advanced aspects of the language (notably those defined in Annex D) are not implemented in the most accessible Ada platforms, i.e., the compilers and run-times for general purpose operating systems as Linux or Windows. This fact could discourage some newcomers who wanted to gain experience with these aspects of the language. However these platforms are not designed for real-time applications. It was suggested that new profiles could be defined for this kind of systems in order to clarify the functionality that a user can expect to find in these environments.

4.4. Conclusions

The general opinion of the workshop was that Interrupt Timers are an interesting functionality that deserves being explored. For a full acceptance of this functionality the multiprocessor implications must be understood. The workshop encourages further research in this area in order to be discussed at a future IRTAW.

5 High-Integrity Dynamic Memory Management Issues

Andy Wellings presented the dynamic memory allocation model used in Safety-Critical Java (SCJ) and his experience with implementing this model in Ada [5]. SCJ supports an application structure based on the notion of "missions". An application is a sequence of missions, each mission is comprised of a set of real-time concurrent activities, and each activity executes a sequence of jobs.

In SCJ, dynamic memory allocation can be performed in different memory areas with different lifetimes:

- Immortal memory area: objects allocated in this area will remain for the lifetime of the application.
- Mission memory area: objects can be allocated in this area, but when the mission is finished all the memory is reclaimed and the block reused by the subsequent mission.
- Per-release memory area: memory area associated to each real-time activity. Objects allocated here can only be accessed by the owner real-time activity. The per-release memory is cleared at the end of each release
- Private memory areas: they can be used to store objects that have an even more limited lifetime, perhaps for the lifetime of a method call.

Different Ada implementation models have been explored based on global and local memory pools and Andy raised several issues to be discussed.

5.1 Discussions

The first identified issue was related with the laxity of the RM (Ada Reference Manual) about the automatic storage reclamation: there is no requirement on an implementation to cleanup the memory when an access types goes out of scope. Of course, Unchecked_Deallocation can be used, but this requires the program to keep track of all objects created. It was noticed that real-time garbage collection has improved a lot in the last years and nowadays it is plausible to use it in real-time applications.

Another identified issue is the lack of control that an application has on the allocation of the task stack. Two limitations are detected: (a) there is no way specified in the RM to monitor the size of the stack used by a task (although AdaCore provides a non-standard API), (b) an application can set the stack size of a task but there is no way to control how the stacks are allocated, and consequently no way that they can be integrated with storage pools.

Finally, Andy Wellings noticed that, in order to correctly implement the SCJ memory model in Ada, it would be necessary to avoid the use of the default storage pool. It was suggested that a new restriction could be added to the language to force applications to behave in that way. A finer granularity restriction, applicable to the package level, was discarded by the workshop since it was considered that, when needed, this should be checked using an external tool.

5.2. Conclusions

The main conclusions of this part of the session were:

- It would be interesting that the language provided an aspect for the access types in order to specify that the memory allocated to dynamic objects of that access type should be reclaimed when the type goes out of scope. If the type has an associated storage pool then the compiler would generate appropriate calls to the deallocation operation. A tentative name for that aspect could be "Reclaim".
- The language (in its Systems Programming Annex) should provide more control for stack allocation and deallocation. Two alternatives could be acceptable:
    1. The language should provide an aspect to force the implementation to include the stack space in the same pool than the access type of the task (of course, only in case the access type of the task is associated to a pool).
    2. The language should provide an aspect to specify a storage pool for allocating the stack of the tasks of a given task type.
- The language (in its Systems Programming Annex) should provide an API to dynamically get the current stack size used and the maximum stack size used for a task.
- There should there be a requirement for an implementation to document where the stack space for handling interrupts is allocated.
- A new restriction should be added (No_Default_Storage_Pool_Allocation) to indicate that dynamic allocation can only be done for objects whose access-to-object type is associated to a user defined storage pool.

References

[1] A. Burns and A.J. Wellings (2016), *Synchronous Task Control and Synchronous Barriers*, In Ada Letters 36(1):35-38, ACM New York.

[2] K. N, Gregertsen, *Revising the Ada timers and group budgets to support execution time control for interrupt handling*, In Ada Letters 36(1), ACM New York.

[3] K. N. Gregertsen and A. Skavhaug (2013), *Execution time timers for interrupt handling*, Ada Letters 33(2):87–96, A CM New York.

[4] T. Vardanega and R. White (2013), *Session summary: improvements to Ada*, Ada Letters, 33(2):126–130, ACM New York.

[5] A. J. Wellings, V. Cholpanov and A. Burns (2016), *Implementing Safety-Critical Java Missions in Ada*, In Ada Letters 36(1):51-62, ACM New York.

Session Summary: Ada Language Profiles

Chair: Tullio Vardanega

Rapporteur: Pat Rogers

Abstract

This session of the 18th International Real Time Ada Workshop discussed the use of profiles in Ada, and the possibility and desirability of adding additional language profiles to Ada besides the Ravenscar Tasking Profile.

1 Introduction

The "Profiles" session examined various profiles, official and unofficial, that are used in Ada. The goal was to determine the desirability of formalizing language profiles as was done with great success for the Ravenscar Tasking Profile.

2 Discussion

2.1 Ravenscar Tasking Profile

The session began with a summary provided by Tullio recalling the history of the Ravenscar profile, emphasizing, in particular, that Ravenscar has become a "brand" that we want to ensure remains well-defined and viable. In short, we should not "tamper" with the brand. This point was particularly pertinent to the discussion regarding new profiles based on the current definition of the Ravenscar profile.

2.2 Zero Footprint

The workshop then entertained the question of whether we should define a profile based on the Zero Foot Print (ZFP) runtime "profile" separately provided by AdaCore [3] for embedded systems running specialized kernels. The consensus in the group was against doing so.

2.3 Linux Targets

There was a discussion on the feasibility and desirability of having a profile specifically intended to run on Linux (for reasons of wide availability on hardware platforms that could be used for experiments, teaching, research). The point of that effort would be to determine which constructs/features in the language are problematic for efficient implementation on Linux so that we could then define a profile without those constructs/features (via restriction policies). No decision was reached on that discussion item.

2.4 Multicore/Parallelism

There was general agreement that multicore/parallelism was an important new area, and that there was potential for a profile to support multicore applications. The point was made that we need a forum, or attention topic in some community domain, in which our work will have high visibility to, and interest from, the "non-Ada" (real-time/embedded) community. Tullio suggested that multiprocessing/parallelism is just such a forum and this was widely agreed.

Regarding multiprocessing/parallelism, there was consensus that in the future we should define a parallel version of Ravenscar. Submissions to future workshops on this topic would be welcomed.

2.5 Earliest Deadline First Scheduling and Timing Analyzability

The workshop agreed that Earliest Deadline First scheduling would also be a good candidate for our a profile with analogous forum for our work. There was a question of whether we want to define a subset of the sequential part of the language for the sake of timing analysis. There was no general support for doing so.

2.6 Candidates for a Profile

As part of figuring focal points of community attention, the workshop discussed various topics without exploring them deeply, including how to have mixed-criticality applications on the same runtime (as opposed to a partitioned OS with a distinct runtime per partition). The attractiveness of asynchronous select statements for resiliency was noted, although the problematic aspects of that construct were recognized; what might be required for parallelism, and so forth. There was general consensus that this whole area has potential but needs deeper thinking. It might be a valid discussion topic for the next workshop.

2.7 Extended Ravenscar Profile

The discussion moved on to the new profile under development by AdaCore, which was presented at the previous IRTAW [4]. As that profile allows for multiple calls to queue on a protected entry (where Ravenscar restricts that to at most one). A pertinent question that the workshop identified was to determine whether or not the Ada Reference Manual [2] requires open protected entries (with pending calls) to be serviced in priority order within a protected action. On scrutiny, no such requirement was found in the manual, Yet, it was recalled that group consensus at the previous meeting was that FIFO queuing should be preferred, which puts the priority-ordering question to sleep unless good reasons are found to revive it.

As part of the discussion, the comment was made that an AdaCore extended "Ravenscar" might possibly obviate the current Ravenscar profile, in that no-one would want to use the more restricted profile given access to a less restrictive alternative. The AdaCore representative indicated that such was definitely not the intention, and that they would maintain the position that both profiles are appropriate. Indeed, AdaCore's new profile does not preclude the use of the canonical Ravenscar profile. Although the extended profile is planned to be the default, programmers can specify Ravenscar with the pragma Profile to get both the canonical Ravenscar semantics and the other benefits, such as maximized efficiency and analysis for protected entry handling.

Although a program conformant to the canonical Ravenscar profile would run successfully on the extended profile, the new capabilities were thought to be sufficiently different in "flavor" as to warrant a nonhierarchical profile name, i.e., one not including "Ravenscar" as part of the name.

By the end of the session, no decision was made for a name for the new profile, however we took up the topic again on Wednesday. At that time the name "Dewar" was proposed and was agreed by the group to be a good suggestion. AdaCore management will be consulted on this matter.

2.8 Hierarchical Names for New Profiles

The workshop took up the question of names for new profiles. It was agreed that those profiles that are sufficiently similar to an existing profile should use hierarchical names, i.e., with the extended profile as the parent name for an extension. In addition, consensus was reached that a test for being "sufficiently similar" would be whether a program conforming with the parent would run successfully on the extension. Successfully passing that test was agreed to be necessary for the name of the extension to include the existing profile name. In particular, it was agreed that one cannot remove functionality from an existing profile and consider the result to be an extension of that profile. This notion does not fully fit a possible EDF version of the Ravenscar profile, which would in fact build on exactly the same set of restrictions casting them to EDF scheduling, combined with the Deadline Floor Protocol, instead of fixed-priority scheduling, combined with the priority ceiling protocol. The name that was contemplated for that profile would likely still be Ravenscar-EDF. No formal decision was made on this issue, however.

3 Conclusions

The session was enjoyable and had active participation, with lively discussions and decisions that will steer the path of future work on Ada language profiles. The discussion held were more strategic in nature than deeply technical. It is likely – and it is in fact expected – that future editions of the workshop will see authors table concrete technical proposals for novel language profiles along the lines anticipated in this session.

References

[1] Garrido J., Lacruz B., Zamorano J., de La Puente J.A. (2016), *In Support of Extending the Ravenscar Profile*, Ada Letters 36(1): 63-67, ACM New York.

[2] International Standards Organization (2012), ISO IEC 8652 "*Information Technology – Programming Languages – Ada*".

[3] AdaCore, Gnat Pro, *Runtime Profiles*, http://www.adacore.com/gnatpro/toolsuite/runtimes/

[4] P. Rogers, J. Ruiz, T. Gingold (2015), *Toward Extensions to the Ravenscar Profile*, Proceedings of the 17th International Real-Time Ada Workshop, April 2015, Ada Letters 35(1):32-37, ACM New York.

Session Summary: Experience

Chair: Jorge Real

Rapporteur: Pat Rogers

1 Introduction

The Experience session examined the proposal [1] to integrate time-triggered scheduling with priority based scheduling.

2 Discussion

The session began with an overview of time-triggered (TT) scheduling, in which tasks are scheduled using static, predetermined "plans" in order to minimize release jitter. After this overview, Jorge described a software architecture and implementation that combines TT tasks with those scheduled dynamically by priority (priority-based, PB tasks). In particular, he presented the model for the approach, the interface for the TT scheduler, and selected implementation details. He closed with descriptions of selected patterns combining TT and PB tasks and showed experimental evidence of the results under MaRTE OS [2] in bareboard configuration.

Jorge noted that TT tasks experience higher minimum jitter than PB tasks, due to the TT scheduler overhead, but overall the TT tasks experience much less jitter since they are not subject to interference from higher priority tasks.

The presentation and approach were both very well received.

During the presentation a number of points and questions arose. The resulting discussions are presented in the following paragraphs.

The implementation is not currently compatible with the Ravenscar profile restrictions. For example, it uses dynamic priorities to "demote" overrunning tasks or tasks that explicitly leave the TT level. (Demoted tasks continue to execute, but at a selectable, harmless priority level.) Similarly, it uses timing events that are declared locally, rather than at the library level, and entry families. That said, it should be possible to rewrite the implementation to be compatible.

The patterns combining time-triggered and priority-based tasks received considerable attention, especially the "Initial-Optional-Final" pattern because it suggests others. For example, the "Optional" part could just as easily be considered required, but since it is executed under priority-based scheduling it would allow the benefits of priority-based scheduling for that part of the plan while retaining the benefits of time-triggered scheduling for the "Initial" and "Final" parts.

The discussion led to the general point that one should be able to divide a long-running time-triggered task into segments that would be executed across several slots. A TT task following this pattern and failing to complete in one particular slot is not an overrun, provided that the plan includes more slots in the future for the completion of the TT task's activity. Spreading the TT task execution across several slots gives chances for other tasks to execute in between those slots. This enhancement was considered important by the Workshop.

The issue of multiprocessor support was discussed again, as it had been for essentially all other sessions. The current implementation approach is to have one TT plan per processor, with tasks fixed to processors, a reasonable approach given that this is a static, off-line scheduling regime. In a multiprocessor setup, there would be up to one statically allocated plan per processor. In this context, it was argued, it would be much convenient to be able to set the affinity of timing events, as proposed in [3]. However one could allow TT tasks to migrate across plans when a sufficient TT slot is not available in the current plan but can be found in a plan running on a different processor, thereby migrating jobs across processors. This was considered a valuable feature.

3 Conclusion and Follow-up

The session closed with a discussion of what to do next. Should the implementation be part of a library, a new profile, made publicly available as-is? No specific decision was reached, but the proposed enhancements were re-confirmed as valuable and additional directions and refinements were proposed by the group. In particular:

- Explore the ability to break a long-running TT task into segments, get some usage experience, and then make the resulting facility available.
- Explore an integration with a real-time framework, e.g., those in [4, 5, 6].
- Explore additional patterns, in particular an "Initial-Required-Final" pattern as described above.

References

[1] J. Real, S. Sáez and A. Crespo (2016), *Combined Scheduling of Time-Triggered Plans & Priority Scheduled Task Sets*, Ada Letters 36(1):68-76, ACM New York.

[2] M. Aldea, M. and M. González-Harbour (2001), *MaRTE OS: An Ada Kernel for Real-Time Embedded Applications*, Reliable Software Technologies - Ada Europe 2001, LNCS 2043, pp. 305–316.

[3] S. Sáez, J. Real and A. Crespo (2015), *Implementation of Timing-Event Affinities in Ada/Linux*, Ada Letters 35(1): 80-92, ACM New York.

[4] A. J. Wellings and A. Burns (2007), *A Framework for Real-Time Utilities for Ada 2005*, Ada Letters 27(2), ACM New York.

[5] J. Real and A. Crespo (2010), *Incorporating Operating Modes to an Ada Real-Time Framework*, Ada Letters 30(1) : 73–85, ACM New York.

[6] S. Sáez, J. Real and A. Crespo (2012), *An Integrated Framework for Multiprocessor, Multimoded Real-Time Applications*, In Proc. Reliable Software Technologies -- Ada-Europe 2012. LNCS 7308, Springer.

Session Summary: Time Vulnerabilities

Chair: Stephen Michell
Rapporteur: Juan Antonio de la Puente

1 Introduction

The aim of the session was to identify and discuss time vulnerabilities in Ada at the request of ISO/IEC SC22 WG 23, in order to update the Ada part in the TR 24722 document [1]. The basis for the discussion was the paper by Stephen Michell on *Time Issues in Programs Vulnerabilities for Programming Languages and Systems* [3]. The vulnerabilities identified in the paper were discussed, as described in the next sections. The reader is referred to the full paper for details on the definition of the different vulnerabilities. The vulnerability on 'external visibility of usage parameters' was not discussed as it was not considered a time vulnerability.

The group agreed on grouping the time vulnerabilities described in the paper as follows:

- Clock issues
- Resource consumption errors
- Missed events

The vulnerabilities are described in the next sections of this report

2 Clock issues

This vulnerability is related to issues caused by clock handling, such as synchronization between clocks, representation of time as dispensed by a clock, and non-monotonic behaviour of some clocks

2.1 Description of vulnerability

Errors in clock synchronization, time conversion, and clock roll-over may result in incorrect behaviour in programs depending on time, possibly leading to application failure.

Examples of such errors are:

Differing time bases. Multiple clocks are often available, with different notions of time (e.g. calendar and time of day, seconds from epoch, elapsed time, execution time) in the same system. Different clocks usually have different representations, scaling and semantics, which may give raise to conversion, rounding and cumulative errors leading to application failures.

Clock rollover. Time is usually represented using a fixed length of bits. As a consequence, there is a possibility that a clock rolls over, which may lead to application failure as time values come back to a smaller value.

Synchronizity issues. Local clocks on multiprocessor systems will drift with respect to each other after some time. The possible consequences are errors in time comparisons and different time values in different parts of an application, which in turn may give rise to missed events, lost deadlines and communication errors, eventually resulting in potential application failures. A further consequence of this issue is that using time stamps to guarantee order is not reliable.

The key issue with respect to synchronizity is whether the programming language (and the platform underneath) provides clocks with bounded drift.

2.2 Possible mitigations

- Understand the differences between different time bases and develop appropriate conversions.

  In particular, always convert time values from the most precise and stable time bases to less precise ones. For instance, avoid converting from time-of-day clocks or network time to real time clocks.

- Protect application code against clock rollover, e.g. by detecting when a time value is near the highest possible value and taking into account this possibility.
- Account for communication delays and relative clock drift in communicating tasks across multiple CPUs.
- Measure drift between clocks periodically on multiprocessor platforms.
- Use only clocks with known synchronicity properties

3 Resource consumption issues

This vulnerability deals with issues of resource consumption errors, including vulnerabilities associated with monitoring resources, and vulnerabilities associated with changes to resource consumption due to issues such as virtualization, cache effects and processor speed changes.

3.1 Description of vulnerability

Some applications depend on measurements of time associated with monitoring resource usage. Changes in clock rate, processor speed, or errors related to execution time monitoring may cause failures possibly leading to total application failure. Some specific issues associated with this vulnerability are:

Virtualization issues. In a virtualized system, virtual clocks are used instead of physical clocks. This may lead to virtual clocks running faster than normal in order to catch up with real time, which may result in synchronization errors or events not being received. Another issue is possible interference from other applications with a high load generated either accidentally or by external attackers. In this situation the amount of available resources for a critical application may not be enough for it to execute correctly.

Concurrent setting of real-time resources. Real-time systems must interact with low-level resources such as hardware timers, interrupts, or external events. Errors can arise if calls to system services related to these resources are not protected against concurrent access, which may resulting in an incorrect temporal behaviour of the system, possibly leading to system failures.

Time accounting issues Time accounting may be affected by system services, e.g. garbage collection, or by the presence of multiple CPUs. Execution time inaccuracies, e.g. when some operations are executed after reading the value of a clock with an unknown effect.

3.2 Possible mitigations

- Take steps to guarantee that processors, memory and time resources are locked to the application and not shared with other virtual services.
- Do not virtualize critical applications.[1]
- Protect system-level operations on timers, interrupts, and events against concurrent access.
- Validate any assumptions about time accounting mechanisms.

4 Missed event issues

This vulnerability deals with issues arising from missed deadlines or events related to the scheduling and monitoring of work based on time.

4.1 Description of vulnerability

Real-time systems must react to external events, or perform actions at specific times, within specified deadlines. Failing to do so, or to properly monitor the time behaviour of the system, may lead to catastrophic failures.

Specific issues in this area include:

Missed deadlines. This is a typical vulnerability of systems with a cyclic behaviour. If the work allocated to a cycle does not end in time and overruns into the next cycle, the behaviour of the system may be seriously compromised and possible lead to an application failure.[2]

[1]It was noted that critical applications can be virtualized on specific kernels such as XtratuM [2].

[2]It should be noted that this is a scheduling issue, rather than a clock or time one, although it may be caused by clock errors.

Iteration scheduling. This issue is related to programming the start time of the next iteration in a real-time system consisting of periodic and sporadic tasks. If the start time is computed using a non-real-time clock, or is based on the completion time of the last iteration, jitter in the start time of the task operations may arise, resulting in an improper time behaviour.

4.2 Possible mitigations

- Improve analysis to detect potential overruns in cyclic systems.
- Program using a more flexible, priority-based, scheduling approach.
- Base next wake-up on previous programmed wake-up time.
- Only use real time monotonic time clocks to schedule events.

5 Conclusions

The group concluded that the above vulnerabilities should be confirmed to WG23, with the comments on mitigations that have been made in the meeting.

References

[1] International Standards Organization (2013), ISO/IEC. *TR 24772:2013 — Information Technology — Programming Languages — Guidance to avoiding vulnerabilities in programming languages through language selection and use.*

[2] M. Masmano, I. Ripoll, A. Crespo, and J. Metge (2009), *Xtratum: a hypervisor for safety critical embedded systems*, In 11th Real-Time Linux Workshop, pages 263–272.

[3] S. Michell (2016), *Time issues in programs vulnerabilities for programming languages and systems*, In 18th International Real-Time Ada Workshop, IRTAW-18, Benicàssim, Spain.

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 in Sweden

attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.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