

ADA USER JOURNAL

Volume 40
Number 3
September 2019

Contents

	<i>Page</i>
Editorial Policy for <i>Ada User Journal</i>	130
Editorial	131
Quarterly News Digest	132
Conference Calendar	143
Forthcoming Events	149
Special Contribution	
J. Cousins <i>"ARG Work in Progress III"</i>	153
Ada-Europe 2019 Industrial Presentations	
A. R. Mosteo <i>"RCLAda, or Bringing Ada to the Robotic Operating System"</i>	159
Proceedings of the "Workshop on Challenges and New Approaches for Dependable and Cyber-Physical Systems Engineering" of Ada-Europe 2019	
M. Schranz, M. Sende, A. Bagnato, E. Brosse, A. Eckel <i>"Modeling CPS Swarms: An Automotive Use Case"</i>	165
M. Schranz, M. Sende, A. Bagnato, E. Brosse <i>"Modeling Swarm Intelligence Algorithms for CPS Swarms"</i>	169
Ada-Europe 2019 Speaker's Corner	
J. P. Rosen <i>"Experience in 40 Years of Teaching Ada"</i>	179
Article	
M. Gajdzica <i>"Ada-Europe 2019 – Newcomer Experience"</i>	183
Ada-Europe Associate Members (National Ada Organizations)	186
Ada-Europe Sponsors	Inside Back Cover

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

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

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

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

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

I must start this editorial by presenting myself, as the new Editor-in-Chief of the Ada User Journal. I'm Associate Professor at the Department of Informatics of the University of Lisbon Faculty of Sciences, and a member of the LASIGE Research Unit, where I lead the research line on Cyber-Physical Systems. My involvement in the Ada community is quite recent, despite my first contact with the language itself being dated back to 1998, when I was teaching Ada 95 to undergraduate students. Last year I organized and was the Program Chair for the 2018 Ada-Europe Conference in Lisbon, which gave me the opportunity to realise how vast is the community, to learn about the many past and ongoing projects with strong Industry involvement in which reliable software technologies and Ada in particular are being used, and to witness the strong involvement of companies and entities working to promote and evolve the language. After the conference I was kindly invited to follow the activities of the Ada-Europe Board and eventually was proposed and elected as one of its members, earlier this year. Almost without noticing, I became involved!

Self-presentation done, my first words are of appreciation to Luís Miguel Pinho, who served as Editor-in-Chief of the Ada User Journal for 12 years and is now passing me the token. The great work he did, along with the Ada User Journal editorial team, to provide us 48 high-quality issues, sets a huge challenge and responsibility that I'm well aware, an obligation to pursue that work and keep the Journal high-quality standards to which the readers have been made used to. My promise is to do my best, looking for ways to improve or renovate the Journal contents and presentation, aiming to satisfy the Journal reader. For this task I'm fortunate to count with the help of knowledgeable, competent and committed people: Patricia López Martínez, Jorge Real, Dirk Craeynest, Kristoffer Nyborg Gregertsen and Alejandro R. Mosteo. And the experienced advice of Miguel, who will continue to serve the Ada User Journal as Associate Editor, is reassuring.

Regarding the contents of this issue, we start with an article that reports on the work of the Ada Rapporteur Group (ARG), written by Jeff Cousins, member and former chair of the ARG. Being the third article in the series, it presents a further update of the forthcoming revision of the Ada language, Ada 2020, following previous articles published in March 2017 and one year ago.

We continue the publication of the proceedings of the industrial track of the 2019 Ada-Europe Conference, this time with a single article, by Alejandro R. Mosteo. The article is about bringing the Ada language to the Robot Operating System (ROS) through the provision of RCLAda, which consists of an API and accompanying tools.

Then we start the publication of the proceedings of the workshop on Challenges and New Approaches for Dependable and Cyber-Physical System Engineering (DeCPS 2019), which was co-located with the 2019 Ada-Europe conference. The reader will find two papers, both related to modelling swarm behaviour in Cyber-Physical Systems. The first one focuses on the application of existing models to an automotive use case and the second proposes an approach to model the local behaviour of individual CPSs using swarm intelligence algorithms.

Finally, the issue includes two articles that directly derive from the 2019 Ada-Europe conference. During the conference, Jean-Pierre Rosen gave a talk on his Experience in 40 Years of Teaching Ada, in a special session named "The speaker's corner". This talk originated a paper that we now have the pleasure to publish. We also publish a paper that, interestingly, and in contrast, provides the perspective of a newcomer to the Ada-Europe conference. It is authored by Maciej Gajdzica, who, being a first-time attendant, shares his impressions on several aspects of the conference, from the venue to the talks in the technical and vendor sessions.

Last but not the least, this issue includes the usual News Digest and Calendar sections, prepared by their respective editors, Alejandro R. Mosteo and Dirk Craeynest.

*Antonio Casimiro
Lisboa
September 2019
Email: AUJ_Editor@Ada-Europe.org*

Quarterly News Digest

Alejandro R. Mosteo

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

Contents

Ada-related Organizations	132
Ada-related Events	132
Ada-related Resources	133
Ada-related Tools	134
Ada-related Products	136
Ada and Operating Systems	137
Ada and other Languages	138
Ada Practice	139

Ada-related Organizations

Additional Comment Period for Upcoming Ada Revision

From: "Randy Brukardt"
<randy@rrsoftware.com>
Subject: Additional Comment Period for Upcoming Ada Revision
Date: Fri, 26 Jul 2019 21:53:30 -0500
Newsgroups: comp.lang.ada

ISO/IEC JTC 1/SC 22/WG 9 (WG 9) is responsible for the maintenance and revision of the Ada Programming Language and associated standards and technical reports. As part of the language maintenance activity, WG 9 has established a group of Ada experts as the Ada Rapporteur Group (ARG). The ARG receives input from the Ada community at large to consider for inclusion in revision to the Ada programming language standard. The WG 9 has produced a number of revisions to the language in accordance with ISO policy and to address the evolution of technology (Ada 83, Ada 95, Ada 2005 and Ada 2012).

Presently, the ARG is nearing completion on a revision to Ada 2012 (known for now as Ada 202x) which includes new contracts and lightweight parallelism features. Concern has been raised that these new proposals have not been prototyped nor has the suitability for diverse target environments been assessed.

Therefore, the ARG is seeking comments, based on prototyping and review, on the new features (focused on the parallelism features) incorporated within the current draft of the Ada 202X standard. Comments should be submitted to ada-comment@ada-auth.org as described in

the Ada Reference Manual Introduction (http://www.ada-auth.org/standards/rm12_w_tc1/html/RM-0-3.html#p58). Please include the draft number with any Ada Reference Manual references in your comment. Comments should be sent by 1 June 2020 in order to be considered for the revision. (Note: While not required, joining the mailing list as described at <http://www.ada-auth.org/comment.html> is recommended so that you receive any queries on or responses to your comment.)

The draft revision can be found at <http://www.ada-auth.org/standards/ada2x.html>.

A list of issues addressed in Ada 202x can be found at http://www.ada-auth.org/ai-files/grab_bag/2020-Amendments.html.

(You can find an on-line version of this announcement at <https://www.adaic.org/2019/07/additional-comment-period-for-upcoming-ada-revision/>.)

From: "Randy Brukardt"
<randy@rrsoftware.com>
Date: Fri, 26 Jul 2019 22:02:39 -0500n

To translate this announcement into plain English, the completion date of Ada 202x has been pushed back a year and a half in order to get more feedback on the proposed changes. Most of the major features went from rough outlines last fall to a completed standard with detailed wording by May. This rate of completion was just too much for most interested parties outside of the ARG to keep up with.

Rather than standardize something underbaked that might have to be changed in a few years, we're dialing back the amount of work and letting the Ada community catch up.

This comment period is not intended to introduce additional new features; such comments are always welcome but most will be deferred until the following revision. (Of course, additional features related to the ones already intended for the revision are possible.)

From: "Yannick Moy"
<moy@adacore.com>
Date: Mon, 29 Jul 2019 02:44:22 -0700

I would add that participation in the new Ada/SPARK RFC website hosted by AdaCore is very welcome for anyone who

wants to influence the future of Ada and/or SPARK:

<https://github.com/AdaCore/ada-spark-rfcs>

Participation can come in many flavors:

- signal your opinion on Pull Requests (PR) by adding a thumb-up/thumb-down on the first message of the PR
- comment on a PR to refine your opinion
- propose an RFC as a PR for others to comment

Ada-related Events

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

Ada-Europe 2019 Final Call for Participation

From: Dirk Craeynest
<dirk@cs.kuleuven.be>
Subject: Press Release - Reliable Software Technologies, Ada-Europe 2019
Date: Tue, 4 Jun 2019 22:21:31 -0000
Newsgroups: comp.lang.ada

 FINAL Call for Participation

*** UPDATED Program Summary ***

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

11-14 June 2019, Warsaw, Poland

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

**Check out tutorials and workshop! **

<http://www.ada-europe.org/conference2019/tutorials.html>

<http://www.ada-europe.org/conference2019/workshops.html>

*** Exhibition Opening & Welcome Aperitif on Tuesday ***

*** Full Program available on conference web site ***

*** Register now! ***

Press release:

24th Ada-Europe Conference on Reliable Software Technologies

International experts meet in Warsaw

Warsaw, Poland (5 June 2019) - Ada-Europe together with EDC (the Engineering Design Center, a partnership of General Electric and the Institute of Aviation), organize from 11 to 14 June 2019 the "24th International Conference on Reliable Software Technologies - Ada-Europe 2019" in Warsaw, Poland. The event is in cooperation with the Ada Resource Association (ARA), and with ACM's Special Interest Groups on Ada (SIGAda), on Embedded Systems (SIGBED) and on Programming Languages (SIGPLAN).

[...]

This year's conference offers tutorials and a workshop, two keynotes, a technical program of refereed papers and industrial presentations, an industrial exhibition and vendor presentations, and a social program.

Two tutorials are scheduled on Tuesday, targeting different audiences: "An Introduction to Ada", for those who want to understand the benefits of using Ada; and "Controlling I/O Devices with Ada, using the Remote I/O Protocol", for those willing to develop Ada programs that control external hardware devices. On Friday the conference hosts for the 6th consecutive year the workshop on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering" (DeCPS 2019): registration is complementary for conference participants.

The industrial exhibition opens Tuesday mid-afternoon in the networking area and runs until the end of Thursday afternoon. Exhibitors include AdaCore, PTC Developer Tools, Rapita Systems, Vector, and Ada-Europe. All tutorial and conference participants are invited to the exhibition opening, as well as to the Welcome Aperitif afterwards.

Two eminent keynote speakers have been invited to open each day of the core conference program: Michael Klemm (OpenMP, Germany), on "OpenMP API: A Story about Threads, Tasks and Devices"; and Tucker Taft (AdaCore, USA), on "A 2020 View of Ada".

The technical program on Wednesday and Thursday presents 9 refereed technical papers and 8 industrial presentations in sessions on Assurance Issues in Critical Systems, Tooling Aid for Verification, Best Practices for Critical Applications, Uses of Ada in Challenging Environments, Verification Challenges, and Real-Time Systems. Also included is a speaker's corner on "Experience from 40 years of teaching Ada", and vendor presentations. Peer-reviewed papers will

be published in an open-access journal, industrial presentations and tutorial abstracts in the Ada User Journal, the quarterly magazine of Ada-Europe.

The social program includes on Tuesday evening a Welcome Aperitif on the terrace of the Institute of Aviation, enjoying a wonderful view of the Warsaw airport and city center, accompanied by drinks and typical Polish snacks. On Wednesday evening will be the traditional Ada-Europe Conference Banquet, with Polish cuisine, drinks, and live piano music, in the restaurant "Przepis na kompot" in the town where Chopin was born.

The Best Paper Award will be presented during the Conference Banquet, the Best Presentation Award during the Closing session.

The full program is available on the conference web site. [...]

Latest updates:

The 12-page "Final Program" is available at <http://www.ada-europe.org/conference2019/AE-2019-Final-Program.pdf>

Check out the tutorials in the PDF program, or in the schedule at <http://www.ada-europe.org/conference2019/tutorials.html>.

[...]

A printed Conference Booklet with abstracts of all technical papers and industrial presentations will be included in every conference handout.

Help promote the conference by advertising for it:

<http://www.ada-europe.org/conference2019/promotion.html>

Put up the poster at

http://www.ada-europe.org/conference2019/picts/AE2019_poster.pdf

Recommended Twitter hashtags:
#AdaEurope and/or #AdaEurope2019.

For more info and latest updates see the conference web site at <http://www.ada-europe.org/conference2019>.

Update about Ada-Europe Conferences 2019 and 2020

From: Dirk Craeynest

<dirk@cs.kuleuven.be>

Subject: Update about Ada-Europe Conferences 2019 and 2020

Date: Sat, 29 Jun 2019 13:53:33 -0000

Newsgroups: comp.lang.ada

The 24th edition of Ada-Europe's International Conference on Reliable Software Technologies took place on 11-14 June in Warsaw, Poland, with considerable success.

The conference, graciously hosted by the Institute of Aviation, had nearly 100 participants, enjoyed a rich technical and social program, and saw much active interaction between participants, presenters, and exhibitors.

For your information, the following material is now available online:

- the "Conference Booklet" in PDF, which contains the abstracts of all presentations in the core program (see first section on [1]);
- copies of conference presentations (see "Download" links in "Conference Core Schedule" table on [1]);
- copies of DeCPS workshop presentations (see "Download links in "Program" table on [2]);
- pictures of the exhibition booths (see final part of [3]).

[1] www.ada-europe.org/conference2019/overview.html

[2] www.ada-europe.org/conference2019/workshops.html

[3] www.ada-europe.org/conference2019/sponsors.html

As announced in Warsaw, next year's conference will be held in Santander, Spain, in the week of 8-12 June 2020.

The preliminary Call for Contributions is already available on the (mini) conference web site at [4]. More details will follow later.

[4] www.ada-europe.org/conference2020/

On this occasion, the Ada-Europe Board announces a slight update of the name of its conference series:

- the complete name is "25th Ada-Europe International Conference on Reliable Software Technologies";
- the short name is "Ada-Europe Conference 2020";
- the acronym is "AEiC 2020".

Hence on social media when referring to the Ada-Europe organization we'll use #AdaEurope, and when referring to next year's Ada-Europe Conference we'll use #AEiC2020.

Ada-related Resources

Ada on Social Media

From: Alejandro R. Mosteo

<amosteo@unizar.es>

Subject: Ada on Social Media

Date: 2019/Aug/06

To: Ada User Journal readership

Ada groups on various social media:

- LinkedIn: 2_848 (+35) members [1]
- Reddit: 2_307 (+64) members [2]
- StackOverflow: 1_685 questions [3]

- Freenode: 76 (-11) users [4]
 - Gitter: 42 (=) people [5]
 - Telegram: 45 (-2) users [6]
 - Twitter: 32 (+26) tweeters [7]
 - 36 unique tweets [7]
- [1] <https://www.linkedin.com/groups/114211/>
- [2] <http://www.reddit.com/r/ada/>
- [3] <http://stackoverflow.com/questions/tagged/ada>
- [4] #Ada on irc.freenode.net
- [5] <https://gitter.im/ada-lang>
- [6] https://t.me/ada_lang
- [7] <http://bit.ly/adalang-twitter>

Repositories of Open Source Software

From: Alejandro R. Mosteo
<amosteo@unizar.es>
Subject: Repositories of Open Source software
Date: 2019/Aug/06
To: Ada User Journal readership

- GitHub: 573 (-30) developers [1]
 - Rosetta Code: 666 (+2) examples [2]
 - 36 (=) developers [3]
 - Sourceforge: 270 (=) projects [4]
 - Open Hub: 209 (=) projects [5]
 - Bitbucket: 87 (=) repositories [6]
 - Codelabs: 47 (+1) repositories [7]
 - AdaForge: 8 (=) repositories [8]
- [1] <https://github.com/search?q=language%3AAda&type=Users>
- [2] <http://rosettacode.org/wiki/Category:Ada>
- [3] http://rosettacode.org/wiki/Category:Ada_User
- [4] <https://sourceforge.net/directory/language:ada/>
- [5] <https://www.openhub.net/tags?names=ada>
- [6] <https://bitbucket.org/repo/all?name=ada&language=ada>
- [7] https://git.codelabs.ch/?a=project_index
- [8] <http://forge.ada-ru.org/adaforge>

Language Popularity Rankings

From: Alejandro R. Mosteo
<amosteo@unizar.es>
Subject: Ada in language popularity rankings
Date: Thu May 23 2019
To: Ada User Journal readership

Note: positive ranking changes means to go down in the ranking.

- TIOBE Index: 37 (+1) 0.296% (-0.03%) [1]
 - IEEE Spectrum (general): 42 (-4) [2]
 - IEEE Spectrum (embedded): 13 (=) [2]
- [1] <https://www.tiobe.com/tiobe-index/>
- [2] <https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018>

Ada-related Tools

Pure Ada libraries for Artificial Intelligence

From: Daniel
<danielnorberto@gmail.com>
Subject: Artificial Intelligence libraries in Ada
Date: Wed, 10 Jul 2019 00:25:48 -0700
Newsgroups: comp.lang.ada

Does anybody knows pure Ada libraries for AI?

Specially, I'm interested in Decision Trees, but I can't find anything on internet.

In case of a negative answer, does anybody knows a good CPU performance AI C/C++ Library working good binded to Ada code?

From: "J-P. Rosen" <rosen@adalog.fr>
Date: Wed, 10 Jul 2019 09:39:39 +0200

There is FannAda (<https://sourceforge.net/projects/lfa/>), a binding to the Fann neural network library. No idea what it's worth.

From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Wed, 10 Jul 2019 12:52:40 +0200

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

This includes decision trees both fuzzy and crisp. It is 100% Ada, except the database persistence back ends.

From: "Jeffrey R. Carter"
<spam.jrcarter.not@spam.not.acm.org>
Date: Wed, 10 Jul 2019 18:13:14 +0200

I guess you're not interested in neural networks, but there's an implementation of REM NNs in the PragmAda Reusable components.

<https://github.com/jrcarter/PragmARC>

[...] It's NNs with the REM 2nd-order learning algorithm.

http://pragmada.x10hosting.com/REM_Eq.pdf

Gnu Emacs Ada mode 6.1.1

From: Stephen Leake
<stephen_leake@stephe-leake.org>
Subject: Gnu Emacs Ada mode 6.1.1 released.
Date: Fri, 12 Jul 2019 11:10:22 -0700
Newsgroups: comp.lang.ada

Gnu Emacs Ada mode 6.1.1 is now available in GNU ELPA. This is a minor feature and bug fix release; partial file parsing is now supported for 'which-function-mode', and error correction is improved. See the NEWS files in `~/emacs.d/elpa/ada-mode-6.1.1` and `wisi-2.1.1`, or at <http://www.nongnu.org/ada-mode/>, for more details.

The process parser requires a manual compile step, after the normal list-packages installation:

```
cd ~/emacs.d/elpa/ada-mode-6.1.1
./build.sh
```

This requires AdaCore gnatcoll packages which you may not have installed; see ada-mode.info Installation for help in installing them.

dcf-ada 2.0.0 Library for Document Container Files

From: onox <denkpadje@gmail.com>
Subject: ANN: dcf-ada 2.0.0 -- A library for document container files, a Zip-based archive format
Date: Tue, 23 Jul 2019 14:15:03 -0700
Newsgroups: comp.lang.ada

An Ada 2012 library for document container files, a Zip-based archive format standardized in ISO/IEC 21320-1:2015. Document container files are Zip files with several restrictions:

- * Only "store" (uncompressed) and "deflate" compression methods are allowed
- * Archives may not be encrypted or contain digital signatures
- * Archives may not span multiple volumes or be segmented

This library is based on the Zip-Ada library, with extensive modifications:

- * Binary and Windows-specific files have been removed with The BFG Repo Cleaner
- * Reformatted code to Ada default style guide
- * Removed obsolescent features and implementation-defined extensions
- * All packages except one that uses Ada.Calendar are preelaborated
- * Removed features prohibited by ISO standard
- * Removed lots of duplicated code and simplified the API, reducing SLOC from 12k to 4.5k

Although the tools can (un)zip basic .zip files, the purpose of the library is to be able to read container files, including a future binary storage format for 3D meshes.

See the README.md at <https://github.com/onox/dcf-ada> on how to list or extract files from an archive.

Qt5Ada 5.13.0

From: leonid.dulman@gmail.com
Subject: Announce : Announce : Qt5Ada version 5.13.0 (594 packages) release 01/07/2019 free edition
Date: Sat, 3 Aug 2019 05:09:13 -0700
Newsgroups: comp.lang.ada

Qt5Ada is Ada-2012 port to Qt5 framework (based on Qt 5.13.0 open source final)

Qt5Ada version 5.13.0 open source and qt5c.dll(win64),libqt5c.so(x64) built with Microsoft Visual Studio 2017 x64 in Windows, gcc x86-64 in Linux.

Package tested with gnat gpl 2012 Ada compiler in Windows 64bit, Linux x86-64 Debian 9.4.

It supports GUI, SQL, Multimedia, Web, Network, Touch devices, Sensors, Bluetooth, Navigation and many others thinks.

My configuration script to build Qt 5.13.0 is: configure -opensource -release -nomake tests -opengl dynamic -qt-zlib -qt-libpng -qt-libjpeg -openssl-linked OPENSSL_LIBS="-lssl32 -libeay32" -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -icu -prefix "e:/Qt/5.13"

As a role Ada is used in embedded systems, but with QTADA(+VTKADA) you can build any desktop applications with powerful 2D/3D rendering and imaging (games, animations, emulations) GUI, Database connection, server/client, Internet browsing, Modbus control and many others thinks.

Qt5Ada and VTKAda for Windows, Linux (Unix) is available from <https://r3fowwcolhrzycn2yzlzzw-on.driv.tw/AdaStudio/>

The full list of released classes is in "Qt5 classes to Qt5Ada packages relation table.docx"

VTKAda version 8.2.0 is based on VTK 8.2.0 (OpenGL2) is fully compatible with Qt5Ada 5.13.0.

Qt5AVAda

From: leonid.dulman@gmail.com
Subject: Announce : QtAVAda version 1.12.0 release 01/08/2019 free edition
Date: Sat, 3 Aug 2019 05:09:13 -0700
Newsgroups: comp.lang.ada

Qt5AVAda is ada-2012 port to QtAV multimedia playback framework based on Qt + FFmpeg. Cross platform. High performance. Easy to use and base on QtAV 1.12 developed by wang-bin <https://github.com/wang-bin/QtAV>.

QtAVAda build widgets inside Qt5Ada application(5.13.1 release 01/08/2019).

QtAVAda for Windows, Linux (Unix) is available from <https://r3fowwcolhrzycn2yzlzzw-on.driv.tw/AdaStudio>

If you have any problems or questions, tell me know.

String edit v3.5

From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Subject: ANN: String edit v3.5 released
Date: Sun, 4 Aug 2019 16:56:40 +0200
Newsgroups: comp.lang.ada

The library provides various means for editing and formatting strings:

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

This release adds implementations of some standards actively used in communication RFC 3061, 4514; ISO 8601.

Changes to the previous version:

- Added the package Strings_Edit.Long_Floats, an instance of String_Edit.Floats with Long_Float;
- The package Strings_Edit.UTF8.ITU_T61 provides ITU T.61 encoding conversions;
- The package Strings_Edit.Object_Identifier provides implementation of RFC 3061 object identifiers (OID);
- The package Strings_Edit.Distinguished_Names provides implementation of RFC 4514 distinguished names (DN);
- The package Strings_Edit.ISO_8601 provides ISO 8601 representations of time and duration;
- Encoding and decoding Base64 streams were added to the package Strings_Edit.Base64.

Simple Components for Ada v4.41

From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Subject: ANN: Simple components for Ada v4.41 released
Date: Mon, 5 Aug 2019 13:57:16 +0200
Newsgroups: comp.lang.ada

The new release is focused on ASN.1 support. The implementation does not require ASN.1 compiler. It is based on reflection of Ada attributes. The objects corresponding to ASN.1 objects are put together into record types and the encoding is deduced from the placement. The implementation provides arena pool to allocate data associated with ASN.1 objects. This allows to handle very large and indefinite ASN.1 objects without allocating maximum possible memory in advance. This also enables sharing memory between ASN.1 CHOICE alternatives as well as recursively defined ASN.1 objects. Implementations of LDAP and X.509 certificates based on ASN.1 are provided.

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

Changes to the previous version:

- The package OpenSSL was extended;
- Added implementation of ASN.1 encoding;
- X.509 ASN.1 certificates implementation added;
- LDAP implementation added.

From: Shark8
<onewingedshark@gmail.com>
Date: Mon, 5 Aug 2019 07:22:43 -0700

Wow!

This is incredible news, especially for things like the Wasabee browser project.

There was someone who was working on an Ada/SPARK ASN.1 compiler (Peter Chapin?) and I think the people doing this project -- <https://github.com/ttsiodras/asn1sec> -- which *is* an ASN.1 compiler.

WRT the OpenSSL dependency, how much work would it be to get rid of it?

From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Mon, 5 Aug 2019 17:56:43 +0200

> Wow!

> This is incredible news, especially for things like the Wasabee browser project. There was someone who was working on an Ada/SPARK ASN.1 compiler (Peter Chapin?) and I think the people doing this project [...]

I am aware of ASN1SCC, but I wanted an alternative approach that does not require code generator and can handle constraints dynamically.

ASN.1 specifications are infested with objects defined up to "MAX" items. E.g. the LDAP filter is a variable record (CHOICE) with disjunctive and conjunctive forms as alternatives containing the LDAP filter recursively as terms. The number of terms is an unspecified MAX and the depth of recursion is kind of infinite. I have no idea how the generators handle this mess. If compiled literally, e.g. with MAX=256 depth=32, it would take a huge amount of memory while in reality it is bounded from above just by the message length.

> WRT the OpenSSL dependency, how much work would it be to get rid of it?

There is no dependency on OpenSSL.

OpenSSL and GNUTLS are two back-ends used in the corresponding implementations of the secure connection handler. Both are separate gpr-projects.

All network stacks are designed to work with any handler implementation. Should Ada TLS become available I would use it in yet another implementation of.

Ada-related Products

Embedded Boards for Ada

From: Ricardo Brandão

<rbrandao.br@gmail.com>

Subject: Which embedded devices do you use?

Date: Tue, 4 Jun 2019 08:01:50 -0700

Newsgroups: comp.lang.ada

I worked with embedded systems for a long time.

I started with Z-World devices on late 80's. And now I'm working mainly with ESP32 boards.

I'm learning Ada and I'd like to use it on my new projects. So, I'd like to know what boards/processors you guys are using.

Normally, my projects need Digital IOs, Analog Inputs, and any way to wireless communication: Bluetooth, BLE, WiFi...

And I'm used to work with I2C devices as well (OLED displays, sensors, RTC, and so on).

From: "Dmitry A. Kazakov"

<mailbox@dmitry-kazakov.de>

Date: Tue, 4 Jun 2019 17:14:33 +0200

On 2019-06-04 17:01, Ricardo Brandão wrote:

> So, I'd like to know what boards/processors you guys are using.

ARM-based boards with a Linux on it.

> Normally, my projects need Digital IOs, Analog Inputs, and any way to wireless communication: Bluetooth, BLE, WiFi...

For quality analogue I/O we are using EitherCAT or ModBus terminals. For digital I/O on board GPIO could serve but usually it is terminals as well. CAN and Serial is used too.

From: "Dmitry A. Kazakov"

<mailbox@dmitry-kazakov.de>

Date: Tue, 4 Jun 2019 17:56:39 +0200

On 2019-06-04 17:26, Ricardo Brandão wrote:

> So, it could be a good idea use Beaglebone as a start point?

Yes. We are using BB a lot, for prototyping etc.

From: Optikos <optikos@verizon.net>

Date: Tue, 4 Jun 2019 08:55:21 -0700

I like Marvell's ESPRESSObin board, as distributed in the USA by Globalscale Technologies (shipped direct from PRChina).

<http://ESPRESSObin.net>

With an Armada 3720 SOC, it is capable of doing some serious telecom/datacom high-speed packet processing with some hardware assist (instead of slow software-processor speed) on its 2 LAN and 1

WAN Ethernet ports. (Of course better would be the 7000 or 8000 series Armadas which have full-fledged SR-IOV on their SOC, but hey there is always room for improvement in the future.)

There is also the ESPRESSObin's baby brother (with fewer Ethernet ports): the new Sheeva64 in wall-wart form-factor, continuing the venerable SheevaPlug family.

<https://www.GlobalscaleTechnologies.com/p-86-sheeva64.aspx>

What is nice about the ESPRESSObin and Sheeva is that they embrace Yocto-Project Linux, so you are not tied to any one Linux distro. Instead, Yocto Project requires that you roll your own Linux distro from near-scratch (e.g., mimicking whichever distro or bleeding edge referent* that you prefer).

* e.g., Linus Torvalds' git repository

<https://www.YoctoProject.org>

Each ARM hobbyist SBC community has a different specialty. I wouldn't do high-packet-rate telecom/datacom processing on a Raspberry Pi, for example. That is what the Marvell Armada line is better suited for.

Btw, Marvell's Armada series is the descendent whose ancestors include the DEC StrongARM and the Intel XScale, so in some ways this is one of the "main trunks" in the ARM-processor community, especially for industrial usage—not some twig on a branch.

https://www.TheRegister.co.uk/2006/06/27/intel_sells_xscale

Plus, Marvell's MoChi (modular chip multi-die SOCs) technology (•not• in the Armada 3720) is one of the industry leaders in DARPA's MoChi endeavors in recent years. DARPA is trying to seed some of the major SOC processor manufacturers with MoChi. Getting on board with Marvell now likely prepares you for the aggressive MoChi future as the 1st-generation-MoChi 7000 and 8000 series eventually migrates into the hobbyist SBCs, and then aggressive-MoChi successors follow after that in coming years.

<https://www.marvell.com/architecture/mochi>

From: Olivier Henley

<olivier.henley@gmail.com>

Date: Tue, 4 Jun 2019 11:51:10 -0700

You can dig here:

- <https://github.com/ohenley/awesome-ada#Runtimes> (the bb-runtimes repo by AdaCore)

- <https://github.com/ohenley/awesome-ada/blob/master/README.md#Hardware-and-Embedded> (The main repo to check is ada-drivers-library. Adacore is behind and they are of great assistance.)

- <https://github.com/ohenley/awesome-ada/blob/master/README.md#Books>
Do not forget to check the book about embedded by Maciej Sobczak.

Hope it helps and any PR/Suggestions to refactor the list is welcome.

From: Niklas Holsti

<niklas.holsti@tidorum.invalid>

Date: Tue, 4 Jun 2019 22:14:12 +0300

The AdaCore "Make with Ada" competition entries use a wide range of hardware. See <https://www.hackster.io/contests/adacore/submissions#challengeNav>.

(As for myself, I've recently used Ada for embedded systems only in space applications, so only on made-for-space computers, usually with SPARC processors and a high price tag.)

From: Philip Munts

<philip.munts@gmail.com>

Date: Wed, 5 Jun 2019 01:33:09 -0700

BeagleBone (more and better I/O) and Raspberry Pi (faster). Both running my own embedded Linux distribution:

<https://github.com/pmunts/muntsos>

Debian and Raspbian are fine general purpose operating systems, but IMHO they are wretched for embedded systems.

Anything on mains power should be running Linux. The networking capabilities and development tools are just so far beyond microcontrollers.

I'm especially fond of the PocketBeagle and the Raspberry Pi Zero Wireless. Running Ada programs, of course.

Janus/Ada 3.2.1

From: "Randy Brukardt"

<randy@rrsoftware.com>

Subject: Janus/Ada 3.2.1 Released!

Date: Wed, 26 Jun 2019 00:12:16 -0500

Newsgroups: comp.lang.ada

A new version of Janus/Ada has finally made it to release. This version includes recognition of the full Ada 2012 syntax, null exclusions, private with, a number of language-defined libraries from both Ada 2005 and 2012, and code quality warnings to detect likely bugs early.

Read the full announcement at

<http://www.rrsoftware.com/html/blog/ja-321a-rel.html>.

Existing customers with a current support agreement (including those in their first 90 days of ownership) can download the new version and use their existing key to unlock it. For everyone else, see our website for pricing: <http://www.rrsoftware.com/html/companyinf/prices.htm>.

Randy Brukardt.

P.S. I apologize to anyone that would rather not see the blatant ad. I try not to

do this more often than once per year, and the information ought to be relevant to those who sometimes forget that there are other, actively developed Ada compilers out there.

From: "Jeffrey R. Carter"
 <spam.jrcarter.not@spam.not.acm.org>
Date: Wed, 26 Jun 2019 08:53:02 +0200

Good news. I see that the website still refers to the compiler as Janus/Ada 95. How much additional work is needed before you have a full Ada-12 compiler?

From: "Randy Brukardt"
 <randy@rrsoftware.com>
Date: Wed, 26 Jun 2019 17:40:42 -0500

Probably more years than I have left on the planet. While I've mapped out a design for most new features, a few things have been pretty much ignored (esp. interfaces and real-time stuff).

If I was able to find a business plan that made sense, it could get done faster, but as it stands I don't expect to ever break even with it and as such one can't really spend \$\$\$ (as opposed to time) on it.

From: Optikos <optikos@verizon.net>
Date: Wed, 26 Jun 2019 08:41:54 -0700

On Wednesday, June 26, 2019 at 3:52:51 AM UTC-5, Dmitry A. Kazakov wrote:

[...]

> P.S. I hope Janus will target Linux someday. It could be a Windows-hosted cross. I think many would buy that thing.

I concur, but the highest-RoI would be for Janus/Ada to have the LLVM backend in one fell swoop. Then we as users would naturally get various object-file formats (e.g., ELF, XCOFF) and ISAs (e.g., Apple ARM) and debug formats (e.g., gdb's, lldb's)—both native and cross-compiled—inherited as a by-product, killing multiple birds with one stone.

Randy, would putting Janus/Ada's front end on

0) LLVM backend

be more difficult than any major target feature listed above alone (e.g.:

1) Janus/Ada as-is without LLVM plus ELF on x86;

2) Janus/Ada as-is without LLVM plus PE-on-ARM for the forthcoming ARM-based bendable/foldable Surface Phone thingy-whatever-it-will-be-called, deriving from Andromeda & Courier prototypes with Composable-Shell and Windows Core OS)?

From: "Randy Brukardt"
 <randy@rrsoftware.com>
Date: Wed, 26 Jun 2019 17:36:27 -0500

[Replying to the numbered list in the previous post.]

These are almost completely orthogonal: the existing code generator would work for Linux, and the (old) Unix JLink did

ELF. The issue with Linux is updating the runtime to use Linux system calls (these are different than the ones from the old Unix).

OTOH, attaching LLVM is a totally different level of work, and I don't know enough about LLVM to say how easy or hard it would be. OTOH, we did something similar of Unisys, so we already have most of the ability available.

But again, note that a code generator is a small (and usually easiest) part of porting to a new target. Making a usable runtime (that is, exception handling, finalization, overflow checking, divide-by-zero traps, basic I/O, and most of all, tasking) is generally a bigger job.

AdaControl 1.21r3

From: "J-P. Rosen" <rosen@adalog.fr>
Subject: [Ann] AdaControl version 1.21r3 released

Date: Thu, 11 Jul 2019 14:42:04 +0200
Newsgroups: comp.lang.ada

Adalog is pleased to announce version 1.21r3 of AdaControl. There are now 71 rules, 579 subrules.

This version includes new checks to ease the transition to Ada 2012 (like for-in loops that can be changed to for-of loops), improvements to the auto-fixing features, extensions to existing rules (like use-package that can be changed to use use-type or use-all-type), bug fixes... See file HISTORY for the complete list of improvements.

The pre-compiled version uses now GNAT Community 2019.

Available from <http://www.adacontrol.fr>

Enjoy!

Ada and Operating Systems

GNAT CE 2019 and Impending Changes on MacOS

From: Simon Wright
 <simon@pushface.org>
Subject: Re: GNAT CE 2019 macOS
Date: Thu, 30 May 2019 20:34:44 +0100
Newsgroups: comp.lang.ada

[The following post discusses an issue with missing system libraries during linking in MacOS, due to changes in the operating system SDK.]

Bill Findlay
 <findlaybill@blueyonder.co.uk> writes:

```
>> gnatlink
    /Users/wf/mekhos/MacOSX/e.ali -
    funwind-tables -fdata-sections -
    ffunction-sections -mtune=native -fno-
```

```
stack-check -fomit-frame-pointer -flt0 -
O3
```

```
> ./quad_div.o -Wl,-dead_strip -Wl,-
dead_strip -flt0
```

```
>
```

```
>> ld: library not found for -lSystem
```

```
>> collect2: error: ld returned 1 exit status
```

```
>> gnatmake: *** link failed.
```

```
>
```

```
> -lSystem ??
```

I've had a discussion about this with AdaCore.

The problem they are addressing is that Apple are moving towards having system includes only in the SDKs rather than in /usr/include; see [1], which says "As a workaround, an extra package is provided which will install the headers to the base system. In a future release, this package will no longer be provided".

"this package" is the one I reference at [2].

AdaCore's approach is to build the compiler with a "system root" that references the SDK in situ; the actual link takes place with

```
/usr/bin/ld -syslibroot
```

```
/Library/Developer/CommandLineTools/
Platforms/MacOSX.platform/Developer/S
DKs/MacOSX.sdk/
```

and, unfortunately for us, that's the full Xcode and not the CommandLineTools subset; so if you only have the CommandLineTools, ld looks for libSystem.dylib in a non-existent directory.

One approach is to build with

```
-largz -Wl,-syslibroot/
```

Another one is to install the full Xcode.

I guess Xcode is the way to go.

For the future

I don't think it's possible to have multiple syslibroots.

I don't think the GCC developers would be happy with building knowledge of xcode-select into the compiler, so it could make the same runtime choices as Apple tools.

Since the SDKs really only impact the includes, at any rate as long as you're on macOS and not iOS, I'm wondering whether it'd be possible to add both SDK include paths to GCC's include paths and avoid the syslibroot impact on libraries.

Nothing yet about this on the GCC mailing lists, that I can see.

[1] https://developer.apple.com/documentation/xcode_release_notes/xcode_10_release_notes#3035624

[2] <https://forward-in-code.blogspot.com/2018/11/mojave-vs-gcc.html>

From: Simon Wright
<simon@pushface.org>
Date: Tue, 18 Jun 2019 18:00:52 +0100
 I did something on this, written up here:
<https://forward-in-code.blogspot.com/2019/06/mac-os-software-development-kit-changes.html>

Ada in Genode OS

From: Kay-Uwe Genz <kug1977@web.de>
Subject: Genode OS Framework 19.05 goes SPARK
Date: Mon, 17 Jun 2019 03:43:40 -0700
Newsgroups: comp.lang.ada

you might be interested to see, that Genode OS Framework 19.05 is integrating Ada/SPARK runtime and SPARK-based cryptography

Spunky: A kernel using Ada - Part 1: RPC
 For me these news were new.
<https://www.osnews.com/story/130141/ada-spark-on-genode/>

From: Kay-Uwe Genz <kug1977@web.de>
Date: Wed, 19 Jun 2019 07:30:34 -0700

> I don't understand why they put c++ on one end and SPARK on the other...
 Don't they know "normal" Ada includes quite enough "non-static" features? Or is that compatibility with existing libraries the problem? Not really said in that article.

Most of the L4 development which is where Genode OS Framework came from is done in C++ and Ada/SPARK is more a hobbyist project, I guess. The Muen kernel is focussed 100% on Ada/SPARK.

Ada and other Languages

Specification/Body Separation in Ada

From: John Perry <john.perry@usm.edu>
Subject: Why .ads as well as .adb?
Date: Sat, 1 Jun 2019 17:48:16 -0700
Newsgroups: comp.lang.ada

I understand that Ada, like Modula-2 and Modula-3, and arguably like C++, requires a definition file (.ads) as well as an implementation file (.adb). With Oberon, Wirth moved away from definition files, using a symbol to indicate which module identifiers should be exported. (Someone else may have done this before him; it's just that I'm most familiar with this history.) Most languages I'm familiar with these days do something similar, either via public/private or some other mechanism.

As far as I can tell, though, Ada has stuck with the two separate files, rather than, say, generating an .ads from an .adb with export markup.

Is there a reason Ada hasn't moved to this simpler structure?

From: "J-P. Rosen" <rosen@adalog.fr>
Date: Sun, 2 Jun 2019 07:42:58 +0200

[...]

One of the main (huge) benefits of Ada is in being able to use specifications even before the body exists. You can:

- 1) write the specification, compile it to make sure that it make sense
 - 2) write the code that uses the specification, to make sure that the specification meets the needs of the using code
 - 3) write the body, with the assurance that what you do is the right thing.
- You can even add:
- 2.5) write a prototype body to check that the behaviour is correct, before writing the full body that meets all requirements.

[...]

From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Sun, 2 Jun 2019 08:39:23 +0200

It is general design principle of separation specifications from implementations. [...] [It] has evident advantages for code base maintenance, team development, testing, separate compilation etc. BTW, you can stuff bodies and specifications in the same file. It is purely compiler's business. See gnatshop for GNAT. [...]

On 2019-06-02 02:48, John Perry wrote:

> As far as I can tell, though, Ada has stuck with the two separate files, rather than, say, generating an .ads from an .adb with export markup.

That is not possible. You cannot generate specification from implementation and conversely. In both cases there is additional information missing. It could be two different languages. Even in the languages that confuse these things, declarations have syntax different from definitions. [...]

From: Maciej Sobczak
<see.my.homepage@gmail.com>
Date: Tue, 4 Jun 2019 01:03:26 -0700

[Written by J-P. Rosen
 <rosen@adalog.fr>]

> If you have a body with many subprograms, how can you tell which ones are intended to be exported, and which ones are private to the body?

By annotating them appropriately? Keywords "private" or "export" or similar are commonly used for this purpose.

Please note that your question could also refer to the concept of DLLs, which is not directly addressed by Ada (nor C++). Yet, somehow we do manage to solve this problem.

[...]

From: Keith Thompson <kst-u@mib.org>
Date: Mon, 03 Jun 2019 12:51:14 -0700

"Dmitry A. Kazakov" <mailbox@dmitry-kazakov.de> writes:

> No. Specification describes a class of implementations. You cannot deduce class from its single member.

I suspect the point is that you *could* have an Ada-like language in which specifications could be unambiguously generated from implementations. You'd need some kind of additional annotation to specify whether a given declaration is to be exported.

You can't in Ada as it is, because Ada isn't designed that way.

[Editor's note: the following subthread discusses the readability concerns of separating specifications, but also that clarity and separation may be achieved not only via specifications.]

From: Brad Moore
<bmoore.ada@gmail.com>
Date: Fri, 7 Jun 2019 07:10:11 -0700

On Friday, June 7, 2019 at 1:59:26 AM UTC-6, Maciej Sobczak wrote:

> 1. There *are* languages that don't use separate spec files. Java and Python are well known examples, representing both compiled and scripted approaches.

[...] I think it is a big mistake of languages that encourage the specification and implementation to be in the same source file, and very surprised to see that anyone would be arguing for that.

The separation of specification and implementation ties into the "separation of concerns" attributed to Dijkstra way back in 1974.

When wanting to make use of a 3rd party package in Ada, I value being able to generally understand how to use that package by looking at the specification without having to look at the implementation. You generally only need to look at the public part of a package specification, as you can rely on anything past that as being implementation details.

Even with C++, one cannot stop reading when you see a private: keyword in a class definition, because there can be many public and private sections in a class. You have to keep reading the class specification until to hit the end of the class specification, in case you missed more public parts.

[...]

> 2. Programs written in those languages do *not* need to be written in one giant file. Actually, Java is frequently criticized (it was even in this thread) for forcing the programmer to use too many (!) files. Even though it does not have separate specs.

Maybe Ada offers a benefit here. In languages like Java, there is a tendency to

want to put each class in a separate file. With Ada packages, it can make more sense to organize related types in the same package.

[Editor's note: another subthread explores the implications for "Programming in the large"]

From: "Randy Brukardt"
<randy@rsoftware.com>

Date: Mon, 10 Jun 2019 17:07:38 -0500

[...]

"Maciej Sobczak"

<see.my.homepage@gmail.com> wrote
[...]

> What I don't accept is the religious attitude that Ada is the only language that got the software engineering right and (consequently) that everything else is broken.

The truth hurts. So far as I can tell, no other language has really tried to "get software engineering right". It's possible, of course, but everyone either is trying to graft engineering onto some preexisting base without it (C++, Java) or is building something that's more about fast construction than engineering (Python).

[...]

From: Optikos <optikos@verizon.net>
Date: Mon, 10 Jun 2019 17:32:36 -0700

There was only one other programming language that tried to "get software engineering right" and that achieved significant industrial usage and an open-source GCC compiler and that was ISO standardized: CHILL. While DoD & NATO were busy with their HOLWG effort for the military, ITU-T (in the United Nations) launched a somewhat competing effort for telecom in the EU (and AT&T steadfastly rejected both for the most part except for some monitoring of the 2 other efforts, so that AT&T pushed forward with C).

As can be seen in the following example CHILL source code, if Ada was envisioned as a Pascal/Wirth-esque-family language, CHILL was envisioned as a PL/I-esque-family language. As such, Ada is beautiful & refined by comparison, whereas CHILL is rather abrupt & uncouth, as if it is most at home on an IBM mainframe with its fellow brethren CICS and JCL and of course PL/I. CHILL and Ada share many of the same goals and as such have some analogous language features that are absent in most other programming languages. Except for some maintenance of CHILL-based telecom equipment from Alcatel and Siemens, CHILL has become a dead language.

<http://psc.informatik.uni-jena.de/languages/chill/chill.htm>

[...]

From: "Jeffrey R. Carter"
<spam.jrcarter.not@spam.not.acm.org>
Date: Tue, 11 Jun 2019 17:49:24 +0200

On 6/11/19 12:07 AM, Randy Brukardt wrote:

> So far as I can tell, no other language has really tried to "get software engineering right".

Precisely. It's important to remember that separation of spec and body have been part of Ada from the beginning, and Ada was designed to support the way S/W engineers think and work from the beginning. Like many Ada features, S/W engineers understand and like separation of spec and body, and coders don't. For me, much of this thread can be viewed simply as people saying "I'm a S/W engineer" or "I'm a coder".

From: John Perry <john.perry@usm.edu>
Date: Mon, 3 Jun 2019 06:37:43 -0700

Thanks to everyone for the replies. Personally, I find three of them especially compelling:

"As a teacher, I keep fighting with students who jump to writing bodies too early."

[I know exactly what this is like.]

"teams can work separated from each other as needed, without the project having to distribute all of the implementation to everyone"

[Having separate specification files against which one can *compile* would be useful, not just convenient, though I think it's arguable that one can do this in Oberon, too, via .smb files and documentation.]

"convenience"

[not a direct quote, but several people point to this, and until I read their explanations I thought the convenience ran in the other direction]

[...]

[Editor's note: the author proposes to follow-up with the impossibility of generating unambiguous specifications from bodies. If the conversation catches up, this will be reported in the next issue.]

Issues with Fortran Calling Convention

From: Chris M Moore
<zmower@ntlworld.com>
Subject: Making the same mistake as the broken C interface to fortran
Date: Tue, 25 Jun 2019 00:33:39 +0100
Newsgroups: comp.lang.ada

Read this interesting article today:

<https://lwn.net/SubscriberLink/791393/41d57555202e8cdb/>

Synopsis: C interfaces to Fortran makes some assumptions about how to call fortran ABIs (I don't need to pass the

hidden length parameter if it is a character*1) but now Gfortran has optimisations which assume a different calling convention (Thou shalt pass the hidden length).

There are work around (compile fortran with -fno-optimize-sibling-calls) but it seems that the proper fix is to pass the hidden length parameter.

I had a quick look at the LAPACK bindings and they both seem to use Ada characters. :/

[Editor's note: after some back and forth discussion, it seems Ada may be affected by the same issue. What follows is the last post in the thread with an Ada reproducer.]

From: Chris M Moore
<zmower@ntlworld.com>
Date: Sun, 7 Jul 2019 17:33:46 +0100

I spoke too soon when I said

> I'm sure GNAT does the right thing if you're using Fortran_Character.

If I change callee.f to

```
subroutine callee (c)
  character (len=*), intent (in) :: c
  print *, 'parameter c is ', c
end
```

then STORAGE_ERROR is the order of the day no matter the call used. Looking at the assembler, this is because GNAT does not pass the length of the string.

I compared it to fcall.f:

```
program fcall
  call callee("OK")
  call callee("Oh noes")
  stop
end
```

and this unsurprisingly does pass the lengths.

I've used the webform on the Community section of the GNAT website to provide feedback. I've pointed out that the issue also affects single character parameters.

Ada Practice

References vs. Access Types

From: "Alejandro R. Mosteo"
<alejandror@mosteo.com>
Subject: References vs access types
Date: Fri, 31 May 2019 17:44:34 +0200
Newsgroups: comp.lang.ada

So, part of the point of reference types is to be able to return an item "by reference" without being able to store the pointer:

```
type Item;
type Item_Access is access Item;
```

```
type Reference (Ptr : access Item) is
  limited null record;
```

```
function Get (...) return Reference; -- (1)
```

In Gem #107 this is said as advantageous against, for example,

```
function Get (...) return Item_Access;
-- (2)
```

because "access discriminants are unchangeable. The discriminant also cannot be copied to a variable [like Item_Access]" [1].

Now, without thinking much about it, while fighting old bugs, I have sometimes replaced a problematic Reference with

```
function Get (...) return access Item;
-- (3)
```

And here comes the question: besides losing the ability to use aspects on the Reference type, or using it for some fancy refcounting, does (3) give the same safeties wrt to copying as (1)? Are there any other hidden traps in (3) (assuming the pointee thread-safety/lifetime is properly managed)?

Or, put it another way, is (1) always preferable? Or may (3) suffice for simple uses?

[1] <https://www.adacore.com/gems/gem-107-preventing-deallocation-for-reference-counted-types/>

From: "Dmitry A. Kazakov"
<mailbox@dmitry-kazakov.de>
Date: Fri, 31 May 2019 18:55:53 +0200

My preferences list would be:

#1 - Never, visually ugly, semantically questionable, lacking transparent access to the target object and technically not a reference at all, plus unstable with GNAT compilers.

#2 - Construction of new stand-alone objects (frequently class-wide), implementation-dependent stuff.

#3 - Access to a component of an existing object.

As for hidden traps, only #3 is safe upon inheritance, if primitive operation and thus covariant.

From: AdaMagica
<christ-usch.grein@t-online.de>
Date: Fri, 31 May 2019 16:55:01 -0700

I'm quite opposed to Dmitry ['s statement about #1].

I admit that #1 is clumsy. But see Gem 123 to learn how this syntax may be improved with some aspects.

(Compiler problems are never an argument to avoid some feature forever.)

From: "Randy Brukardt"
<randy@rsoftware.com>
Date: Fri, 31 May 2019 16:33:18 -0500

(1) and (3) have the same accessibility rules, so they have the same safety from copying (no more and no less). However, since (3) is returning an access type, one can directly assign the function result into an access type, and that will work as the function will then have the accessibility of

the access value. (But of course, you might get an accessibility failure inside the function in that case.)

An important part of the reference mechanism is the use of aliased parameters. For a function, those are required to have the same accessibility as the function result. This makes most problematic calls illegal. For instance, in:

```
function Get (Obj : aliased in out
             Some_Type) return access
             Some_Other_Type;
```

Ptr : Some_Access_Type;

```
procedure Whatever is
  Local : Some_Type;
begin
  Ptr := Get (Local); -- illegal.
  Get (Local).all := ...;
end Whatever;
```

The first call to Get here is illegal as the actual parameter is more nested than the level of the function call (which is that of Ptr). This prevents Get from keeping a pointer longer than the object exists. The second call to Get is legal because the level of that call is local, and therefore the object lives long enough.

Create and Append_File

From: "Jeffrey R. Carter"
<spam.jrcarter.not@spam.not.acm.org>
Subject: Create and Append_File
Date: Thu, 6 Jun 2019 22:45:09 +0200
Newsgroups: comp.lang.ada

You can call Create with mode Append_File. I'm trying to figure out what that's supposed to do (as opposed to what compilers do). I've read ARM A.7, A.8.2, and A.10.2, and am still not sure.

It seems there are 2 likely interpretations:

1. Create creates a file, so this is the same as using mode Out_File
2. Since mode Append_File was given, it means to open the file in append mode if it exists, or create it as for mode Out_File if it doesn't

If 1., then why allow Append_File for Create? A subtype excluding it could be defined for Create.

Of course, you can also Create a file with mode In_File, which I presume means to create an empty file and open it for reading, which doesn't seem very useful, so maybe I shouldn't expect these to make sense.

From: "Randy Brukardt"
<randy@rsoftware.com>
Date: Thu, 6 Jun 2019 16:24:52 -0500

I believe it means the same as Out_File. Other requirements in RM (not very clear ones, I'm afraid) require the file opened by Create to be empty, whether or not the file previously existed. So, if Create allows (re)creating an existing file (it

doesn't have to, it could raise Use_Error), that file will be empty. In that case, Out_File and Append_File are the same.

As you note, Create (In_File) is already nonsense, so Create (Append_File) might as well be nonsense as well (it's *less* nonsense in any case, since a modeless Reset preserves the mode, and the file wouldn't necessarily be empty at that point).

From: "Jeffrey R. Carter"
<spam.jrcarter.not@spam.not.acm.org>
Date: Fri, 7 Jun 2019 17:59:39 +0200

On 6/7/19 10:01 AM, Simon Wright wrote:

> I guess I should add this to my StackOverflow answer which may have been the trigger for this question. I have No Idea why I thought it sensible to Create the file in Append_File mode.

Yes, I saw Create with Append_File and wondered what that should do. It seemed reasonable that it would open the file in append mode if it existed, and create it in output mode otherwise, but that's not what GNAT does, so here we are.

Conventions Applied to Entity Views

From: Jere <jhb.chat@gmail.com>
Subject: Convention Question related to access types
Date: Thu, 6 Jun 2019 18:51:29 -0700
Newsgroups: comp.lang.ada

The RM in section B.1 talks about Ada Standard requirements for convention compatibility. In it however it doesn't mention anything about private types, full views, etc.

Say you are wanting to bind to an opaque type in C:

```
package Bindings is
  type Opaque_Type(<>) is limited
    private;
  type Binding is access Opaque_Type
    with Convention => C;
```

```
procedure Some_Procedure(
  Value : Binding) with Import,
  Convention => C;
```

private

```
type Opaque_Base is limited null
  record with Convention => C;
type Opaque_Type is new
  Opaque_Base;
```

```
end Bindings;
```

GNAT happily accepts that, but I am unsure if that is because of the "The implementation permits T as an L-compatible type." part or because Opaque_Base is a proper convention compatible type and Opaque_Type derives from it and is thus convention

compatible as well, even though it is a private type.

I couldn't find anything dictating whether the convention compatibility rules applied to the full view or the public view.

From: "Randy Brukardt"
<randy@rsoftware.com>
Date: Sat, 8 Jun 2019 00:11:08 -0500

Conventions apply to **entities**. See 6.3.1(2/1): "a convention can be specified for an entity". Views like a partial view is **of** an entity, not an entity itself. Thus there is only a single convention for a type. Where it is specified doesn't matter

outside of Legality Rules. Thus the rules in B.1 only need to talk about types, not views.

I just had this argument about "entity" with other ARG members vis-a-vis a different topic (I lost :-), so I'm very certain this is correct.

Revolutionize your software verification

+ *Efficiency, Automation, Reliability* +



 **RAPITA** Systems
A DANLAW Company

Unit testing · System testing · Coverage analysis · Timing analysis
V&V services · Multicore timing services · DO-178C training
Ada · C · C++

www.rapitasystems.com

Conference Calendar

Dirk Craeynest

KU Leuven, Belgium. Email: Dirk.Craeynest@cs.kuleuven.be

This is a list of European and large, worldwide events that may be of interest to the Ada community. Further information on items marked ♦ is available in the Forthcoming Events section of the Journal. Items in larger font denote events with specific Ada focus. Items marked with © denote events with close relation to Ada.

The information in this section is extracted from the on-line *Conferences and events for the international Ada community* at <http://www.cs.kuleuven.be/~dirk/ada-belgium/events/list.html> on the Ada-Belgium Web site. These pages contain full announcements, calls for papers, calls for participation, programs, URLs, etc. and are updated regularly.

2019

- October 01-04 **38th IEEE International Symposium on Reliable Distributed Systems (SRDS'2019)**, Lyon, France. Topics include: distributed systems design, development and evaluation, with emphasis on reliability, availability, safety, dependability, security, and real-time.
- October 07-11 **23rd International Symposium on Formal Methods (FM'2019)**, Porto, Portugal (aka 3rd World Congress on Formal Methods). Topics include: formal methods in a wide range of domains including software, computer-based systems, systems-of-systems, cyber-physical systems, human-computer interaction, manufacturing, sustainability, energy, transport, smart cities, and healthcare; formal methods in practice (industrial applications of formal methods, experience with formal methods in industry, tool usage reports, ...); tools for formal methods (advances in automated verification, model checking, and testing with formal methods, tools integration, environments for formal methods, ...); formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, ...); etc.
- October 07 **8th Formal Methods for Interactive Systems Workshop (FMIS'2019)**. Topics include: general design and verification methodologies which take account of human behaviour; application areas include pervasive and ubiquitous systems, cyber-physical systems, augmented reality, scalability and resilience, mobile devices, embedded systems, safety-critical systems, high-reliability systems, shared control systems, digital libraries, eGovernment, human-robot interaction.
- October 07 **5th Formal Integrated Development Environment Workshop (F-IDE'2019)**. Topics include: F-IDE building (design and integration of languages, development of user-friendly front-ends); how to make high-level logical and programming concepts palatable to industrial developers; integration of Object-Oriented and modularity features; integration of static analyzers; integration of automatic proof tools, theorem provers and testing tools; documentation tools; impact of tools on certification; experience reports on developing F-IDEs; experience reports on using F-IDEs; experience reports on formal methods-based assessments in industrial applications; etc.
- October 11 **Industry Day (i-Day'2019)**. Topics include: industrial applications of formal methods, experience with introducing formal methods in industry, tool usage reports and experiments with challenge problems; how the use of formal methods has overcome engineering and certification/qualification problems, led to improvements in design or provided new insights, with safety and/or security consideration in mind; etc.
- October 08-11 **19th International Conference on Runtime Verification (RV'2019)**, Porto, Portugal. Topics include: monitoring and analysis of the runtime behaviour of software and hardware systems. Application areas include cyber-physical systems, safety/mission critical systems, enterprise and systems software, cloud systems, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy.
- October 13-18 **Embedded Systems Week 2019 (ESWEEK'2019)**, New York City, USA. Topics include: all aspects of embedded systems and software.

- October 13-18 **International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES'2019)**. Topics include: latest advances in compilers and architectures for high-performance, low-power embedded systems; compilers for embedded systems: multi- and many-core processors, GPU architectures, reconfigurable computing including FPGAs and CGRAs; security, reliability, and predictability: secure architectures, hardware security, and compilation for software security; architecture and compiler techniques for reliability and aging; modeling, design, analysis, and optimization for timing and predictability; validation, verification, testing & debugging of embedded software; special day on the Internet of Medical Things; etc.
- October 13-18 **International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'2019)**. Topics include: system-level design, modeling, analysis, and implementation of modern embedded, IoT, and cyber-physical systems, from system-level specification and optimization down to system synthesis of multi-processor hardware/software implementations.
- October 13-18 **ACM SIGBED International Conference on Embedded Software (EMSOFT'2019)**. Topics include: the science, engineering, and technology of embedded software development; research in the design and analysis of software that interacts with physical processes; results on cyber-physical systems, which compose computation, networking, and physical dynamics.
- ☺ October 14-20 **TOOLS 50+1: Technology of Object-Oriented Languages and Systems (TOOLS'2019)**, Innopolis (Kazan), Russia. Topics include: new development in object technology; experience reports, technology transfer; challenges of developing software for embedded systems and Internet of Things; reliability and dependability; hybrid and cyber-physical systems modeling and verification; etc.
- October 17-18 **9th Workshop on Model-Based Design of Cyber Physical Systems (CyPhy'2019)**, New York City, NY, USA. In conjunction with ESWEEK 2019.
- ☺ October 20-25 **ACM Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH'2019)**, Athens, Greece. Topics include: all aspects of software construction and delivery, at the intersection of programming languages and software engineering.
- October 20-25 **Onward! 2019**. Topics include: everything to do with programming and software, including processes, methods, languages, communities, and applications; different ways of thinking about, approaching, and reporting on programming language and software engineering research.
- October 21-22 **12th ACM SIGPLAN International Conference on Software Language Engineering (SLE'2019)**. Topics include: areas ranging from theoretical and conceptual contributions, to tools, techniques, and frameworks in the domain of software language engineering; generic aspects of software languages development rather than aspects of engineering a specific language; software language design and implementation; software language validation; software language integration and composition; software language maintenance (software language reuse, language evolution, language families and variability); domain-specific approaches for any aspects of SLE (design, implementation, validation, maintenance); empirical evaluation and experience reports of language engineering tools (user studies evaluating usability, performance benchmarks, industrial applications); etc.
- Oct 28 - Nov 01 **30th IEEE International Symposium on Software Reliability Engineering (ISSRE'2019)**, Berlin, Germany. Topics include: development, analysis methods and models throughout the software development lifecycle; primary dependability attributes (i.e., security, safety, maintainability) impacting software reliability; secondary dependability attributes (i.e., survivability, resilience, robustness) impacting software reliability; reliability threats, i.e. faults (defects, bugs, etc.), errors, failures; reliability means (fault prevention, fault removal, fault tolerance, fault forecasting); reliability of open source software; etc.
- Oct 30 - Nov 04 **16th International Colloquium on Theoretical Aspects of Computing (ICTAC'2019)**, Hammamet, Tunisia. Topics include: semantics of programming languages; theories of concurrency; theories of distributed computing; models of objects and components; timed, hybrid, embedded and cyber-physical systems; static analysis; software verification; software testing; model checking and automated theorem proving; interactive theorem proving; verified software, formalized programming theory; etc.

- November 10-13 24th **International Conference on Engineering of Complex Computer Systems (ICECCS'2019)**, Hong Kong, China. Topics include: verification and validation, security and privacy of complex systems, model-driven development, reverse engineering and refactoring, software architecture, design by contract, agile methods, safety-critical and fault-tolerant architectures, real-time and embedded systems, systems of systems, cyber-physical systems and Internet of Things (IoT), tools and tool integration, industrial case studies, etc.
- November 11-15 34th **IEEE/ACM International Conference on Automated Software Engineering (ASE'2019)**, San Diego, California, USA. Topics include: foundations, techniques, and tools for automating the analysis, design, implementation, testing, and maintenance of large software systems; empirical software engineering; maintenance and evolution; model-driven development; program comprehension; reverse engineering and re-engineering; specification languages; software analysis; software architecture and design; software product line engineering; software security and trust; etc.
- November 25-29 22nd **Brazilian Symposium on Formal Methods (SBMF'2019)**, São Paulo, Brazil. Topics include: techniques and methodologies (such as model-driven engineering, development methodologies with formal foundations, software evolution based on formal methods, ...); specification and modeling languages (such as well-founded specification and design languages, formal aspects of popular languages, logic and semantics for programming and specification languages, code generation, formal methods of programming paradigms (such as objects, aspects, and component), formal methods for real-time, hybrid, and safety-critical systems, ...); theoretical foundations (such as type systems, models of concurrency, security, ...); verification and validation (such as abstraction, modularization and refinement techniques, correctness by construction, model checking, static analysis, formal techniques for software testing, software certification, ...); experience reports regarding teaching formal methods; applications (such as experience reports on the use of formal methods, industrial case studies, tool support).
- November 27-29 20th **International Conference on Product-Focused Software Process Improvement (PROFES'2019)**, Barcelona, Spain. Topics include: experiences, ideas, innovations, as well as concerns related to professional software development and process improvement driven by product and service quality needs.
- December 02-04 17th **Asian Symposium on Programming Languages and Systems (APLAS'2019)**, Bali, Indonesia.
- December 02-05 26th **Asia-Pacific Software Engineering Conference (APSEC'2019)**, Putrajaya, Malaysia. Topics include: agile methodologies; component-based software engineering; configuration management and deployment; cyber-physical systems and Internet of Things; debugging, and fault localization; embedded real-time systems; formal methods; middleware, frameworks, and APIs; model-driven and domain-specific engineering; open source development; parallel, distributed, and concurrent systems; programming languages and systems; refactoring; reverse engineering; security, reliability, and privacy; software architecture, modelling and design; software comprehension, and traceability; software engineering education; software engineering tools and environments; software maintenance and evolution; software product-line engineering; software reuse; software repository mining; testing, verification, and validation; etc.
- December 02-06 15th **International Conference on integrated Formal Methods (iFM'2019)**, Bergen, Norway. Topics include: hybrid approaches to formal modelling and analysis; i.e. the combination of (formal and semi-formal) methods for system development, regarding modelling and analysis, and covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.
- ☺ December 03-06 40th **IEEE Real-Time Systems Symposium (RTSS'2019)**, Hong Kong. Topics include: all aspects of real-time systems, including theory, design, analysis, implementation, evaluation, and experience.
- December 10 **Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!**

2020

- January 14-17 12th **Software Quality Days (SWQD'2020)**, Vienna, Austria. Topics include: improvement of software development methods, processes, and artifacts; 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 20-24 **46th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'2020)**, Limassol, Cyprus. Topics include: novel and innovative methods and technologies in the broader field of software engineering, including both software product and development process aspects; theories, methods and tools aiming at significantly increasing both the quality of software-intensive systems and the productivity of software development. Deadline for early registration: November 25, 2019.
- ☺ January 29-31 **10th European Congress on Embedded Real Time Systems (ERTS'2020)**, Toulouse, France. Topics include: all aspects of critical embedded real-time systems, such as embedded computing platforms and networked systems, model-based system engineering, formal methods, new programming and verification languages, dependability, safety, cyber security, quality of service, fault tolerance, maintainability, certification, etc. Deadline for submissions: October 15, 2019 (regular papers), November 10, 2019 (final short and regular papers).
- ☺ February 01 **Ada Developer Room at FOSDEM 2020**, Brussels, Belgium. FOSDEM 2020 is a two-day event (Sat 1 - Sun 2 Feb). This years' edition includes once more a full-day Ada Developer Room, organized by Ada-Belgium, which will be held on the first day.
- February 22-23 **29th ACM SIGPLAN International Conference on Compiler Construction (CC'2020)**, San Diego, CA, USA. Co-located with CGO'2020, HPCA'2020, and PPOPP'2020. Topics include: processing programs in the most general sense (analyzing, transforming or executing input programs that describe how a system operates, including traditional compiler construction as a special case); compilation and interpretation techniques (including program representation, analysis, and transformation; code generation, optimization, and synthesis; the verification thereof); run-time techniques (including memory management, virtual machines, ...); programming tools (including refactoring editors, checkers, verifiers, compilers, debuggers, and profilers); techniques for specific domains (such as secure, parallel, distributed, embedded or mobile environments); design and implementation of novel language constructs programming models, and domain-specific languages. Deadline for submissions: October 23, 2019 (abstracts), October 30, 2019 (papers), December 13, 2019 (artifacts).
- February 27-29 **13th Innovations in Software Engineering Conference (ISEC'2020)**, Jabalpur, India. Deadline for submissions: October 4, 2019 (research track papers), October 31, 2019 (Ph.D. symposium papers, tutorials, startups and tech-briefing track proposals), November 15, 2019 (student software project contest track submissions).
- ☺ March 23-26 **International Conference on the Art, Science, and Engineering of Programming (Programming'2020)**, Porto, Portugal. Deadline for submissions: October 1, 2019 (workshops).
- March 23-27 **13th IEEE International Conference on Software Testing, Verification and Validation (ICST'2020)**, Porto, Portugal. Topics include: manual testing practices and techniques, security testing, model based testing, test automation, static analysis and symbolic execution, formal verification and model checking, software reliability, testability and design, testing and development processes, testing in specific domains (such as embedded, concurrent, distributed, ..., and real-time systems), testing/debugging tools, empirical studies, experience reports, etc. Deadline for submissions: October 14, 2019 (full research and industry papers), December 9, 2019 (testing tool track), December 12, 2019 (tool demos, posters), January 13, 2020 (doctoral symposium).
- Mar 30 - Apr 03 **35th ACM Symposium on Applied Computing (SAC'2020)**, Brno, Czech Republic.
- Mar 30- Apr 03 **Track on Software Verification and Testing (SVT'2020)**. Topics include: new results in formal verification and testing, technologies to improve the usability of formal methods in software engineering, applications of mechanical verification to large scale software, model checking, correct by construction development, model-based testing, software testing, static and dynamic analysis, analysis methods for dependable systems, software certification and proof carrying code, fault diagnosis and debugging, verification and validation of large scale software systems, real world applications and case studies applying software testing and verification, etc.
- Mar 30- Apr 03 **Embedded Systems Track (EMBS'2020)**. Topics include: system-level design and simulation techniques for embedded systems; testing, debugging, profiling and performance analysis of embedded systems; multicore and SoC-based embedded systems and applications; multithreading in embedded systems design and development; security and dependability support within embedded systems; RTOS for embedded

systems, safety-critical embedded systems; compilation strategies, code transformation and parallelization for embedded systems; memory and storage management for embedded systems; case studies; etc.

- April 14-17 **24th International Conference on Evaluation and Assessment in Software Engineering (EASE'2020)**, Trondheim, Norway. Topics include: assessing the benefits / costs associated with using chosen development technologies; empirical studies using qualitative, quantitative, and mixed methods; evaluation and comparison of techniques and models; replication of empirical studies and families of studies; etc. Deadline for submissions: November 1, 2019 (workshops), December 15, 2019 (full research track abstracts), December 17, 2019 (full research track papers), December 22, 2019 (short paper and artefact track papers, vision paper and emerging results track papers, industry track papers, doctoral symposium and posters track papers), January 13, 2020 (workshop papers), January 17, 2020 (tutorials, discussion panel proposals).
- ◆ April 22-24 **20th International Real-Time Ada Workshop (IRTAW'2020)**, Benicàssim, Spain.
- April 25-30 **23rd European Joint Conferences on Theory and Practice of Software (ETAPS'2020)**, Dublin, Ireland. Events include: ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FoSSaCS (Foundations of Software Science and Computation Structures), TACAS (Tools and Algorithms for the Construction and Analysis of Systems). Deadline for submissions: October 24, 2019 (papers).
- May 23-29 **42nd International Conference on Software Engineering (ICSE'2020)**, Seoul, South Korea. Topics include: the full spectrum of Software Engineering.
- ◆ June 08-12 **25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2020 aka Ada-Europe 2020)**, Santander, Spain. Sponsored by Ada-Europe. Deadline for submissions: January 7, 2020 (journal-track papers, industrial presentation outlines, tutorial and workshop proposals), 31 March 2020 (Work-in-Progress papers).
- July 29-31 **32nd International Conference on Software Engineering Education and Training (CSEET'2020)**, Munich, Germany. Topics include: Teaching formal methods (TFM), Teaching "real world" SE practices (TRW), Software quality assurance education (SQE), Global and distributed SE education (GDE), Open source in education (OSE), Cooperation between Industry and Academia (CIA), Training models in industry (TMI), Continuous education (CED), Methodological aspects of SE education (MAE), etc. Deadline for submissions: November 30, 2019 (workshops, tutorials), February 1, 2020 (abstracts for research papers, industrial experience reports, Journal First submissions, posters, tools, panels), February 8, 2020 (research papers, industrial experience reports, Journal First submissions, posters, tools, panels).
- December 10 **Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!**

ptc® apexada | ptc® objectada®

Complete Ada Solutions for Complex Mission-Critical Systems

- Fast, efficient code generation
- Native or embedded systems deployment
- Support for leading real-time operating systems or bare systems
- Full Ada tasking or deterministic real-time execution

Learn more by visiting: ptc.com/developer-tools



20th International Real-Time Ada Workshop – IRTAW 2020

Hotel Voramar, Benicàssim, Spain

22-24 April 2020

<http://www.ada-europe.org/irtaw2020>

Call for Papers

Since its inception, the International Real-Time Ada Workshop (IRTAW) series has provided a forum where members of the research, user, and implementer communities have singled-out issues with the real-time systems support in Ada, and explored possible solutions to them, and approaches to evolve those aspects of the language to the new frontiers of real-time computing. Over the years, the IRTAW has made important contributions to the 2005 and 2012 revisions of the Ada programming language standard, for the tasking features, the real-time and high-integrity systems annexes, the Ravenscar and the Yorvik Profiles, and – more recently – the parallel execution model of the language due to appear in the 202X revision.

The topics of interest to the 20th edition of IRTAW include but are not limited to:

- Review of the Ada 2012 Issues *vis-a-vis* real-time systems;
- Experiences of use of Ada 2012, in full or by profile, for real-time applications, on single- and multi-processor as well as distributed systems;
- Implementation approaches for Ada 2012 real-time features;
- Review of the Ada 202X Issues with respect to the parallel execution model;
- Early prototyping and use experiences of Ada 202X parallel features, including integration with OpenMP;
- Uses of Ada, possibly with SPARK, in certification-aware domains;
- Review of the Ada language vulnerabilities in relation to TR 24772-2 and -6 by ISO/IEC JTC 1/SC 22/WG 23;
- Contributions to the Ada's conformance test suite for the Real-Time Annex and the upcoming parallel execution model.

Participation in the 20th IRTAW is by invitation, following the submission of a position paper that addresses pertinent topics. Anyone unable to produce a position paper, but still wishing to receive an invitation, is encouraged to send a one-page position statement, in PDF, to the Program Chair, indicating their interests and reasons for attending.

Priority in selection will be given to the authors of submitted papers.

Submission Requirements

Position papers should not exceed ten pages in typical IEEE conference layout, excluding code snippets. They shall be submitted, in PDF, to the Program Chair (see email address below). All accepted papers will appear, in their final form after the event, in a special issue of Ada Letters (ACM Press) also reprinted in the Ada User Journal. Authors with a relevant paper submitted to the 25th Ada-Europe International Conference on Reliable Software Technologies, AEiC 2020 (deadline 7 January, 2020) may offer an extended abstract of the same material to IRTAW.

Important Dates

Paper Submission: **14 February, 2020**

Notification of Acceptance: 6 March, 2020

Confirmation of Attendance: 20 March, 2020

Final Paper Due: 3 April, 2020

Workshop: **22-24 April, 2020**

Program Chair

Tullio Vardanega, *University of Padova, Italy*
tullio.vardanega@unipd.it

Workshop Chair

Jorge Real, *Universitat Politècnica de València, Spain*
jorge@disca.upv.es



UNIVERSITAT
POLITÀCNICA
DE VALÈNCIA

25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2020)

8-12 June 2020, Santander, Spain

(C) RMR

Conference Chair

Michael González Harbour
Universidad de Cantabria, Spain
mgh@unican.es

Program Chair

Mario Aldea Rivas
Universidad de Cantabria, Spain
aldeam@unican.es

Work-in-Progress Chair

Kristoffer Nyborg Gregertsen
SINTEF Digital, Norway
kristoffer.gregertsen@sintef.no

Tutorial & Workshop Chair

Jorge Garrido Balaguer
Universidad Politécnica de Madrid, Spain
jorge.garrido@upm.es

Industrial Chair

Patricia Balbastre Betoret
Universitat Politècnica de València, Spain
patricia@ai2.upv.es

Exhibition & Sponsorship Chair

Ahlan Marriott
White Elephant GmbH, Switzerland
software@white-elephant.ch

Publicity Chair

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



General Information

The **25th Ada-Europe International Conference on Reliable Software Technologies (AEiC 2020 aka Ada-Europe 2020)** will take place in Santander, Spain. The conference schedule includes a technical program, vendor exhibition and parallel tutorials and workshops.

The 2020 edition of the conference continues the major revamp in the registration fees introduced in 2019, redesigned to extend participation from industry and academia, and to reward contributors, especially but not solely, students and post-doc researchers.

Schedule

- 7 January 2020 Submission of journal-track papers, industrial presentation outlines, and tutorial and workshop proposals
- 20 March 2020 Notification of acceptance for journal-track papers, industrial presentations, tutorials and workshops
- 31 March 2020 Submission of Work-in-Progress (WiP) papers
- 30 April 2020 Notification of acceptance for WiP papers

Topics

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

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

- [Design and Implementation of Real-Time and Embedded Systems](#),
- [Design and Implementation of Mixed-Criticality Systems](#),
- [Theory and Practice of High-Integrity Systems](#),
- [Software Architectures for Reliable Systems](#),
- [Methods and Techniques for Quality Software Development and Maintenance](#),
- [Ada Language and Technologies](#),
- [Mainstream and Emerging Applications with Reliability Requirements](#),
- [Achieving and Assuring Safety in Machine Learning Systems](#),
- [Experience Reports on Reliable System Development](#),
- [Experiences with Ada](#).

Refer to the conference website for the full list of topics.

www.ada-europe.org/conference2020

(C) Pachi Hondal

Call for Journal-Track Papers

The **journal-track papers** submitted to the conference are full-length papers that must describe mature research work on the conference topics. They must be original and shall undergo anonymous peer review.

Accepted journal-track papers will get a presentation slot within a technical session of the conference and they will be published in an open-access special issue of the **Journal of Systems Architecture** with no additional costs to authors.

The corresponding authors shall submit their work by 7 January 2020 via the Special Issue web page:

<https://www.journals.elsevier.com/journal-of-systems-architecture/call-for-papers/advances-in-reliable-software-technologies>

Call for WiP-Track Papers

The **Work-in-Progress papers (WiP-track)** are short (4-page) papers describing evolving and early-stage ideas or new research directions. They must be original and shall undergo anonymous peer review. The corresponding authors shall submit their work by 31 March 2020, via <https://easychair.org/conferences/?conf=aeic2020>, strictly in PDF and following the Ada User Journal style (<http://www.ada-europe.org/auj/>).

Authors of accepted WiP-track papers will get a presentation slot within a regular technical session of the conference and will also be requested to present a poster. The papers will be published in the Ada User Journal as part of the proceedings of the Conference.

The conference is listed in the principal citation databases, including DBLP, Scopus, Web of Science, and Google Scholar. The Ada User Journal is indexed by Scopus and by EBSCOhost in the Academic Search Ultimate database.

Call for Industrial Presentations

The conference seeks **industrial presentations** that deliver insightful information value but may not sustain the strictness of the review process required for regular papers. The authors of industrial presentations shall submit their proposals, in the form of a short (one or two pages) abstract, by 7 January 2020, via <https://easychair.org/conferences/?conf=aeic2020>, strictly in PDF and following the Ada User Journal style (<http://www.ada-europe.org/auj/>).

The Industrial Committee will review the submissions anonymously and make recommendations for acceptance. The abstract of the accepted contributions will be included in the conference booklet, and authors will get a presentation slot within a regular technical session of the conference.

These authors will also be invited to expand their contributions into articles for publication in the Ada User Journal, as part of the proceedings of the Industrial Program of the Conference.

Awards

Ada-Europe will offer an honorary award for the best presentation.

Call for Educational Tutorials

The conference is seeking tutorials in the form of educational seminars including hands-on or practical demonstrations. Proposed tutorials can be from any part of the reliable software domain, they may be purely academic or from an industrial base making use of tools used in current software development environments. We are also interested in contemporary software topics, such as IoT and artificial intelligence and their application to reliability and safety.

Tutorial proposals shall include a title, an abstract, a description of the topic, an outline of the presentation, the proposed duration (half day or full day), and the intended level of the tutorial (introductory, intermediate, or advanced). All proposals should be submitted by e-mail to the Educational Tutorial Chair.

The authors of accepted full-day tutorials will receive a complimentary conference registration. For half-day tutorials, this benefit is halved. The Ada User Journal will offer space for the publication of summaries of the accepted tutorials.

Call for Workshops

Workshops on themes 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 days. Workshop proposals should be submitted by e-mail to the Workshop Chair. The workshop organizer shall also commit to producing the proceedings of the event, for publication in the Ada User Journal.

Call for Exhibitors

The commercial exhibition will span the core days of the main conference. 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.

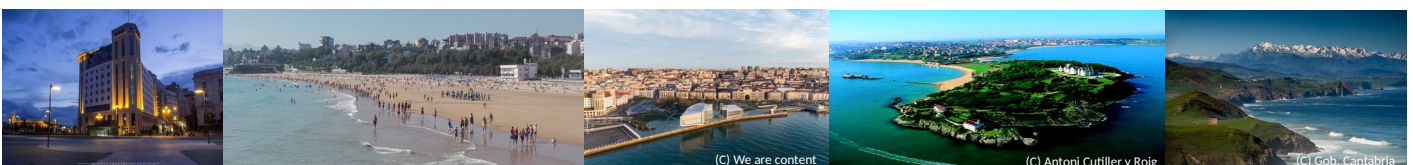
Special Registration Fees

Authors of accepted contributions and all students will enjoy reduced registration fees.

Venue

Santander is a nice tourist city in the north of Spain, with a well-connected airport and at a 100 km drive from Bilbao airport.

The conference venue and hotel is the Bahia Hotel in the city center and beside Santander bay.



Tools to get you there. Safely.

Ada and The GNAT Pro High-Integrity Family



www.adacore.com

AdaCore
The GNAT Pro Company

ARG Work in Progress III

Jeff Cousins CEng FIET

Member and former chair of the Ada Rapporteur Group; email: jeffrey.cousins@btinternet.com

Abstract

The bulk of the work on proposing the next edition of Ada is complete, including the support for parallelism, and we are now entering a prototyping and validation phase.

1 Introduction

This paper presents a further update on the proposed changes for the next edition of Ada, previously referred to as Ada 2020 in anticipation of publication in 2020, but now likely to be some time later. The previous papers were published in the Vol. 38, No. 1, March 2017 and Vol. 39, No. 3, September 2018 editions of the AUJ.

As before, Ada Issues (AIs) are first worked on and approved by the Ada Rapporteur Group (ARG). They are then passed to WG 9 (the ISO/IEC Working Group responsible for Ada) for consideration and approval before eventually being consolidated and sent to ISO for formal processing to create a revised international standard.

In addition to the usual October meeting in the USA in 2018, the ARG held many electronic meetings from December 2018 to May 2019, in an endeavour to get the draft Ada 202X standard ready for submission to WG 9 in June 2019.

Near the end of April 2019 a major Ada compiler vendor asked that the schedule be slipped by a year or two in order to allow time for prototyping the new features, particularly in the priority area of finer grained parallelism, before standardising them. As a result, no AIs were submitted to WG 9 for approval in June 2019.

This caution seems justified as this is a difficult area to get right – the proposal for parallel programming extensions to the C language was recently withdrawn.

In the light of the request to delay the submission of the Ada 202X standard, WG 9 proposed to allow vendors a year to prototype the proposed new features and for comments on the new features to be submitted, then a year to update the proposed new features in response. This proposal was balloted, and the result was unanimously in favour.

Many thanks to John Barnes for his review comments.

2 WG 9 approved

This section describes some of the more important changes to the language that have been approved by WG 9.

The issues listed in the previous paper as being in the pipeline were approved by WG 9, viz:

- Parallel operations (AI12-0119);
- Loop body as anonymous procedure (AI12-0189);
- Generalize expressions that are objects (AI12-0226);
- Contracts for generic formal parameters (AI12-0272);
- Make subtype_mark optional in object renames (AI12-0275).

3 In the pipeline

These have been approved by the ARG but have yet to be approved by WG 9. With many AIs being ARG approved and entering the "pipeline", but none being WG 9 approved and leaving since October 2018, the pipeline is now rather bulging!

3.1 From previous “The Future”s

The following proposals to support parallelism were approved by the ARG:

- Global-in and global-out annotations to specify which global objects a subprogram may access, and in which mode (AI12-0079);
- Reduction Expressions (AI12-0242);
- Explicit chunk definition for parallel loops (AI12-0251-1). (Alternative Parallel loop chunking libraries AI12-0251-2 was voted No Action);
- Map/Reduce Attribute (AI12-0262);
- Parallel Container Iterators (AI12-0266);
- Data race and non-blocking checks for explicit parallelism (AI12-0267).

Some of these are expanded below. Section 5 attempts to describe how these features fit together.

The bulk of the Real-Time proposals were ARG approved:

- Deadline Floor Protocol (AI12-0230);
- Compare-and-swap for atomic objects (AI12-0234);
- Admission Policy Defined for Acquiring a Protected Object Resource (AI12-0276);
- Dispatching Needs More Dispatching Points (AI12-0279);
- CPU Affinity for Protected Objects (AI12-0281);
- Atomic and Volatile generic formal types (AI12-0282);

- Restriction Pure_Barriers (AI12-0290);
- Jorvik Profile (AI12-0291).

Other proposals that were ARG approved include:

- 'Image for all types (AI12-0020);
- Predefined big numbers support (AI12-0208);
- Declare expressions (AI12-0236). This is expanded below;
- Access value aliasing and parameter aliasing (AI12-0240). This was split into 6 alternatives, of which the simplest (relatively!), "Global aspect and access types used to implement Abstract Data Types" (AI12-0240-6), was eventually approved. This is expanded below;
- User-defined literals (AI12-0249);
- Iterator Filters (AI12-0250).

3.2 Array Aggregates; generalized array aggregates (AI12-0212)

Currently, it is quite tedious to initialise a container, one has to create it as an empty container and then add elements one at a time, as in:

```
X : My_Set := Empty_Set;
Include (X, 1);
Include (X, 2);
Include (X, 3);
```

Using the new positional container aggregate, this could be replaced by simply:

```
X : My_Set := [1, 2, 3];
```

Note that this uses square brackets not round ones. This allows the use of [] to indicate an empty container, analogous to "" indicating an empty string.

This is achieved using a new aspect indicating what the appropriate function is for returning an empty container of the particular container type, and what the appropriate procedure is for adding an element to the particular container type, for example:

```
type Set is tagged private
with -- Ada 2012 has these
    Constant_Indexing => Constant_Reference,
    Default_Iterator   => Iterate,
    Iterator_Element  => Element_Type,
    -- This is new
    Aggregate         => (Empty      => Empty_Set,
                          Add_Unnamed => Include);
```

Iteration is also possible within the container aggregate, for example to create a set whose elements all have double the value of the corresponding elements of another set:

```
Doubles_Set : My_Set := [for Item of X => Item * 2];
```

Note that this uses similar syntax to that introduced by *Index parameters in array aggregates (AI12-0061)*.

And – prepare yourself for a shock! – array aggregates are also allowed to use square brackets as an alternative to round ones. This is to emphasise the similarity in characteristics between containers and arrays, allow the use of [] for an empty array, and allow the use of positional notation for a single element array. Remember that

```
Two_Array : array (1 .. 2) of Positive := (1, 2);
```

is allowed, but:

```
One_Array : array (1 .. 1) of Positive := (1);
```

is not allowed.

3.3 Declare expressions (AI12-0236)

As the power of expressions has grown, some felt that it would help to allow local constants and object renamings within an expression, to avoid repeated subexpressions. For example, the postcondition for Fgetc could be clarified from:

```
(if Stream.The_File (Stream.Cur_Position'Old) =
 EOF_Ch
then Stream.Cur_Position = Stream.Cur_Position'Old
and then Result = EOF
elsif Stream.The_File (Stream.Cur_Position'Old) =
 ASCII.LF
then Stream.Cur_Position = Stream.Cur_Position'Old
and then Result = Character'Pos (ASCII.LF)
else
    Stream.Cur_Position = Stream.Cur_Position'Old + 1
and then Result = Character'Pos (Stream.The_File
 (Stream.Cur_Position'Old)))
```

to:

```
(declare
    Old_Pos : constant Position :=
        Stream.Cur_Position'Old;
    The_Char : constant Character :=
        Stream.The_File(Old_Pos);
    Pos_Unchg : constant Boolean :=
        Stream.Cur_Position = Old_Pos;
begin
    (if The_Char = EOF_Ch
     then Pos_Unchg and then Result = EOF
     elsif The_Char = ASCII.LF
     then Pos_Unchg and then Result =
        Character'Pos(ASCII.LF)
     else
        Stream.Cur_Position = Old_Pos + 1
     and then Result = Character'Pos (The_Char)))
```

This uses the reserved words **declare** and **begin**, as for a block, but not **end**, as it is only used within parentheses.

3.4 Global aspect and access types used to implement Abstract Data Types (AI12-0240-6)

Abstract data types are normally provided in Ada by defining the internal structure of the data type in the private part of a package spec, and defining subprograms to access the data in the visible part. For an unbounded container, the implementation will typically include a "root" object with

components of an access type pointing to "subordinate" objects allocated from the heap, but all logically one "compound" object; for example a linked list. Given that the internal details of the data type are not visible, it is not possible to be more specific in the **Global** aspect (added by *Global-in and global-out annotations (AI12-0079)*) about the access types used other than to apply the global mode to the whole package. This would also tend to lump together all objects of the same type whereas two distinct objects of a container type are unlikely to share any storage. Thus data-race checking introduced by *Data race and non-blocking checks for explicit parallelism (AI12-0267)* would be likely to give a lot of false positives.

This proposal adds the aspect **Compound** to indicate that a type is a root with a component of an access type pointing to a subordinate object. This type must be controlled or have a limited partial view to avoid the use of predefined assignment, which wouldn't do a "deep copy" of all the subordinate objects.

The aspect **Internal** is used to indicate that an access object is an internal pointer. Dereferences of such a pointer are not considered **Global** effects. The aspect may also be applied to an access subtype, in which case it specifies the default for any object of the subtype.

For an example, a singly linked list type might have:

```
private
  type Node;
  type Node_Ptr is access Node;
  type Node is limited record
    Elem : Element_Type;
    Next : Node_Ptr with Internal;
  end record
  with Compound;
  type List is ... with record
    Head : Node_Ptr with Internal;
    ...
  end record
  with Compound;
```

The new attribute 'Move is used to move such internal pointers, for example:

```
New_Node.Next := Preceder.Next'Move;
```

when inserting a node into a list. The value of the RHS is copied to the LHS, then the RHS is set to **null**, so that only one pointer can be pointing to the subordinate object.

The second part of this proposal deals with "handles", for example the **File_Type** of **Text_IO** or the **Generator** of **Discrete_Random**. In:

```
procedure Put (File : in File_Type; Item : in String);
```

the **File** parameter is of mode **in** as the parameter itself isn't modified, yet the state associated with the file is modified. This can now be indicated using an overriding global mode, thus:

```
procedure Put (File : in File_Type; Item : in String)
with
  Global => overriding in out File;
```

The third part of this proposal says that access parameters are assumed to be dereferenced without the **Global** aspect having to bother giving details.

3.5 Map-Reduce Attribute (AI12-0262)

This provides a mechanism to take a stream of values – a "value sequence"- from an aggregate (making use of *Array Aggregates; generalized array aggregates (AI12-0212)* described above), and repeatedly apply the same operation to combine the values to produce a single result. Examples include adding a sequence of squares for "sum of squares" or multiplying a sequence of numbers when calculating a factorial. An initial value is required, usually something neutral that has no effect on the result, such as 0 for addition or 1 for multiplication. A parallel version is provided, though if the combining operation is something simple such as addition then the overhead of managing the parallelism is likely to outweigh any performance benefit of performing the additions in parallel. Some examples:

```
-- A reduction expression that outputs the
-- sum of squares
Put_Line ("Sum of Squares is" & Integer'Image
  ([for I in 1 .. 10 => I**2]'Reduce("+", 0));
-- An expression function that returns its result as
-- a Reduction Expression
function Factorial (N : Natural) return Natural is
  ([parallel for J in 1..N => J]'Reduce("*", 1));
```

It is important to note that the values are not put in some temporary array then combined, but are combined "on the fly" as each value is produced.

3.6 Shorthand Reduction Expressions for Objects (AI12-0242)

This provides a shorthand for cases where the object is an array or iterable container. For example:

```
Sum : constant Integer := A'Reduce("+", 0);
```

is short for:

```
Sum : constant Integer :=
  [for Value of A => Value]'Reduce("+", 0);
```

Similarly:

```
Sum : constant Integer := A'Parallel_Reduce("+", 0);
```

is short for:

```
Sum : constant Integer :=
  [parallel for Value of A => Value]'Reduce("+", 0);
```

3.7 User-defined string literals (AI12-0295)

This allows the user to define string literals to be used with a non-string type, by identifying (using an aspect) a function that will do the interpretation. For example:

```
type Varying_String is private
  with String_Literal => To_Varying_String;
```

```
function To_Varying_String (Source :
                           Wide_Wide_String)
  return Varying_String;
...
X : constant Varying_String := "This is a test";
```

This is equivalent to:

```
X : constant Varying_String :=
  To_Varying_String
  (Wide_Wide_String("This is a test"));
```

3.8 Revise the conflict check policies to ensure compatibility (AI12-0298)

AI12-0267 introduced the notion of Conflict Check policies, to control the degree of checking for potential data races. It is important that the default for the new parallel constructs is that all possible conflicts are checked, but for backward compatibility we want the default for tasking constructs to be no checks. Thus the pragma `Conflict_Check_Policy` now permits two separate policies, one for parallel constructs, and one for tasking. The policies for parallel constructs include:

- `No_Parallel_Conflict_Checks`
- `Known_Parallel_Conflict_Checks`
- `All_Parallel_Conflict_Checks`

and similarly the policies for tasking include:

- `No_Tasking_Conflict_Checks`
- `Known_Tasking_Conflict_Checks`
- `All_Tasking_Conflict_Checks`.

The default policy is:

```
pragma Conflict_Check_Policy
  (All_Parallel_Conflict_Checks,
   No_Tasking_Conflict_Checks);
```

If the policies for parallel constructs and tasking checking are the same then the pragma may just take one parameter, thus:

```
pragma Conflict_Check_Policy (No_Conflict_Checks);
```

is a shorthand for:

```
pragma Conflict_Check_Policy
  (No_Parallel_Conflict_Checks,
   No_Tasking_Conflict_Checks);
```

Similarly for `Known_Conflict_Checks` and `All_Conflict_Checks`.

3.9 Nonblocking for Unchecked_Deallocation is wrong (AI12-0319)

This tidies up some of the details of the Nonblocking aspect. The Nonblocking aspect can be used to specify whether a type's initialization, finalization, and assignments allow blocking. It can also be used for objects to describe the storage pool of an access type. These abilities are then used to ensure that the `Unchecked_Deallocation` generic cannot be instantiated with a type whose `Finalize` routine blocks. The standard storage pool(s) are also defined to be Nonblocking.

4 The Future

4.1 Carried over from previous "The Future"s

Ghost code (AI12-0239), code that is added to support specification and verification, remains under investigation.

Thread-safe Ada libraries (AI12-0139) and several alternatives of *Generators/co-routines* (AI12-0197) remain open but are not being actively worked on.

Defaults for generic formal parameters (AI12-0205), and its spin-off *Defaults for generic formal packages and formal "in out" objects* (AI12-0297), were voted Hold (until a future edition). But there are also a couple of other AIs relating to making generics easier for the user: *Implicit instantiations* (AI12-0215) and *Automatic instantiation for generic formal parameters* (AI12-0268) – so it is now intended to merge the most promising features of all of these AIs into a "Best of" AI.

4.2 Making 'Old more sensible (AI12-0280-2)

Currently the following is illegal as the `A.all` in `A.all'Old` is "potentially unevaluated":

```
procedure Proc (A : access Integer)
  with Post =>
    (if A /= null then (A.all'Old > 1 and A.all > 1));
```

It is proposed to relax this. Thinking less of what it may not be possible to evaluate, but more of what CAN be evaluated in advance, the term "known on entry" is introduced to cover such expressions (the most obvious example being a static expression), and if it is possible to tell on entry to a subprogram that an `X'Old` need not be evaluated then it isn't. In the example, `A` is an `in` parameter of an elementary type (which includes access types) so it is passed by copy and cannot change, so if `A` is `null` on entry then `A.all'Old` would not be evaluated.

4.3 Default Global aspect for language-defined units (AI12-0302)

AI12-0079 added the Global aspect, with a default value of "in out all" (i.e. read and write of an unspecified set of global variables), or "null" for Pure packages (i.e. no read or write of any global variable). The former is inconsistent with the requirement that language-defined packages be reentrant. So for most language-defined packages (that are not Pure) it is proposed to add an explicit value of "synchronized in out <unit_name>" (i.e. read and write of the set of global variables that are tasks, protected objects, or atomic objects, of the containing package).

But where some unknown, unsynchronised variable holds state (such as `Current_Input` or `Current_Output` for `Text_IO`) then only "in out <unit_name>" can be stated. This would mean that two concurrent subprogram calls using either `Current_Input` or `Current_Output` would be considered to conflict.

Some parameters may be "handles", for example the `File_Type` of `Text_IO`, which even if of mode `in` may be used by a subprogram to update state. For these this AI

proposes that the value should be "**overriding in [out]** <param>".

5.4 Bounded errors associated with procedural iterators (AI12-0326-2)

AI12-0189-1 introduced the `Allows_Exit` aspect to specify that a subprogram is designed to allow use in a procedural iterator. However, that AI did not explain what restrictions, if any, exist on a subprogram that specifies `Allows_Exit`. It is proposed to make it a bounded error for an `Allows_Exit` subprogram to call the loop body procedure from an abort-deferred operation (unless the whole loop_statement was within this same abort-deferred operation), as this would interfere with implementing a transfer of control.

It is also proposed to add the reserved word **parallel** to the syntax for procedural iterators, and to make it a bounded error to call a loop-body procedure from multiple logical threads of control unless **parallel** is specified.

5 Parallelism - How the pieces fit together

5.1 Parallel constructs and the need to consider potential blocks and conflicts over the access to global data

Parallel operations (AI12-0119) is the prime AI for satisfying the first instruction from WG 9 to the ARG, i.e. "Improving the capabilities of Ada on multi-core and multi-threaded architectures". The parallel constructs were described in one of the previous articles. It is intended that they use light weight threading so as to incur fewer overheads than tasking. To reduce implementation complexity and reduce the risk of deadlock, blocking is not allowed in a parallel construct, thus it is a bounded error to invoke an operation that is potentially blocking during the execution of a parallel construct. The compiler may complain if a parallel sequence calls a potentially blocking operation. It may also complain if parallel sequences have conflicting global side-effects.

5.2 Nonblocking and data race checks

Data race and non-blocking checks for explicit parallelism (AI12-0267) provides the rules to check for blocking operations and for race conditions within parallel constructs. A "data race" occurs when two concurrent activities attempt to access the same data object without appropriate synchronization and at least one of the accesses updates the object. Such "conflicting" concurrent activities are considered erroneous.

This AI was subsequently amended by *Revise the conflict check policies to ensure compatibility (AI12-0298)*, described above.

5.3 Defining Nonblocking

Nonblocking subprograms (AI12-0064-2) provides the mechanism for specifying that a subprogram should not block. As well as contributing to the parallelism proposals, this may also assist timing analysis and deadlock avoidance. *Specifying Nonblocking for Language-Defined Units (AI12-0241)* then uses this mechanism to specify the

Nonblocking status for Ada's own units (that is, child units of packages `Ada` and `System`).

5.4 Defining access to global data

Global-in and global-out annotations (AI12-0079) provides the mechanism for describing the use of global objects by subprograms. It is intended that *Default Global aspect for language-defined units (AI12-0302)* (not yet ARG approved) then uses this mechanism to specify the Global aspect for Ada's own units.

5.5 Parallel iteration over containers besides arrays

Parallel operations (AI12-0119) provided the mechanism iterating in parallel over the elements of an array, *Parallel iterators are defined for containers (AI12-0266)* provides the equivalent mechanism for iterating over containers.

5.6 Control over the degree of parallelism

Explicit chunk definition for parallel loops (AI12-0251-1) gives the user control of the degree of parallelism, for example if processing 100 elements of an array on a 20 core machine one may wish to have 10 logical threads of control (potentially executing on one core each) each processing a group of 10 elements. Such a group is referred to as a "chunk".

5.7 Reduction

Map-Reduce Attribute (AI12-0262), described above, provides a mechanism for taking a stream of values and repeatedly applying the same operation to combine the values to produce a single result. This has a parallel form.

Shorthand Reduction Expressions for Objects (AI12-0242), also described above, provides a shorthand for cases where the object is an array or iterable container.

5.8 An example of the features working together

Not directly related to parallelism, *Index parameters in array aggregates (AI12-0061)* is also used in the example below as it is just such a useful new feature.

```
-- AI12-0241 Specifying Nonblocking for
-- Language-Defined Units
-- AI12-0079 Global-in and global-out annotations --
-- default Global => null" (i.e. no read or write of any --
-- global variable) for Pure packages
-- package
-- Ada.Numerics.Generic_Elementary_Functions
-- with Pure, Nonblocking is
-- function Sqrt (X : Float_Type'Base) return
--                                     Float_Type'Base;
-- ...
with Ada.Numerics.Elementary_Functions;
-- ...
declare
  Max_CPUs_To_Use : constant := 10;
  Max               : constant := 100;
  subtype Range_Type is Positive range 1 .. Max;
  type Float_Array_Type is
                                     array (Range_Type) of Float;
```

```

type Positive_Array_Type is
    array (Range_Type) of Positive;
-- AI12-0061 Index parameters in array aggregates
-- modified by Aggregates; generalized array
-- aggregates (AI12-0212) to use []
Numbers : constant Positive_Array_Type :=
    [for I in Range_Type => I];
Squares      : Positive_Array_Type;
Square_Roots : Float_Array_Type;
Sum_Of_Squares : Integer;
-- AI12-0064-2 - Nonblocking subprograms
-- AI12-0079 Global-in and global-out annotations
function Square (P : Positive) return Positive
with Nonblocking, Global => null;
function Square (P : Positive) return Positive is
(P**2);
begin
-- Ignoring any risk of overflows...
-- AI12-0119 Parallel operations
-- Iteration over the elements of an array
-- AI12-0251-1 Explicit chunk definition for parallel
-- loops
parallel (Max_CPUs_To_Use) for I in Range_Type
loop
    Squares      (I) := Square (I);
    Square_Roots (I) :=
        Ada.Numerics.Elementary_Functions.Sqrt (Float (I));
end loop;
-- AI12-0242 Shorthand Reduction Expressions for
-- Objects (dependent on AI12-0262 Map-Reduce
-- Attribute)
Sum_Of_Squares := Squares'Reduce ("+", 0);
end;

```

6 Conclusions

The Ada 202X proposals are largely complete, but we are now entering a validation phase before putting them forward to WG 9.

RCLAda, or Bringing Ada to the Robot Operating System

Alejandro R. Mosteo

Centro Universitario de la Defensa, Zaragoza, Spain.

Instituto de Investigación en Ingeniería de Aragón, Zaragoza, Spain; email: amosteo@unizar.es

Abstract

The Robot Operating System (ROS) is a commonly used framework in many fields of robotics research, with increasing presence in the industry. The next iteration of this framework, ROS2, aims to improve observed shortcomings of its predecessor like deterministic memory allocation and real-time characteristics. The officially supported languages in ROS2 are C++ and Python, although several other contributed APIs for other languages exist. RCLAda is an API and accompanying tools for the ROS2 framework that enable the programming of ROS2 nodes in pure Ada with seamless integration into the ROS2 workflow.

Keywords: Robotics Software Frameworks, Open Source, Ada 2012.

1 Introduction

The ROS project [1] was born after the experiences with the Player/Stage [2] robotic programming and simulation tools, lead in development by Willow Garage, with the objective of easing the programming of service robots and fostering a common platform for open source robotics. Today, many research institutions contribute and maintain components of ROS. The scope of ROS goes from build tools and communication libraries to reference-frame composition facilities among robotic parts. These core elements enable the bringing together of heterogeneous code bases, with which the community has built a myriad of almost plug-and-play components that provide hardware drivers and software algorithms. Most research-oriented robotic hardware comes with a ROS node that enables its integration out of the box.

ROS was not designed with safety and hard real-time considerations in mind, which has hindered its penetration in industrial domains, despite efforts like the ROSin [3] and ROS-industrial [4] projects. ROS2 design [5] squarely addresses these concerns, documenting dynamic memory allocation and threading characteristics of its API calls, and adopting the Data Distribution Service (DDS) standard [6] for data exchange.

There are three chief parts to ROS2 that give its flexibility and that a developer will be involved with. Firstly, there is a build system (Collective Construction [7], or *colcon* for short), that

relies on XML metadata files to administer dependencies and orchestrate the parallel build of so-called packages. These packages, written in any programming language, constitute the second part and implement the functionality proper of ROS2. Their contents range from low-level hardware drivers for sensor and actuators to high-level algorithms that process and generate data, like planning and navigation strategies, visual recognition, SLAM [8], and so on. Finally, as the third part, there is a client library which is the API that enables seamless interaction between packages and the use of ROS2 concepts like topics, services and actions. This API is structured in a low level C API (RCL, from ROS2 Client Library), intended to enable the integration of other languages on an equal footing, but not necessarily as a direct target for programmers; and high-level client libraries that are bound to the RCL, of which the C++ and Python client libraries are the only officially supported by the ROS2 project, while several others in other languages (Java, C#, Objective C, Swift, Node.js) [9] are provided and supported by external contributors.

The RCLAda project targets the first and third parts just described; it is, fundamentally, an Ada binding to the RCL, supplemented by supporting integration facilities into the *colcon* toolchain. The RCLAda project has reached a level of completeness where it can already be used to write ROS2 packages using only Ada code, with some bits of CMake for the building integration.

This paper presents the design of the RCLAda project in the next section. Ada client packages and examples are shown in Sections 3 and 4, respectively. Facilities for integration into the build pipeline of ROS2 are presented next in Section 5, and conclusions in Section 6 close the paper.

2 Design

This section discusses several design aspects of both the Ada API and its integration in the build system.

2.1 Ada binding

At its core, RCLAda is a binding to a C library and, as such, has to deal with the pitfalls of such a binding: should it be a thin or a thick binding? Should it be partial or complete? Also, ROS2 is still an evolving project in which APIs are not frozen. How can be ensured that no breaking changes go unnoticed?

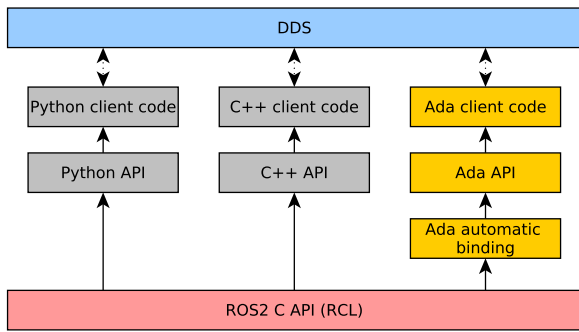


Figure 1: Dependencies and interactions among different kinds of code. At the bottom is the ROS2 Client Library, in C, intended as a common denominator on which all languages build its idiomatic API. The Ada API is a thick binding manually written on top of the autogenerated thin binding. ROS2 client code exchanges data through a common implementation of the Data Distribution Service, which is accessed through the RCL API too (i.e., there is no need to have a DDS API for every language).

In the past, there was little recourse in the Ada world but to manually create such a binding. This was how, for example, the precursor of ROS, the Player project, was made accessible to Ada [10]. Fortunately, the GCC Ada binding generator¹ has reached a point where, at least for the kind of C found in ROS2, its output is readily compilable. Preliminary tests with this generator suggested that the bulk of the mechanical work could be automated for the generation of a direct, low-level binding, with only minimal fixes needed in corner cases where GNAT would bug out even with proper source code generated by the binding generator.

This led to the adoption of a two-layer solution: a low-level thin binding is automatically generated that, consequently, is complete and a direct mapping to the RCL API, if not very Ada-developer friendly. On top of this binding, a thick, Ada-idiomatic one is provided. This solution eliminates the tedious manual work of generating a 1-1 mapping to a large number of C headers, and any changes in the underlying C API are immediately detected, as the thick Ada binding will fail to compile against the thin one.

This approach is reassuring in that the Ada code is safely bound to its underlying C counterpart by an automatic tool; and in that, in the worst case, a complete thin binding exists in case the thick binding were incomplete in some respect (no such need has been identified to date, though).

As it will be shown in the next subsection, the generation of the automatic binding is done as part of the build of the ROS2 code base. Each thin Ada binding is encapsulated within a GNAT project that supplies both the corresponding thick binding, when available, and any overriding files needed for the few cases in which a fix is needed for the automatic binding. To give an idea, in the version of RCLAda at the time of this writing, there are 60 C headers with a counterpart automatic Ada specification, of which only one² needed manual

¹Invoked with `gcc -fdump-ada-spec`.

²https://github.com/ada-ros/rclada/blob/master/gpr_rcl/src/overrides/rcl_allocator_h.ads

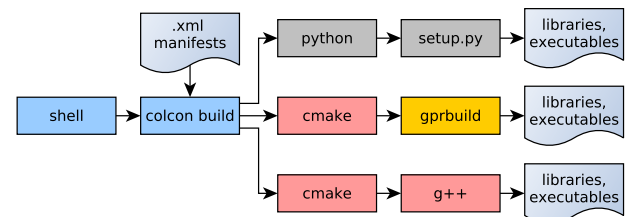


Figure 2: Simplified view of ROS2 build process with *colcon*. Packages may produce final executables or libraries to be linked in by other executables. For example, each package may define new message types, which are materialized as C data types and helper functions (a *typesupport* in ROS2 parlance) that are exported in one such library.

adjustments to circumvent a GNAT bug (for the version of the compiler in Ubuntu 18.04, which is the ROS2 supported development platform).

For the remainder of this paper there will be no more references to the low-level automatic bindings, since they are not used anywhere visible to clients of the thick Ada API. Fig. 1 schematizes the logical dependencies among code layers.

2.2 Allocators

ROS2 provides a way of tailoring dynamic memory allocation via *allocators*, which are passed as an extra argument to calls that allocate dynamic memory in the C API. Since Ada uses storage pools for this use case, a conscious effort has been done to relate both facilities. In practice, Ada storage pools can be transparently used as ROS2 allocators.

2.3 Build integration

ROS2 uses the *colcon* framework for building all its components. In order to be able to combine code coming from heterogeneous languages and organizations, each component is isolated in its own *package*. A package is thus a folder with an XML manifest which provides (simplifying) a list of dependencies on other packages, and a *colcon* build method. Dependencies are used to allow parallel compilation of independent packages, while the build method informs *colcon* on how the package is built (see Fig. 2). Readily available methods are:

- **CMake:** instructs *colcon* to look for a regular CMake `CMakeLists.txt` file to drive the build.
- **ament_CMake:** some extra ROS2-specific CMake macros are made available to the package that simplify finding build artifacts of other packages, and conversely exporting the package artifacts for dependent packages. This is particularly useful for the generation, importing and exporting of *messages*, which are the main data structures transmitted through DDS to exchange information among running ROS2 processes, as explained in Section 3.
- **python:** intended for Python packages, or packages that use Python's `Distutils setup.py`.


```

# LaserScan.msg:
# Single scan from a planar laser range-finder

std_msgs/Header header # acquisition timestamp

float32 angle_min      # start angle of the scan [rad]
float32 angle_max      # end angle of the scan [rad]
float32 angle_increment # angular distance [rad]

float32 time_increment # time between measurements [s]

float32 scan_time      # time between scans [s]

float32 range_min      # minimum range value [m]
float32 range_max      # maximum range value [m]

float32[] ranges       # range data [m]
float32[] intensities  # intensity data
                    # [device-specific units]
# If your device does not provide intensities,
# please leave the intensities array empty.

```

Figure 3: Example of a ROS2 message type definition.

Although *colcon* allows defining new build methods, given the current inability of GNAT's *gprbuild* to execute other commands, or process environment information, RCLAda resorts to using the *ament_cmake* method, providing on top of it a few CMake functions that simplify the integration of GNAT's building toolchain. Details are presented in Section 5.

3 Ada Client Packages

This section delves into the available Ada packages within RCLAda that allow the creation of ROS2 programs. To be precise, ROS2 processes are called *nodes*. A node is one or several threads (typically together in a single executable) that provide some functionality and that are reachable by other nodes via *topics*, *services* and *actions*:

- Topics are named ROS2 addresses (e.g., */robot/laser/readings* to which nodes can publish, to broadcast information, or subscribe, to receive information published to the topic. Each topic has an associated data type. Topics are used for data flows that go only from producer to consumers (e.g., sensor readings).
- Services implement a request/response mechanism, either in blocking or callback fashion. Services are used typically for node configuration calls, sending of data with acknowledgements, or other such two-way communications. Both the request and response have a different associated data type.
- Actions are used to provide high-level support for a sequence of interactions via service calls. Actions are just being introduced to ROS2 and were unavailable at the time of RCLAda first release, and for that reason they are not yet available in the Ada API.

```

declare
  Support : ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get_Message_Support
      (Pkg_Name, Msg_Type);
  -- This call retrieves the typesupport for a
  -- message type defined in a given ROS2 package.

  Msg : ROSIDL.Dynamic.Message := Init (Support);
  -- The Init call allocates a message. Receiving
  -- a message also requires specifying its type.

begin
  Msg ("valid").As_Bool := True;
  Msg ("range_min").As_Float32 := 1.0;
  -- Predefined types are accessed as equivalent Ada types.

  Msg ("ranges").As_Array.Resize (200);
  -- Dynamic arrays can be resized. Static arrays are
  -- allocated during initialization and cannot be resized.

  Msg ("ranges").As_Array (42).As_Int8 := 0;
  -- Array indexing of an array field. Indexing aspects
  -- of Ada 2012 are used to allow a more compact syntax.

  Msg ("image").As_Matrix ((100, 50, 1)).As_Int8 := 0;
  -- A matrix is indexed by providing a tuple of the
  -- appropriate dimensions.

end;

```

Listing 3.1: Example of use of messages. The general structure is to access a field by its name and give its expected type, which returns a reference to the value. There are specific indexing/resizing subprograms available for arrays and matrices.

Knowing this information, there are two main features in the ROS2 API: access to data types to be exchanged, and management of a node and its communications. From another point of view, these correspond to the static data type definitions and the run time node lifetime. ROS2 separates these features into two packages, which naming has been preserved in the Ada API to simplify the use of ROS2 documentation:

On the one hand, the *rosidl³* package deals with *typesupports*, which are C structs and associated functions for initialization/destruction of these data types. ROS2 types are defined in textual files (see Fig. 3) starting from a few predefined types (integers, reals, strings) that can be also composed into arrays or records with named fields. Every file defines one ROS2 type that results in the generation of an associated C struct and its initialization/destruction functions, stored in a linkable library that is available during the building of dependent packages. Packages, in turn, can reuse these types to define and export further types.

On the other hand, the *rcl* package deals with the creation of nodes, topics, services, and actions, and its use by communicating the aforementioned corresponding types through them. Next section will present examples in this regard.

3.1 The ROSIDL Ada packages

RCLAda provides a few Ada packages under the *ROSIDL.** hierarchy that enable the use of ROS2 types; that is, an Ada node uses this hierarchy to *import* typesupports (in the sense of allocating a message of a given type, and accessing its

³ROS2 Interface Definition Language.

Package	Functionality
RCL.Allocators	ROS2 memory management.
RCL.Calendar	ROS2 clock types.
RCL.Clients	Client side of services.
RCL.Errors	ROS2 error codes.
RCL.Executors	Thread pools for nodes.
RCL.Logging	ROS2 logging system.
RCL.Nodes	Node registration.
RCL.Publishers	Publish side of topics.
RCL.Services	Server side of services.
RCL.Subscriptions	Subscribe side of topics.
RCL.Timers	ROS2 timers.

Table 1: Packages and functionality in the RCL.* hierarchy.

fields). Types are made available through subprograms in `ROSIDL.Typesupport`, while access to message fields is done through subprograms in `ROSIDL.Dynamic`.

The reason for the *dynamic* moniker is that there are two ways of using messages in ROS2. The dynamic way (currently used by RCLAda) is through an introspection library that provides a message in-memory layout at run time, and the static way is through generated C/C++/Python data types. RCLAda does not yet generate static Ada types to enable this usage.

Listing 3.1 shows how message fields can be accessed for reading and writing. With the exception of variable-length strings, that require the resizing of the underlying message, all fields are accessed through reference types, allowing minimal copying of information from/to messages. Also, bounds checking is performed as expected in an Ada context.

3.2 The RCL Ada packages

The rest, and bulk, of facilities to create a ROS2 node, and to interact with other nodes, are under the RCL.* hierarchy. Table 1 summarizes the names and functionality in each child package. The use of most of these packages will become clear with the examples presented in the next section. As an introduction, Listing 3.2 shows the creation of an empty node.

```
with RCL.Nodes;

procedure Null_Node is

  Node : RCL.Nodes.Node := RCL.Nodes.Init ("my_node");
  -- The Init call readies a node for use, and gives it
  -- the name with which it will be known to other nodes.

begin
  Node.Spin (During => Forever);
  -- Spin is a blocking call. If we had defined topics,
  -- services or other callbacks, these would be called
  -- as data became available. Also, information could
  -- be published to some topic from another thread.
  -- Alternatively, we could spin for a while and queue
  -- data whenever the node is not spinning.
end Null_Node;
```

Listing 3.2: A minimal null node.

4 Examples

ROS2 client libraries follow a tradition of implementing the same few basic examples. This allows users to compare idioms for the same functionality in different languages and these examples serve as a minimal tutorial. Two pairs of these examples, that work in tandem, are presented in the next subsections. First, publish/subscribe nodes are shown (a topic), followed by a client/server interaction (a service).

4.1 Topics

The simplest form of communication in ROS2 are topics, in which one node publishes information that can be received by any number of other nodes. In the example collection, these nodes receive the names of *talker* and *listener*. The talker creates a topic and starts publishing a message every second. At any time, a listener can subscribe to said topic and it will start to receive the messages. Past messages are unknown to it.

Listings 4.1 and 4.2 show bare-bones code for these two classes of nodes. These nodes can obviously interact with their counterparts in other languages, e.g. the standard C++ and Python examples. A slightly more sophisticated version of the talker, using a ROS2 timer, can be found in the RCLAda repository.

4.2 Services

Services involve a request and a response, not necessarily of the same type. Both the server and the client must register a callback to be able to receive messages having an unknown arrival time. However, for the client, a blocking version exists that hides this callback, for simple use cases. The examples in Listings 4.3 and 4.4 exemplify this modus operandi for a service that receives two integers and returns their sum.

5 Integration

Up to this point, mostly Ada code has been presented. The compilation of such Ada code in the context of ROS2 requires the use of libraries (for ROS2 typesupports) which location cannot be known in advance; also, Ada nodes must be made known to the rest of the ROS2 ecosystem for some features to work seamlessly, like tab-completion, launching through ROS2 command-line tools, and other related issues. This section is devoted to present the RCLAda features that provide this integration.

Since the `ament_cmake` build mode of `colcon` was chosen, in practice this means that ROS2 packages containing Ada nodes must provide a `CMakeLists.txt` file that declares the libraries and executables being built, and the ROS2 data types being imported and exported. Since the GNAT compilation model is not at present supported by CMake, nor are the elaboration and binding stages well-suited to easy CMake integration, a number of CMake functions have been defined that forward the compilation to `gprbuild`. Also, given that importing messages requires the existence of GPR projects for their externally defined C libraries, these GPR projects can be generated on the fly too with appropriate CMake functions.

```

with RCL.Nodes;
with RCL.Publishers;

with ROSIDL.Dynamic;
with ROSIDL.Typesupport;

procedure Talker is
  Support : constant ROSIDL.Typesupport.Message_Support :=
    ROSIDL.Typesupport.Get_Message_Support
      ("std_msgs", "String");
  -- Obtains the type for the messages to be published.
  -- This is the basic String type from ROS2.

  Node : RCL.Nodes.Node := RCL.Nodes.Init ("talker");
  Pub : RCL.Publishers.Publisher :=
    Node.Publish (Support, "/chatter");
  -- Creates a topic using the just initialized node.
  -- The topic requires a type and a hierarchical name.

  Count : Natural := 0; -- A counter of sent messages.
  Msg : ROSIDL.Dynamic.Message :=
    ROSIDL.Dynamic.Init (Support);

begin
  loop
    Count := Count + 1;
    Msg ("data").Set_String ("Hello_World:" & Count'Img);
    -- The String type contains a single field called
    -- data. Its value is updated here to reflect the
    -- amount of messages sent to date.

    Pub.Publish (Msg);
    -- Publish it to any subscribed listeners.

    Node.Spin (During => 1.0); -- Will drift...
    -- Spinning the node ensures events are dispatched.
    -- We take advantage of the spin period to wait 1s.
  end loop;
end Talker;

```

Listing 4.1: Talker example node.

```

-- withs omitted for conciseness.

procedure Listener is

  procedure Callback
    (Node : in out RCL.Nodes.Node'Class;
    Msg : in out ROSIDL.Dynamic.Message;
    Info : ROSIDL.Message_Info) is
    begin
      RCL.Logging.Info
        ("Got_chatter:_" & Msg ("data").Get_String & "'");
      -- Print using the standard ROS2 logging facilities.
    end Callback;

  Node : RCL.Nodes.Node := RCL.Nodes.Init ("listener");

begin
  Node.Subscribe
    (ROSIDL.Typesupport.Get_Message_Support
      ("std_msgs", "String"),
      "/chatter",
      Callback'Access);
  -- Subscribe to a topic. Whenever a message is received,
  -- the provided callback will be called with the data.
  -- The data type of publisher and subscriber must match.

  Node.Spin (During => Forever);
end Listener;

```

Listing 4.2: Listener example node.

```

-- withs omitted for conciseness.

procedure Server is
  -- Node declaration omitted for conciseness.

  procedure Adder
    (Node : in out RCL.Nodes.Node'Class;
    Req : ROSIDL.Dynamic.Message;
    Resp : in out ROSIDL.Dynamic.Message)
    -- The profile of a service includes the request
    -- made by the client and an initialized response
    -- of the appropriate type.
    is
      A : constant ROSIDL.Int64 := Req ("a").As_Int64;
      B : constant ROSIDL.Int64 := Req ("b").As_Int64;
    begin
      Resp ("sum").As_Int64 := A + B;

      -- Once the callback ends, the response will be
      -- sent to the requester automatically.
    end Adder;

begin
  Node.Serve
    (ROSIDL.Typesupport.Get_Service_Support
      ("example_interfaces", "AddTwoInts"),
      "add_two_ints",
      Adder'Access);
  -- Creating a service is as simple as giving the
  -- callback and the typesupport for the service,
  -- which in turn specifies the types of both the
  -- request and the response.

  Node.Spin (During => Forever);
end Server;

```

Listing 4.3: Server example node.

```

-- withs omitted for conciseness.

procedure Client is -- Synchronous version
  -- Node declaration omitted.

  Support : constant ROSIDL.Typesupport.Service_Support :=
    ROSIDL.Typesupport.Get_Service_Support
      ("example_interfaces", "AddTwoInts");
  -- Note the specific Service_Support vs Message_Support.
  Request : ROSIDL.Dynamic.Message :=
    ROSIDL.Dynamic.Init (Support.Request_Support);
  -- The message is initialized using the Request_Support.
  -- An analogous Response_Support function exists.

begin
  Request ("a").As_Int64 := 2;
  Request ("b").As_Int64 := 3;
  -- Fill in the request data.

  declare
    Response : constant ROSIDL.Dynamic.Message :=
      Node.Client_Call (Support,
        "add_two_ints",
        Request);
  -- By using the blocking call profile, the node is
  -- spun internally. An optional timeout can be used,
  -- and RCL_Timeout may be raised if expired.

begin
  RCL.Logging.Info
    ("Got_answer:" & Response ("sum").As_Int64.Image);
end;
end Client;

```

Listing 4.4: Client example node.

```

project(my_ada_ros2_project VERSION 0.1.0)
# Standard CMake declaration of a project.

find_package(rclada_common REQUIRED)
# Mandatory to import the Ada CMake functions.

ada_begin_package()
# This, and its companion ada_end_package(), are in charge
# of setting up and propagating the Ada environment for this
# and dependent packages.

find_package(rclada REQUIRED)
find_package(rosidl_generator_ada REQUIRED)
# Standard calls to find the two ROS2 packages that contain
# the RCLAda thick bindings. Mandatory to write Ada nodes.

ada_add_executables(
  my_ada_project      # CMake target name
  ${PROJECT_SOURCE_DIR} # folder containing a GPR file
  bin                # relative path to binaries
  my_ada_main        # target binaries
# Associates a CMake target with a standard GNAT project.
# At make build time, gprbuild will run with proper context.
# The output executables will be made known to ROS2 tools.
# ${PROJECT_SOURCE_DIR} is provided by CMAKE, and points
# to the folder in which CMakeLists.txt is located; that
# is, to the root of the ROS2 package.
# The path for generated binaries is relative to the prev-
# ious discussed folder. Finally, a list of executables
# generated by the single GPR project file must be given.

ada_end_package()

```

Listing 5.1: Example of configuration file for CMake.

CMake relies heavily on environment variables to provide information about the build environment, like the location of libraries, headers, and so on. The same mechanism is used by RCLAda to propagate Ada-specific information, although this is done transparently to the user. Thus, the user can directly use the RCLAda projects and other generated GPR projects, since their actual locations are passed to `gprbuild` when invoked.

An example of `CMakeLists.txt` file presenting most available CMake Ada-specific functions can be seen in Listing 5.1. In addition, an equivalent function to `ada_add_executable` devoted to exporting libraries exists (named `ada_add_library`). Finally, used internally by RCLAda, but probably useful to people writing nodes that require bindings to C libraries (e.g., for hardware drivers), the `ada_generate_binding` enables the generation at compile-time of an automatic thin binding and its integration into a given GPR project.

6 Conclusions

ROS and ROS2 have positioned themselves as the default go-to frameworks for research in service robotics. ROS2 strives to fix the shortcomings identified in ROS, and is expected to continue playing a significant and growing role in robotics research and industries in the years to come. For this reason, making Ada readily available in its ecosystem is a pursuit that ought to benefit roboticists with interest in a safer, time-proven language such as is Ada, and promote the language in a popular open source context. ROS2 aims to be more amicable to real-time and safety-critical robotics deployments

even outside of the research community, for which Ada and SPARK are ideal client languages.

RCLAda enables the immediate writing of ROS2 nodes in Ada, without the hassle of having to create custom bindings or mixed-language programs, nor of having to battle a build system in which the GNAT toolchain is not trivial to be integrated. To this end, RCLAda provides both an Ada thick-binding to the ROS2 API, and a set of CMake functions for the ROS2 build system. Examples for the use of basic features of both RCLAda and ROS2 are provided that should help newcomers in the writing of new Ada ROS2 nodes.

RCLAda is available under an open source license to interested parties at <https://github.com/ada-ros/ada4ros2/>.

Acknowledgements

This work has been supported in part by research projects ROBOCHALLENGE (DPI2016-76676-R-AEI/FEDER-UE), ASIMOV (CUDZ2018-03), RoPERT (DGA-T45_17R/FSE), and the AdaCore company.

References

- [1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng (2009), *ROS: an open-source Robot Operating System*, in ICRA workshop on open source software, vol. 3, p. 5, Kobe, Japan.
- [2] B. Gerkey, R. T. Vaughan, and A. Howard (2003), *The Player/Stage project: Tools for multi-robot and distributed sensor systems*, in Proceedings of the 11th Int. Conf. on Advanced Robotics, vol. 1, pp. 317–323.
- [3] Fraunhofer IPA, *ROSin: ROS-industrial quality-assured robot software components*, <https://rosin-project.eu/>. Accessed: 2019-Sep-06.
- [4] S. Edwards and C. Lewis (2012), *ROS-industrial: applying the robot operating system (ROS) to industrial applications*, in IEEE Int. Conference on Robotics and Automation, ECHORD Workshop.
- [5] Open Source Robotics Foundation, Inc, *ROS2 design*, <http://design.ros2.org/>. Accessed: 2019-Sep-06.
- [6] G. Pardo-Castellote (2003), *OMG data-distribution service: Architectural overview*, in Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, pp. 200–206, IEEE.
- [7] D. Thomas, *Collective construction (colcon)*, <https://github.com/colcon>. Accessed: 2019-Sep-06.
- [8] ROS Index, *Available packages for ROS2 Dashing*, <https://index.ros.org/packages/#dashing>. Accessed: 2019-Sep-06.
- [9] ROS Index, *About ROS2 client libraries*, <https://index.ros.org/doc/ros2/Concepts/ROS-2-Client-Libraries/>. Accessed: 2019-Sep-06.
- [10] A. R. Mosteo and L. Montano (2007), *SANCTA: An Ada 2005 general-purpose architecture for mobile robotics research*, in International Conference on Reliable Software Technologies, pp. 221–234, Springer.

Modeling CPS Swarms: An Automotive Use Case

M. Schranz, M. Sende

Lakeside Labs, Klagenfurt, Austria; email: {lastname}@lakeside-labs.com

A. Bagnato, E. Brosse

Softteam Research and Development Department, Paris, France; email: {firstname.lastname}@softteam.fr

A. Eckel

TTTech Computertechnik, Vienna, Austria; email: andreas.eckel@tttech.com

Abstract

Swarms of cyber-physical systems (CPSs) find their application in many domains. Smart traffic is a very prominent application increasingly taking advantage of the self-organizing capabilities of swarms. In this paper we use and extend existing models from libraries designed for CPS swarm behavior that we proposed in other publications. We present a three-level hierarchy to add real-world CPS swarm requirements including, e.g., emergency routines and obstacle detection according to CPS aspects. Furthermore, this hierarchical concept is applied to an automotive use case, where autonomous trucks form platoons to save energy and reduce CO2 emission.

Keywords: Cyber-Physical Systems, Design and Modelling, Swarm Behavior, Autonomous Cars.

1 Introduction

The world is getting connected: Cyber-physical systems (CPSs) are characterized by intensive interactions between their hardware and software components, and their operation in the dynamically changing, unpredictable real world [1]. Beside smart homes, search and rescue scenarios, or environmental monitoring, they find their application in the domain of smart traffic. Multiple CPSs are used in this application including autonomous cars from different levels of autonomy, smart dynamic infrastructure (e.g., smart phones, drones) and static infrastructure (e.g., traffic lights, radar). If a set of these agents have, e.g., the same goal, they can form swarms to operate, take decisions, conclude on the outcome and react on the dynamic environment in a self-organized way. Moreover, swarms of CPSs would offer capabilities like adaptability, scalability, and robustness.

In the paper we will focus on an automotive vision scenario developed in the CPSwarm project [2]. The aim is to form platoons of connected freight vehicles. The leading vehicle prescribes the actions and decisions (e.g., navigation, decision on take-over maneuvers, sequencing maneuvers, lane change) for the follow-up vehicle(s) that will make use of the leading vehicle's actions. The follow-up vehicle will need

full autonomous driving capability and environmental awareness. However, they will follow the leading vehicle in a preset distance even when they have to make decisions, e.g., lane change maneuver on their own due to an emergency situation, or leaving the platoon if the route to the destination deviates). The follow-up vehicles will also take over full control in case the lane change needs to be interrupted for the complete swarm due to other traffic prohibiting to change lanes. The focus of this paper is on showing a concrete example of modeling an automotive CPS swarm with a platooning behavior. Moreover, the modeling approach in [3] needs to be expanded to include behaviors and actions that are necessary for CPS swarms to operate in a real-world environment.

The paper is structured as follows: Section 2 recaps work already done in the area of modeling CPS hardware and CPS swarm behaviors. Section 3 describes the vision scenario and the modeling of the automotive use case. Finally, Section 4 concludes the paper and gives an outlook on further work in modeling and abstraction.

2 Previous Work

The CPSwarm project positions itself in the domain of CPS design and engineering. It aims at providing a tool chain enriched with innovative methodologies to model, design, optimize, simulate, and deploy next generation swarms of CPSs. The core of the CPSwarm tool chain is to provide a workbench to explicitly manage behavior and emerging properties of swarms of CPS. This CPSwarm workbench will ease development and integration of complex swarms of heterogeneous CPSs that collaborate based on local policies and that exhibit a collective behavior capable of solving complex, industrial-driven, real-world problems. For more details on the project, we refer the reader to Bagnato et al. [2].

Within the CPSwarm project we refer to models, as used in Model-Based System Engineering (MBSE) methodology, using concepts of the Unified Modeling Language (UML)¹ and/or the Systems Modeling Language (SysML)². SysML block diagrams are used to model the CPS hardware (sensors, actuators, and communication) as proposed in [3], and the interface to the behavior. On the other, UML state machine

¹<https://www.omg.org/spec/UML/2.5.1/>

²<https://www.omg.org/spec/SysML/About-SysML/>

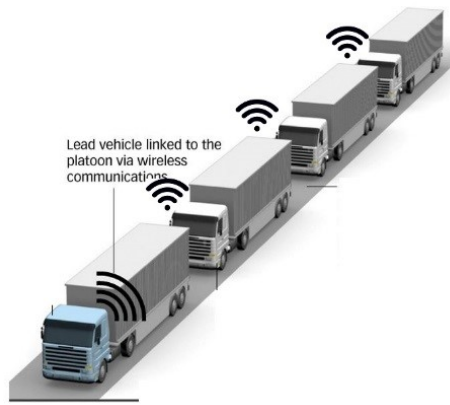


Figure 1: Vehicles in platooning configuration.

diagrams are used to model the behavior of the CPS swarm using states [3].

In this paper, the developed libraries and models are adapted and enhanced for the automotive use case. Therefore, we introduce a three level modeling hierarchy to meet real-world requirements for a CPS swarm. For the automotive use case, we focus on the modeling of the CPS behavior rather than on the modeling of the CPS hardware. The modeling tool we used in [3] and in this paper is Modelio³, an open-source modeling tool.

3 Modeling an Automotive CPS Swarm

3.1 Automotive Scenario

For this scenario, the freight vehicle platooning concept is addressed combined with swarm behavior and evolutionary algorithms. The vehicle platooning has following concept (see Figure 1):

- The leading vehicle has autonomous driving capability and prescribes the actions and decisions (e.g., navigation, decision on take-over maneuvers, sequencing maneuvers, lane change) for the follow-up vehicles.
- The follow-up vehicles have autonomous driving capability and environmental awareness. They follow the leading vehicle's actions.

Freight vehicle platooning holds great potential to make road transport safer, cleaner and more efficient in the future. Platooning results in a lower fuel consumption, as the freight vehicles drive closer together at a constant speed, with less braking and accelerating. Freight vehicle platooning has also the potential to reduce CO₂ emissions. Likewise, connected driving can help improve safety, as braking is automatic with virtually zero reaction time compared to human braking. Finally, platooning also optimizes transport by using roads more effectively, helping deliver goods faster and reducing traffic jams.

As example, the following final vision scenario has been defined (see Figure 2): The yellow vehicle is an autonomously

³<https://www.modelio.org/>

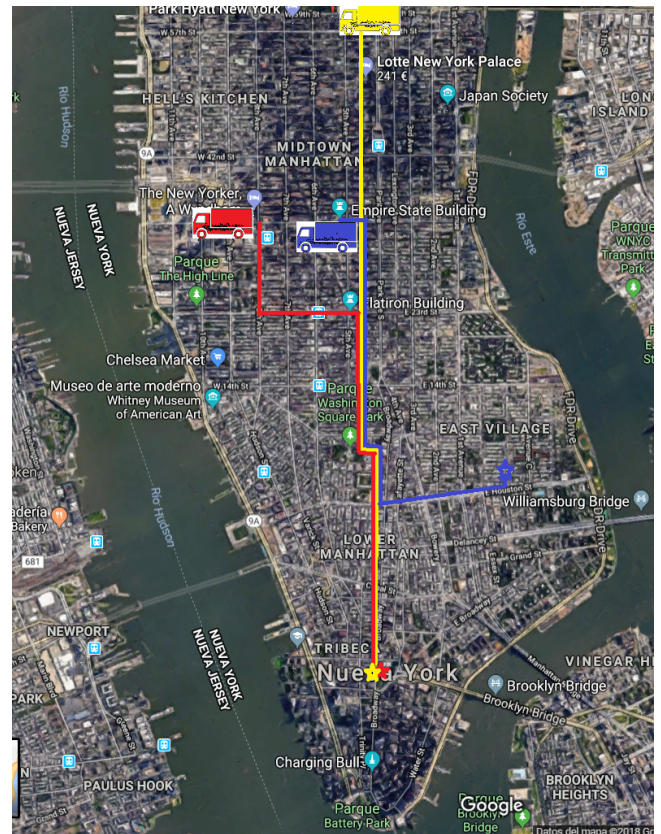


Figure 2: Vision scenario for three freight vehicles in a platoon.

driving passenger transport vehicle that takes a group of tourists at the Hotel Plaza that want to visit the Tribute Museum. The red vehicle is an autonomously driving special goods transport vehicles and is at the Pennsylvania Train Station where it has picked up a new sculpture to bring it to the Tribute Museum. The blue vehicle is also an autonomously driving passenger transport vehicle that has picked up a group of travelers that have already visited the Empire State Building and want to go now to the Nuyorican Poets Cafe located on the East Village, where they will have some rest. Since a certain part of the route is common for the three vehicles, they decide to create a platoon. The blue vehicle joins the yellow vehicle as a follower whereas the yellow one leads the platoon. When they arrive at the Flatiron Building, the red vehicle joins them as a second follower vehicle. The three vehicles run together until Houston street where the blue vehicle leaves the platoon to go to its final destination. The yellow and the red vehicles keep the platoon until the Tribute Museum where they both reach their final destination.

3.2 Models

The main idea of modeling a CPS swarm is to have a formulation and a definition of the models that allows to firstly visually represent the local behavior of the CPSs, and secondly generate source code out of the models.

In this paper we expand the modeling approach of [3] using a three-level hierarchy (see Figure 3). The first hierarchy models each algorithm available in a swarm algorithm library as a block (SysML) together with several inputs and outputs.

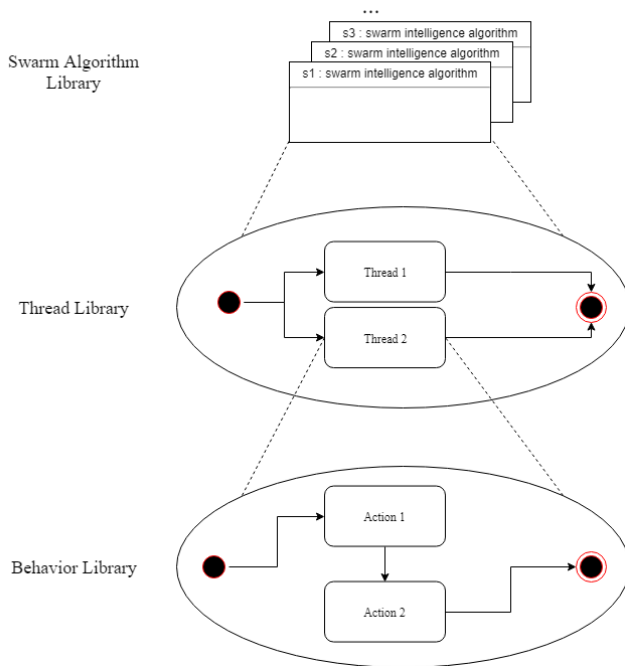


Figure 3: The three-level hierarchy to model real-world CPS applications.

The second and third hierarchy models the individual CPS swarm behavior using state machine diagrams (UML). Each swarm algorithm (first level) can be represented with a state machine diagram using threads (second level). This thread library level has been designed in collaboration with industrial experts. The main reason is to include and realize real-world CPS swarm applications, like the automotive one: Real-world applications demand for additional routines during the processing of their behavior. It includes parallel running tasks, so-called threads, to execute, e.g., emergency routines, centrally triggered or locally exchanged status updates which are mandatory but separate to the platooning (core) behavior. The platooning core behavior is modeled with the behavior library in the third level of the modeling structure. In the course of this paper we focus on the second and third hierarchy level.

For the presented automotive use case, the thread level is modeled as shown in Figure 4. The Platooning Behavior is realized as an state machine running in parallel with Event Monitoring, Obstacle Detection and Status Message behaviors. The Platooning Behavior is terminated in case of a *missionAbort*. The event *obstacleDetected* triggers an emergency routine.

The next hierarchy level of the state machine is given in Figure 5. This is the core behavior belonging to the *Platooning Behavior* state realized as thread in Figure 4. Each freight vehicle is powered on by *Startup* and goes into Idle mode. In case of a *shutdown* event, it triggers a power off. In case of the *missionStart* event it triggers a *Shortest Path Algorithm* behavior to calculate the shortest path to its destination. It starts the route with a *Free Driving* behavior following the lane. As soon as it meets another vehicle (using the *meetVehicle* event) the *Select Role* behavior is initiated to decide on being

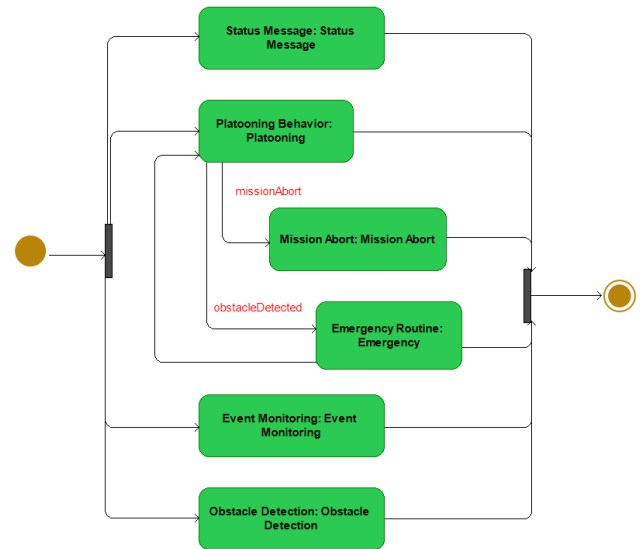


Figure 4: Threads of the automotive use case (level two of the modeling hierarchy).

a leading or a following vehicle. If it is a leading vehicle, it turns again into the *FreeDriving* behavior until it a) meets again another vehicle or b) the mission is over (*missionOver* event). If it is a following vehicle, it follows the lead (instead of the lane) until a) the mission is over (*missionOver* event), or the way to destination changes, so the vehicle is again in the *Free Driving* behavior and follows the lane instead of the leading vehicle.

In addition, we designed also a preliminary state machine for the *Emergency Routine*. If an obstacle is detected (*obstacleDetected* event), the vehicle performs a *Change Lane* or *Emergency Brake* behavior, depending on lane availability (see Figure 6).

4 Conclusion

As we have seen in previous work, modeling CPS swarms is a rather untouched topic [3]. Therefore, we propose a modeling approach to model the CPS hardware and the swarm behavior [3]. In this paper we propose a three level modeling hierarchy on the example of an automotive use case. This approach allows to focus on the swarm behavior, and related real-world driven requirements including emergency routines, obstacle detection, etc. In future work we will introduce additional safety and security routines including available standards to meet additional real-world CPS swarm requirements.

Acknowledgment

The research leading to these results has received funding from the European Union Horizon 2020 research and innovation program under grant agreement No 731946, CPSwarm Project.

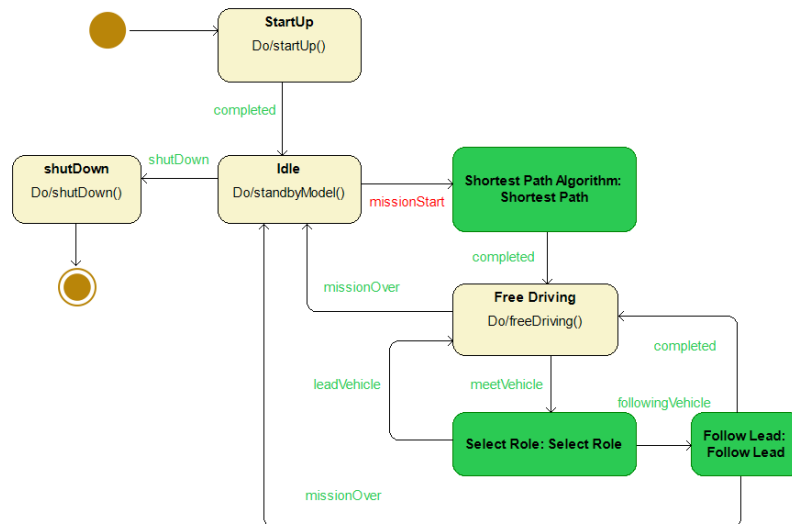


Figure 5: State Machine of the Platooning Behavior (level three of the modeling hierarchy).

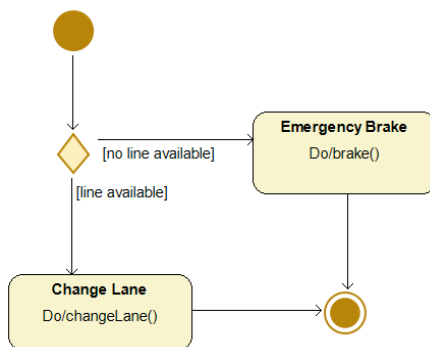


Figure 6: Emergency routine (level three of the modeling hierarchy).

References

- [1] E. A. Lee (2008), *Cyber physical systems: Design challenges*, in Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, pp. 363–369.
- [2] A. Bagnato, R. K. Bíró, D. Bonino, C. Pastrone, W. Elemenreich, R. Reiners, M. Schranz, and E. Arnautovic (2017), *Designing swarms of cyber-physical systems: The H2020 cpswarm project: Invited paper*, in Proceedings of the Computing Frontiers Conference, CF'17, pp. 305–312.
- [3] M. Schranz, A. Bagnato, E. Brosse, and W. Elemenreich (2018), *Modelling a cps swarm system: A simple case study*, in Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, pp. 1–10.

Modeling Swarm Intelligence Algorithms for CPS Swarms

M. Schranz, M. Sende

Lakeside Labs GmbH, Klagenfurt, Austria; email : {lastname}@lakeside – labs.com

A. Bagnato, E. Brosse

Softteam Research and Development Department, Paris, France; email : {firstname.lastname}@softteam.fr

Abstract

Swarms of cyber-physical systems (CPSs) have a high potential for innovative and successful applications. Swarm intelligence algorithms are one approach to handle the increased complexity that comes with the high number of CPSs in a swarm. In such algorithms, individual CPSs follow simple rules that lead to an emergent behavior. This has advantages such as adaptability, scalability, and robustness without relying on a central control. Nevertheless, the design of such systems is still a hard problem. In this paper, we propose an approach to model the local behavior of individual CPSs using swarm intelligence algorithms from a software engineering perspective. Therefore, we introduce a two-level hierarchy: The first level models the swarm intelligence algorithms as opaque blocks which are detailed in the second level by individual actions as activity diagrams. The individual actions allow the designer to customize existing and create new swarm intelligence algorithms. This is done by extracting single actions from original swarm intelligence algorithms and assembling them in the activity diagrams. Furthermore, the two-level approach allows us to link the modeling of the CPS hardware with the modelling of the behavior of each CPS in a swarm. Finally, we demonstrate the modeling technique by applying the swarm intelligence algorithm BEECLUST to the robotic platform Spiderino.

1 Introduction

Cyber-physical systems (CPSs) are one of the big research topics nowadays [1]. In general, a CPS is characterized by intensive interactions between its hardware and software components, and the interactions within a dynamically changing physical world. Thus, CPSs are not working in a controlled environment and must be robust against inputs from a dynamic, unpredictable world [2]. Prominent application domains include smart mobility, smart grids, and smart houses. Typically, more than one CPS is used in these applications. Such a system of CPSs makes the design and operation even more complex. A system of multiple CPSs is hard to program, predict, and control. For daily live processes, the need for

features like adaptability, scalability, and robustness further complicate the operation of systems of CPSs.

To handle this increasing complexity of systems of CPSs, we take inspiration from natural swarms like ant or bee colonies, bird flocks, and fish shoals [3]. They have evolved their behavior over millennia to deal with dynamic environments and complex challenges including non-deterministic polynomial-time (NP) hard problems. The concept of swarms in nature can be mapped to swarms of CPSs, as they come with similar behaviors, including but not limited to pursuing a specific goal, aggregating or dispersing in the environment, communicating, and memorizing (local states, morphologies, etc.) [4]. To sum up, a swarm can be characterized by simple swarm members with basic capabilities following simple rules leading to an emergent behavior yielding the main advantages adaptability, scalability, and robustness—all of this without a central control.

Generally, research on swarm intelligence algorithms is a rather young field that started with the Boids model in 1987 [5]. As such, modeling swarm intelligence algorithms from a software engineering perspective is rather untouched [6]. Swarm behaviors in the robotics community are commonly modeled by textual description [7], finite state machines (FSMs) [8], or activity diagrams [9]. They are well suited to model swarm behaviors, but are limited to the software side of CPSs even though it is important to consider also the hardware platform when designing the behavior [10]. Therefore, when designing swarms of CPSs, engineers reach their limits, as no common approach for modeling swarms of CPSs is currently available.

In this paper, we propose a common modeling standard for swarm algorithms in swarms of CPSs that takes into account the model of the system, both from hardware and software side. The hardware model defines all interfaces for the swarm behavior of the CPSs. The software model defines the swarm behavior of the CPSs using activity diagrams. This work contributes to the EU-H2020 project CPSswarm by focusing on modeling and optimization of swarms of CPSs [11]. In CPSswarm, a workbench is developed through a combination of existing tools. This workbench allows to design, optimize, simulate, and deploy of swarms of CPSs. This approach eases the process in designing and engineering CPS swarms.

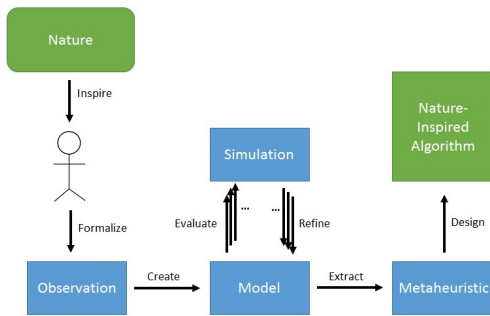


Figure 1: Process of designing a swarm intelligence model and the corresponding algorithm (adapted from [14]).

Section 2 presents the state of the art in modeling swarms of CPSs. It furthermore presents a summary of previous work on a library of models for CPSs that has been presented in a simple case study by Schranz et al. in [12]. Section 3 presents the common approach to model swarm algorithms for swarms of CPSs. Section 4 deals with modeling of inter and intra swarm communication. An example for the common modeling approach is given in Section 5, where we take the robotic platform Spiderino [13] as basis. Finally, Section 6 concludes the paper and gives an outlook on future work.

2 Background

The task of *modeling* swarm intelligence algorithms is understood differently depending on the domain. Typically, modeling swarm intelligence refers to the mapping of swarm behavior from biology to engineering problems. This procedure is roughly sketched in Figure 1, where we observe nature, create a model out of our observations and finally design a swarm algorithm. A very prominent example for a swarm algorithm inspired by nature is the Boids (Bird-oid object = bird like object) model proposed by Reynolds [5]. Only three local rules describe the resulting emergent behavior inspired by the flocking of birds:

- Rule 1: Avoid collision with neighboring birds
- Rule 2: Match the velocity of neighboring birds
- Rule 3: Stay near neighboring birds

In the scope of this paper we do not focus on modeling local rules from biological observation. Our starting point are already existing swarm intelligence algorithms. We rather refer to a model, as used in Model Based System Engineering (MBSE) methodology, using concepts of the Unified Modeling Language (UML)¹ and/or the Systems Modeling Language (SysML)². UML activity diagrams are used to describe the behavior of the CPS swarm created by modeling CPS individual actions (see Section 3.3). On the other, SysML block diagrams are used to model the CPS hardware as proposed in [12], and the interface to the behavior (first level of the proposed two-level hierarchy, see Section 3.2). Currently, SysML is advertised as a language for providing model-based

description of systems. However, during our initial analysis, we found that the SysML language is too general too adequately support the domain-specific needs of CPS swarms. UML and its extension SysML need to be extended and toolled in order to better capture specific aspects, as safety and security for example, of CPS and then CPS swarms. Therefore, we built a CPS swarm profile [12], where we use the SysML language as basis. Specifically, we work with the block definition diagram (BDD) and the internal block diagram (IBD). The modeling tool we used to Modelio³.

2.1 State of the Art

Even with recent advances on swarm intelligence algorithms, the state-of-the-art in modeling such algorithms is rather small. Describing CPS swarm behavior in form of text, pseudo code, or even mathematical formulas is a common way to describe nature-inspired swarm algorithms. This has been done by a number of authors including Parpinelli and Lopes [15], Hassanien and Emary [16], Yang et al. [17], Hamann and Wörn [18], Pinciroli and Beltrame [19], to name but a few. Nevertheless, there is still a lack for a common representation of these algorithms. Furthermore, visualization allows to present many (complicated) relationships at a glance.

For visualizing swarm intelligence algorithms, Finite State Machines (FSMs) have been used by a number of authors, e.g., for modeling the behavior of a mobile robot [20], networked systems [21], and for modeling swarm robotic behaviors [6,8]. The limitation of FSMs is that they describe the behavior on a relatively high level where the CPSs continuously perform several actions within one state of the FSM. To describe the individual actions performed for a specific behavior, Activity diagrams are a useful model. They describe the behavior workflow using concrete actions. This has been done, e.g., by Do et al. [9] and Fernandez-Marquez et al. [22] for designing multi-agent systems. Modelica⁴ is another freely available language which has been used to design CPS. For example, [23] modeled and simulated CPS timing aspect by using it. Note that several commercial multi-discipline modeling softwares, such as MATLAB [24] and AMESim [25], support CPS-design based on MBSE.

Once the behavior is modeled by actions the idea of reusing them to model different behaviors is not far fetched. McLurkin [26] proposed the idea of a swarm behavior library that provides reusable and scalable behaviors that produce predictable group actions. Another approach that has been proposed to reuse swarm behaviors are design patterns [9,22] that are placed in a catalogue. Furthermore, Pitonakova et al. [6] present new ideas on representing control algorithms for robot swarms. They propose a so-called Behaviour-Data Relations Modeling Language (BDRML) that focuses on information exchange between robots including internal and external data structures. They state that currently available modeling languages, like UML, cannot fully express information flow explicitly. From a swarm intelligence point of view,

¹<https://www.omg.org/spec/UML/2.5.1/>

²<https://www.omg.org/spec/SysML/About-SysML/>

³<https://www.modelio.org/>

⁴<https://www.modelica.org/documents/ModelicaSpec34.pdf>

there are still several open points, including the visualization of states, and libraries for swarm algorithms and behaviors. For example, visualization of states is of high importance to observe and analyze the emergent, and thus, complex swarm behavior. Furthermore, we do not want the modeler to learn yet another language, but rather work on existing ones, although they face small variations in their formalization.

2.2 Previous Work

Motivated by the vision of the CPSwarm workbench—a tool chain to model, optimize, simulate and deploy swarms of CPSs—we describe generic libraries of predefined models that describe the CPS as a swarm member, its environment and goal. For more details on the libraries and their implementation, we refer the reader to Schranz et al. [12]. These models can be reused and changed, and new models can be added to the library by the modeler. The models are segmented into three groups of libraries to model a problem:

1. **Swarm member library:** It describes a single CPS in the swarm by using several sub-libraries. They specify the characteristics of a CPS in more detail. This includes local memory, behavior, physical aspects, security, and human interaction.
2. **Environment library:** It describes the environment in which the swarm of CPSs is acting. Several aspects express an environment including 2D/3D map, size, and resolution.
3. **Goal library:** It represents the goal of the CPS swarm in terms of performance using a fitness function.

Such a library structure supports the process of modeling swarms of CPSs. Finally, a swarm can be modeled by a composition of a set of swarm members (see Figure 2). An application example of the libraries is shown in [12].

The focus of this paper is on linking the modeling of the CPS hardware with the modeling of the behavior of each CPS. For the CPS hardware we use the SysML block diagram approach from [12]. This approach is also used for the first level of the proposed hierarchy, the swarm algorithm library extended with design patterns (see Section 3.2). Furthermore, we introduce a second hierarchy, the swarm behavior library, that allows to create behaviors using activity diagrams constituted with individual actions (see Section 3.3).



Figure 2: The swarm architecture.

3 Modeling Swarm Intelligence Algorithms

As we have seen in Section 2, there is no common modeling methodology, technique, or language to visualize swarm algorithms as models in software engineering. Moreover, it is hard to combine different swarm behaviors from different

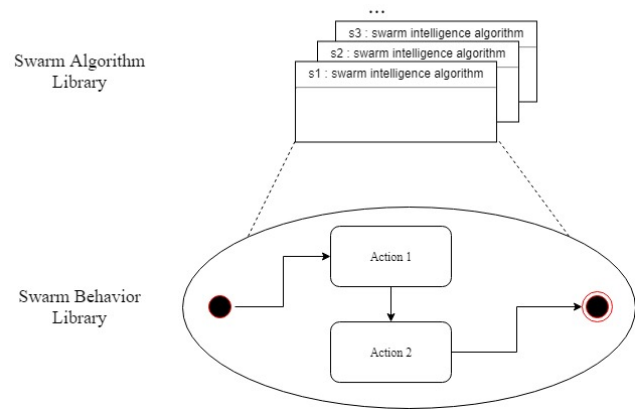


Figure 3: The proposed two-level hierarchy to model swarm algorithms and behaviors.

inspirational sources. To solve this problem, we introduce a common modeling standard for swarm behaviors. The main idea is to have a formulation and a definition of the models that allows to firstly visually represent the local behavior of the CPSs, and secondly generate source code out of the models. The former is covered in this paper while the latter is currently in progress and will be presented in future publications. We propose a representation using a two-level hierarchy as visualized in Figure 3. The first hierarchy models each swarm algorithm available in a swarm algorithm library as a block (SysML) together with several inputs and outputs. The second hierarchy models the individual CPS swarm behavior using activity diagrams (UML).

3.1 Formalization of Models for Swarm Intelligence Algorithms

Each model—belonging to swarm algorithm or swarm behavior library—can be formalized through following three mandatory parts:

1. **Name:** Each model needs to be distinguishable from other models by a unique name. The model name needs to be given in a way that it is associated with the model’s functionality.
2. **Description:** A detailed description is necessary for i) documentation and ii) programming tasks by the software developer. The description is created as parameter of the model with type `string`.
3. **Parameters:** Each model contains a set of property, input, and output parameters.
 - **Property:** Defines a constant parameter of the model.
 - **Input:** Defines an input parameter to the model.
 - **Output:** Defines an output parameter of the model.

Each parameter has the form: `name, type [range]`.

3.2 Swarm Algorithm Library

The swarm algorithm library represents models of existing swarm intelligence algorithms from a high-level view. In the final modeling library, they can be found ready to use, including

- a description of their functionality,
- defined inputs and outputs,
- defined local states (if necessary), and
- deposited code (e.g., Java code).

The designed modeling elements for the swarm algorithms are described and visualized in Table 1, using an adopted version of SysML. Figure 4 shows an exemplary visualization of a swarm algorithm model using the design patterns from Table 1. The model has a specific name (here generically called `s1` of type `Swarm Intelligence Algorithm`), ports for one or more inputs and an outputs, and a local state named `local_position` of type `Position2D`. The description of the model as well as the code are deposited in the model's back-end to not disturb during the modeling process. Further modeling elements from Table 1 can be used to model several interacting swarm algorithms as represented in Figure 3.

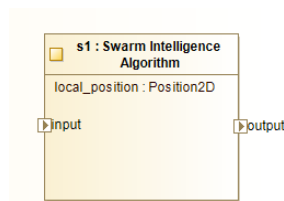


Figure 4: Exemplary model of a swarm algorithm in the swarm algorithm library.

3.3 Swarm Behavior Library

Typically, real-world applications come with needs that cannot be directly modeled with existing nature-inspired swarm behaviors. Therefore, it is useful to have a process that allows the construction of customized or new swarm intelligence algorithms. This is enabled by a library that provides single behaviors extracted out of existing swarm intelligence algorithms in form of actions. A possible collection of such single actions is visualized in Figure 5. From those actions it is possible to construct an activity diagram that describes a certain swarm behavior. Each action has deposited code (e.g., Java code). Furthermore, each action has a number of inputs and outputs that allow to connect with other actions throughout the activity diagram. This indirectly allows to represent the information flow from one action to the other. The activity diagram can be integrated into a swarm algorithm block (see Figure 4) which can be used as model for the behavior of the swarm member. Therein, you can define a unique name, description, number and type of inputs and outputs, and local states (introduced and visualized in Figure 4). For modeling a customized swarm intelligence algorithm in form of an activity diagram, the design patterns described and visualized in Table 2 can be used.

4 Inter/Intra Swarm Communication

One very important building block for modeling swarm intelligence algorithms is the communication interface. It allows the CPSs to interact with each other leading to an emergent behavior of the swarm. Data that are exchanged over the communication interface, are modeled as one or more parameters that are used as inputs and outputs to/from the swarm behavior(s).

In designing swarms of CPSs it is important to consider two types of communication: explicit and implicit. On one hand, explicit communication, as typically used in conventional robotics, requires a communication interface that employs radio technology to interconnect the CPSs [27]. Implicit communication, on the other hand, is achieved by employing the CPS's sensors to interact with other CPSs. This can be done either by directly detecting the state of other CPSs [28] or by indirectly detecting other CPSs' traces in the environment [29]. In the first case, the CPS's internal state must be detectable from its behavior. In the second case, the CPSs manipulate the environment to leave trace for other CPSs. This process is called stigmergy.

Implicit communication typically does not require a communication interface but rather employs the CPS's sensors and actuators whose modeling is described in [12]. In this paper we introduce a model for explicit communication that enriches the modeling of CPS's hardware of [12]. The communication between CPSs is defined in the communication interface. A generic model is given in Figure 6. It consists of several attributes and has the transmitted data as input and the received data as output. The underlying hardware and communication protocol are defined in the technology attribute. The data attribute specifies the type of data that is transmitted over this interface. Finally, transmitter and receiver are specified in the respective attributes. The abstract types are part of the model library.

5 Example: Modeling a Swarm of Spiderinos

To demonstrate the application of the proposed two-level hierarchy architecture of swarm behaviors, we demonstrate the proposed modeling elements on the low-cost robotic⁵ platform Spiderino. Jdeed et al. [13] proposed a CPS, based on

⁵Robots are considered as a CPS subclass in general.

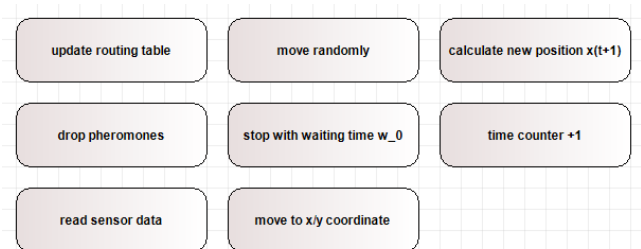


Figure 5: Exemplary selection of available actions.

Table 1: Design patterns to model swarm algorithms.


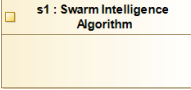









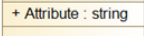
















Name	Icon in the modeling tool	Representation	Description
Part			A part represents the internal instantiation of a given component, i.e., a swarm intelligence algorithm.
Port			A port is used for data flow communications between components.
Connector			A connector represents a specific interaction path between ports.
Provided interface			A provided interfaces of a port exposes services provided by the port owner to its environment.
Required interface			A required interface of a port characterizes services needed by the port owner from its environment.
Attribute			An Attribute is a typed property of its owner.

Table 2: Design patterns to model swarm behaviors using actions in an activity diagram.

Name	Icon in the modeling tool	Representation	Description
State			A state models a notable situation during the life of an object.
Transition			A transition is a directed relationship between two states.
Initial pseudo state			An initial pseudo state represents a default vertex that is the source for a single transition to the default state of a state machine.
Final pseudo state			A final pseudo state signifies that the enclosing region is completed.
Fork pseudo state			A fork pseudo state is a pseudo state that splits an incoming transition into two or more outgoing transitions.
Join pseudo state			A join pseudo state is used to merge several transitions emanating from source in different, orthogonal regions.
Merge pseudo state			A merge pseudo state is a pseudo state that merges conditional branches.
Choice			A choice pseudo state is a pseudo state that realizes a dynamic conditional branch.

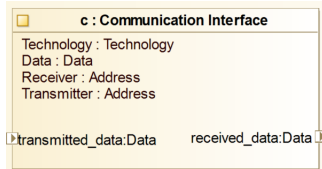


Figure 6: Generic communication interface.

the Hexbug Spider toy for tests and experiments of swarms of CPSs in research and education (see Figure 7). At the moment, the Spiderino offers five CNY70 reflective photo sensors, one GP2D12 distance sensor, a WiFi module, and basic locomotion capabilities (turn head right/left, move forward/backward). The Spiderino platform is an active project and additional components are added steadily.

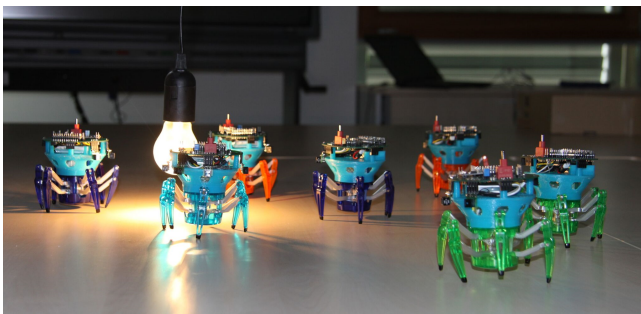


Figure 7: Using a swarm of Spiderinos for research and education.

To model the hardware of the Spiderino, we need a number of models for sensors, actuators (introduced in [12]), and communication (introduced in this paper) that serve as input/output to the behavior:

- rs:CNY70 (5x)
 - Description: “The rs:CNY70 is a photo-reflex optoisolator. It allows to measure reflected light from an integrated IR-LED through an IR-phototransistor.”
 - Output: `light_value`, type: `int`
- dms:GP2D12
 - Description: “The dms:GP2D12 is a IR-distance sensor that measures distances between 10 cm and 80 cm.”
 - Output: `distance`, type: `int`
- l1:Locomotion
 - Description: “The l1:locomotion is a DC-motor, used to turn the head left or right.”
 - Input: `turn_left_right`, type: `int`
- l2:Locomotion
 - Description: “The l2:locomotion is a DC-motor, used to move back and forward.”
 - Input: `move_back_forward`, type: `int`
- c1:Communication Interface

- Description: “The c1:communication interface describes the WiFi module of the Spiderino. It measures the received signal strength `rssi` to recognize other Spiderinos.”

- Properties:

- * `IEEE802.11`, type: `Technology`
- * `MetaData`, type: `Data`
- * `Localhost`, type: `Address`
- * `Broadcast`, type: `Address`

- Input: `transmitted_data`, type: `Data`

- Output: `received_data`, type: `Data`

5.1 Swarm Algorithm

As described, the Spiderino platform comes with a number of sensors, actuators, and a communication module. We model a Spiderino applying the swarm intelligence algorithm BEECLUST [30] as behavior which is inspired by the behavior of young honeybees. BEECLUST is a simple algorithm, mainly using three rules and already applied to underwater vehicles by Bodi et al. in [31]. For the model of the swarm intelligence algorithm we need following modeling elements and formalisms, whereby w_0 indicates the predefined waiting time, w_{max} the maximal waiting time, $w(t)$ the calculated waiting time, $s(t)$ the aggregated light sensor values (from sensors `rs1-rs5`), θ the steepness of the stimulus-response curve, and `rssithr` the threshold for the `rssi` value:

s2:BEECLUST

- Description: “The BEECLUST is used for clustering all swarm members, applying three rules: 1. move randomly, 2. if a Spiderino meets another Spiderino (`rssi > rssithr`), they stop with a certain probability. Their waiting time depends on the local temperature (in our example the light intensity) and w_{max} . The “colder” the local location is, the shorter is the waiting time and the other way around. 3. if a Spiderino hits a wall, it waits for a predefined waiting time w_0 . Pseudo code:

```

move randomly
if Spiderino meets another Spiderino
(rssi > rssithr) then
  stop with waiting time  $w(t)$ 
   $w(t) = (w_{max} s(t)^2) / (s(t)^2 + \theta)$ 
else
  if Spiderino hits a wall
  (distance < 15 cm) then
    stop with waiting time  $w_0$ 
  end if
end if
  
```

- Properties:

- w_0 , type: `int`
- w_{max} , type: `int`
- `rssithr`, type: `int`

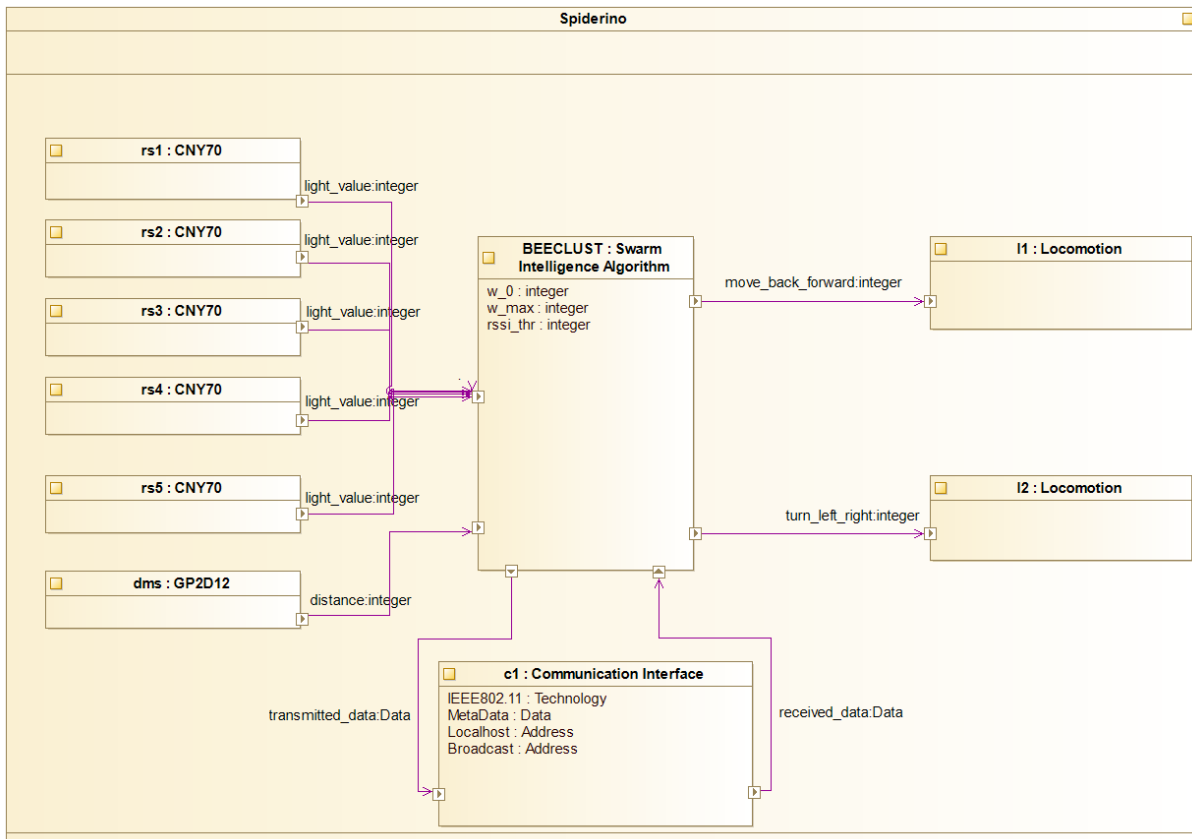


Figure 8: Model of the Spiderino platform using the proposed modeling elements.

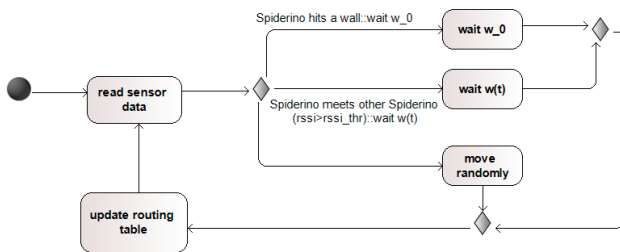


Figure 9: Model of the customized BEECLUST swarm intelligence algorithm using actions in an activity diagram.

- Inputs:
 - light_value, type: int [5]
 - distance, type: int
 - received_data, type: MetaData
- Outputs:
 - move_back_forward, type: int
 - turn_left_right, type: int
 - transmitted_data, type: Data

Applying the CPS modeling concepts presented in Section 2.2 from Schranz et al. [12], the Spiderino model has the form as visualized in Figure 8 aggregating the hardware with the swarm algorithm model using SysML. This model visualizes

a single CPS. The concept of modeling a swarm of individual swarm members was already introduced in Section 2.2, Figure 2.

5.2 Swarm Behavior

As described in Section 3.3, the swarm behavior library can be used to customize a swarm intelligence algorithm by creating an activity diagram from single action elements. A selection of possible actions is visualized in Figure 5.

For the Spiderino example, we decided to customize the BEECLUST algorithm with an additional functionality: each Spiderino shall update an internal list with all x/y coordinates (in a relative coordinate system), whenever it makes a measurement. These coordinates can be read out, transformed to a global coordinate system, and interpreted at the end of a mission. The corresponding activity diagram of the customized BEECLUST with this additional functionality is depicted in Figure 9 and demonstrates the usage of the action library. To gain the desired behavior, we added the action update routing table to the BEECLUST algorithm, described in Section 5.1.

6 Conclusion

Modeling swarm intelligence algorithms for applications of swarms of CPSs from a software engineering perspective has not received much attention in the past. No standards exist, although engineers reach their limits in designing swarms of

CPSs to implement features like adaptability, scalability, and robustness for dynamically changing environments.

In this paper we introduce a new approach for dealing with the complexity in swarms of CPSs. Using SysML and UML as a basis, we propose a two-level hierarchy to model swarm intelligence algorithms: as swarm algorithms or from individual actions as swarm behaviors. In the swarm algorithm library, we present and model swarm intelligence algorithms as they are, the corresponding code is deposited. In the swarm behavior library, the behavior can be created by a sequence of actions in an activity diagram. This allows to create new or customize existing swarm intelligence algorithms to be used as models for the design of swarms of CPSs. We demonstrate the modeling approaches using the Spiderino, a robotic platform with several sensors and actuators, applying the BEECLUST algorithm as example for modeling swarm intelligence algorithms. A video of the Spiderinos using this algorithm can be found online⁶.

In future work, we will further investigate on the theoretical formalisms to extract and position new actions. The idea is to extract them from multiple swarm intelligence algorithms and collect them in an action library. This library drastically reduces the customization effort when designing own swarm intelligence algorithms. As the proposed modeling approach is part of the CPSswarm project, it is also going to be integrated in the CPSswarm workbench—a tool chain to design, optimize, simulate, and deploy a swarm of CPSs. The developed models are available on the Modelio Forge⁷.

Acknowledgment

We thank Arthur Pitman and Midhat Jdeed for their feedback.

The research leading to these results has received funding from the European Union Horizon 2020 research and innovation program under grant agreement No 731946, CPSswarm Project.

References

- [1] E. A. Lee and S. A. Seshia (2016), *Introduction to embedded systems: A cyber-physical systems approach*, MIT Press.
- [2] E. A. Lee (2008), *Cyber physical systems: Design challenges*, in Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, pp. 363–369.
- [3] D. Green, A. Aleti, and J. Garcia (2017), *The nature of nature: Why nature-inspired algorithms work*, Nature-Inspired Computing and Optimization: Theory and Applications (S. Patnaik, X.-S. Yang, and K. Nakamatsu, eds.), pp. 1–27, Springer.
- [4] H. Hamann and T. Schmickl(2012), *Modelling the swarm: Analysing biological and engineered swarm systems*, Mathematical and Computer Modelling of Dynamical Systems, vol. 18, no. 1, pp. 1–12, 2012.
- [5] C. W. Reynolds (1987), *Flocks, herds and schools: A distributed behavioral model*, Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, pp. 25–34.
- [6] L. Pitonakova, R. Crowder, and S. Bullock (2017), *Behaviour-Data Relations Modelling Language for multi-robot control algorithms*, Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 727–732.
- [7] F. Ducatelle, G. A. Di Caro, C. Pinciroli, and L. M. Gambardella (2011), *Self-organized cooperation between robotic swarms*, Swarm Intelligence, vol. 5, no. 2, p. 73.
- [8] O. Soysal and E. Sahin (2005), *Probabilistic aggregation strategies in swarm robotic systems*, Proceedings of the IEEE Swarm Intelligence Symposium, vol. 2005, pp. 325–332.
- [9] T. T. Do, M. Kolp, and A. Pirotte (2003), *Social patterns for designing multi-agent systems*, Proceedings of the 15th International Conference on Software Engineering & Knowledge Engineering, pp. 103–110.
- [10] D. Floreano and F. Mondada(1998), *Hardware solutions for evolutionary robotics*, in Proceedings of the European Workshop on Evolutionary Robotics, pp. 137–151, Springer.
- [11] A. Bagnato, R. K. Bíró, D. Bonino, C. Pastrone, W. Elmenreich, R. Reiners, M. Schranz, and E. Arnautovic (2017), *Designing swarms of cyber-physical systems: The H2020 cpsswarm project: Invited paper*, in Proceedings of the Computing Frontiers Conference, CF'17, pp. 305–312.
- [12] M. Schranz, A. Bagnato, E. Brosse, and W. Elmenreich (2018), *Modelling a cps swarm system: A simple case study*, Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, pp. 1–10.
- [13] M. Jdeed, S. Zhevzyk, F. Steinkellner, and W. Elmenreich (2017), *Spiderino-a low-cost robot for swarm research and educational purposes*, in Proceedings of the 13th Workshop on Intelligent Solutions in Embedded Systems, pp. 35–39.
- [14] H. Ahmed and J. Glasgow (2012), *Swarm intelligence: Concepts, models and applications*, Technical report 2012-585, School of Computing, Queen's University, Ontario, Canada.
- [15] R. S. Parpinelli and H. S. Lopes (2011), *New inspirations in swarm intelligence: a survey*, International Journal of Bio-Inspired Computation, vol. 3, no. 1, pp. 1–16.
- [16] A. E. Hassanien and E. Alamry (2015), *Swarm Intelligence: Principles, Advances, and Applications*, CRC Press.
- [17] X.-S. Yang, S. Deb, Y.-X. Zhao, S. Fong, and X. He (2017), *Swarm intelligence: past, present and future*, Soft Computing, pp. 1–11.

⁶<https://www.youtube.com/watch?v=u5SZUujAsYg>

⁷<http://forge.modelio.org/projects/cpswarm-modelio37/files>

- [18] H. Hamann and H. Wörn (2008), *A framework of space-time continuous models for algorithm design in swarm robotics*, *Swarm Intelligence*, vol. 2, no. 2-4, pp. 209–239.
- [19] C. Pinciroli, A. Lee-Brown, and G. Beltrame (2015), *Buzz: An extensible programming language for self-organizing heterogeneous robot swarms*, arXiv preprint arXiv:1507.05946.
- [20] R. Brooks (1986), *A robust layered control system for a mobile robot*, *IEEE Journal on Robotics and Automation*, vol. 2, no. 1, pp. 14–23.
- [21] W. M. Spears and D. F. Gordon (2000), *Evolving Finite-State Machine Strategies for Protecting Resources*, *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, pp. 166–175.
- [22] J. L. Fernandez-Marquez, G. Di Marzo Serugendo, S. Montagna, M. Viroli, and J. L. Arcos (2013), *Description and composition of bio-inspired design patterns: a complete overview*, *Natural Computing*, vol. 12, no. 1, pp. 43–67.
- [23] H. Dan and H. Elmqvist (2011), *Cyber-physical systems modeling and simulation with modelica*, pp. 1–8.
- [24] M. A. Al Faruque and F. Hourai (2014), *A model-based design of cyber-physical energy systems*, *Proceedings of the 19th Asia and South Pacific Design Automation Conference*, pp. 97–104.
- [25] A. Canedo, M. A. Al Faruque, and J. H. Richter (2014), *Multi-disciplinary integrated design automation tool for automotive cyber-physical systems*, in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 1–2.
- [26] J. D. McLurkin (2004), *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*, PhD thesis, Massachusetts Institute of Technology.
- [27] A. Matsumoto, H. Asama, Y. Ishida, K. Ozaki, and I. Endo (1990), *Communication in the autonomous and decentralized robot system ACTRESS*, *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pp. 835–840.
- [28] M. J. Huber and E. H. Durfee (1995), *Deciding when to commit to action during observation-based coordination*, *Proceedings of the 1st International Conference on Multi-Agent Systems*, pp. 163–170.
- [29] R. Beckers, O. E. Holland, and J.-L. Deneubourg (1994), *From Local Actions To Global Tasks: Stigmergy and Collective Robotics*, *Artificial life IV*, pp. 181–189.
- [30] T. Schmickl, R. Thenius, C. Moeslinger, G. Radspieler, S. Kernbach, M. Szymanski, and K. Crailsheim (2009), *Get in touch: cooperative decision making based on robot-to-robot collisions*, *Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 1, pp. 133–155.
- [31] M. Bodi, C. Möslinger, R. Thenius, and T. Schmickl (2015), *Beeclust used for exploration tasks in autonomous underwater vehicles*, *IFAC-PapersOnLine*, vol. 48, no. 1, pp.819–824.



Automate Your Ada Unit Testing

With VectorCAST/Ada

VectorCAST/Ada is an integrated software test solution that significantly reduces the time, effort, and cost associated with testing Ada software components necessary for validating safety- and mission-critical embedded systems.

- > Complete test-harness construction for unit and integration testing
- > Test execution from GUI or scripts
- > Code coverage analysis
- > Regression Testing
- > Code complexity calculation
- > Automatic test creation based on decision paths
- > User-defined tests for requirements-based testing
- > Test execution trace and playback to assist in debugging
- > Integrations with best of breed requirements traceability tools

More information: www.vector.com/vectorcast

Experience in 40 Years of Teaching Ada

Jean-Pierre Rosen

Adalog, 2 rue du Dr Lombard, 92130 Issy-Les-Moulineaux, France. Email: rosen@adalog.fr

Abstract

In this era of newsgroups, forums, and MOOCs, is there a place for traditional face-to-face training? After years of teaching Ada, we present our experience about what is easy and what requires special emphasis in the various aspects of Ada training. We conclude that “passing the message” is what most requires human presence.

Keywords: Ada, training, software engineering.

Back in 1979, shortly after Green was selected by the DoD, Ichbiah gave the first presentation of this brand new language – preliminary Ada. I attended this presentation, and since at that time I was a teacher in a French engineering school, I immediately returned that presentation to the students. I never ceased teaching Ada ever since then. This paper presents my experience after 40 years of teaching Ada.

1 The students

My experience relates to a quite specialized sample of students: they were all engineers already working in a company, mostly with some years of experience, some of them freshly out of school. Most of them were software engineers, although I received some QA people or project managers sometimes.

None of them were complete beginners as far as programming was concerned. Actually, the knowledge of at least one programming language is a prerequisite to my course. This implies that I assumed that the basics of programming were known (although I had surprises at times). This also meant that quite often, students may have had bad habits that needed to be fixed...

2 Teaching the language

The common language basis known to all students is C, plus varying knowledge of other languages from the C-family (C++, Java, C#...).

The previous knowledge of students evolved over time. Some years ago, many students had used Pascal or Pascal-like languages (at least as a first introductory language), making the syntax look more familiar. More recently, a significant number of students had exposure to VHDL, making the language look even more familiar. Python is often mentioned, but not as frequently as one would expect given the current claimed popularity of this language.

Few students have difficulties with the syntax. Moreover, the syntax differences between Ada and the languages students know are easily overcome, given the excellent

error messages provided by the Gnat compiler for syntax errors.

When showing the syntax of the basic statements, it is important to stress what’s special in Ada (completeness of **case** statements, safety of the **for** loop) and to show that these peculiarities are here on purpose, to serve goals of software engineering. It is also useful to take this opportunity to stress the uniform and consistent design of the language.

Since most students have not been exposed to any block structured language, the notion of being able to declare any construct at any place (like a subprogram or a package within a subprogram) is not easily grasped. It is necessary to explain in great details the nesting of program units, visibility rules, hiding...

Almost all students (except the senior ones) have been trained to object oriented programming with C++ or Java, although none of them really understand where the terms “class” or “method” come from. A bit of methodological introduction, showing the principles of functional vs. object oriented programming is useful, and helps justify the strategy adopted by Ada, since Ada’s OOP model is quite original, and, to be honest, not straightforward. Newcomers are surprised that “class” is not a reserved word! With appropriate explanations, it is reasonably easy to explain the difference between a specific type and a class-wide type.

Exceptions are no more a new concept, and many students had (some) exposure to concurrency.

Note that the teacher should not hesitate to cheat a little bit in places, to provide an usable and easy to understand model, even if it is not the exact truth from a language lawyer point of view. For example, I state (tongue in cheek) that the resolution of integer literals is just a kind of overloading resolution (to avoid having to talk about universal integers), or that a package specification cannot contain any form of body (it can, if it contains a generic instantiation).

It is of course important to make comparison with other languages, not to tell that other languages are bad, but rather that they correspond to different requirements about the very purpose of a programming language. C, for example, was intended to be a “portable assembly language” (and it is very good at that); Ada is intended to keep the programmer away from the machine. I often stress this by saying “C is the best language to program a

computer; *Ada is the best language to develop a software application*¹.

3 Teaching how to use the language

A good Ada course should not teach how to program *with* Ada, but how to program *in* Ada.

At first glance, Ada looks like most other programming languages, and it is easy to use it just like any other language: using only predefined types, using packages just for separate compilation (without any consideration for information hiding), ignoring generics altogether, etc.² The first challenge is to have the students understand that the Ada way of *thinking* is radically different from other languages.

For example, the need for information hiding is not obvious to many. This has to be explained first, then packages come as the tool to enforce information hiding.

The habit of writing specifications without even thinking about the implementation is far from obvious, and must be enforced in the exercises. Students tend to think about implementation first (“how can I do this?”), then write the specification as a way of exporting what they have written. It must be explained that the specification expresses client requirements, and that the body is the implementation of the requirements.

Ada has a rich set of data structures, some with no equivalent in other languages (discriminated types, f.e.). The importance of an appropriate choice of data structures has to be stressed, and more generally the need to *design* types and data structures the same way as algorithms. Although students have no issue with understanding the possibility of defining one’s own types, they have difficulties to put it into practice. The natural trend of many is to do everything with types **Integer** and **Boolean** only.

Finally, it is important to show that the compiler is of great help during the whole coding phase: “the compiler is your friend”. Students tend to write everything, and compile only when the code is complete, producing a discouraging flow of error messages. Explain that the Ada compiler is a helpful companion, willing to tell the errors as early as possible, and that it saves a lot of time to compile often, like immediately after each subprogram is written, even if a lot of code is still missing.

4 Typical student errors (and how to react)

There is a small number of errors that students will make almost systematically in the course of exercises. These are good opportunities to hint on important points of the language.

¹ Please don’t quote me on the first part of this statement without the second part !

² I have seen projects doing this; they didn’t get much gain from using Ada, and spent a huge amount of time fighting the compiler.

Using enumerated types is not natural for many students: for example, in an exercise that involves a state machine, students tend to declare a **Boolean** variable for each state rather than a **Current_State** variable of an appropriate enumerated type.

Hint: ask the student what happens if two states are **True** at the same time (if they assume that it does not happen, ask if they can prove it); show that this cannot happen with an enumerated type.

When faced with a compilation error, students tend to try to make the error disappear, instead of searching for their own design error. Typically, if a student writes:

```
D : Duration ;
begin
```

```
D:= 1;
```

He will get an error message:

```
Expected type Standard.Duration, found an integer type
```

Which he will try to fix by writing:

```
D:= Duration (1);
```

This shows that the student thought “Oh, the compiler wants a **Duration**, let’s convert this to **Duration**”, rather than thinking that **Duration** is a real type, and therefore that the correct fix is:

```
D:= 1.0;
```

Hint: Explain *why* this kind of reaction is wrong: when something doesn’t compile, it is important to understand the problem, not to find a workaround such that the compiler accepts it. In the end, the student should come to think:

```
“Thank you, gentle compiler, for pointing out my
design errors”3
```

Error messages sometimes push the students in the wrong direction. For example, students who want to declare an integer type think that the word “integer” must appear in the declaration and write:

```
type My_Int is Integer range 1..10;
```

Unfortunately, the error message points after the **is** and says “missing **new**”. Students blindly follow the advice and write:

```
type My_Int is new Integer range 1..10;
```

This results in a type derived from Integer, while the correct fix would have been to delete the word **Integer**:

```
type My_Int is range 1..10;
```

Hint: explain that what they wrote makes their type dependent on the definition of **Integer**, while without “**new**

³ Ada is the only language where users are happy to have compilation errors!

Integer” the compiler would choose the most appropriate integer type.

Simple tests are often written as this:

```
if Is_Present = True then ...
```

This may seem just like a slightly redundant formulation (that will be optimized away by the compiler anyway), but it actually reflects a fundamental problem: the student thought “if the variable **Is_Present** contains the value **True**”, not “if my data is present”.

Hint: use this kind of error to stress the need for higher level thinking: the programmer should change his mind from “programming a computer” to “expressing a solution to a problem”.

5 Hard points

5.1 Vocabulary

The compiler has a special lingo. The Ada standard has a precise definition of every technical term, and the compiler is careful to use the appropriate vocabulary. However, terms may not be obvious to the casual user. For example, a common error message is “invalid use of subtype mark in expression or call”; this puzzles the student (*subtype mark?* What’s this?) while it simply means that a type name has been used in place of a variable name – a very common confusion.

Some Ada terms have a different meaning than in other languages. For example, in Ada a *subprogram* is either a *procedure* or a *function*, while in Fortran, a *procedure* is either a *subroutine* or a *function*. An *object* is either a *constant*, a *variable*, or a *formal parameter*... and is not related to object oriented programming. It is up to the teacher to stress these differences to avoid confusion.

5.2 Unlearning

It is a natural move to understand new features of a language by analogy with the similar features of known languages. Unfortunately, this leads to misunderstandings when features look similar, but are in fact quite different. Some concepts (like tasking, or object orientation) are even easier to teach to people *without* corresponding experience, because they don’t have to unlearn the similar looking feature of the other language. It is part of the teacher’s skill to feel when a student is biased in his understanding, and to insist on the differences between the languages.

Some habits are hard to lose, like putting useless parentheses around the condition in an **if** statement! Some students (especially those who know mainly Java or other fully dynamic language) have a hard time understanding that an object can exist simply by being declared, without calling any **new** statement.

5.4 Forget Integer and Float!

The art of Ada programming is in defining appropriate (high level) types. Although the course stresses the need to define appropriate types modeling the problem, students tend to return to the types **Integer** (for integer values) or **Float** (if there is a point in the numbers!), like if these types

were actually the mathematical sets **Z** and **R**, but they are not! During exercises, the teacher should really chase these and make the students think of the very nature of the entities that appear in the program. If two “things” are different in real life, they must not have the same type. For real types, show that other languages are very poor, but that Ada offers a range of possibilities. Take the example of monetary computations, where the use of floating point types is forbidden by law!

5.3 Packages and modularity

When given an exercise that involves providing a package, students tend to hurry into writing the body of the package. One of the benefits of Ada is that you can specify a package, then use the specification (therefore ensuring that the specification meets the needs of the user of the package), and only then turn to the body. The teacher must therefore look after the students to make sure that they always follow these steps:

1. Write the specification of the package, and compile it to check that it is correct Ada.
2. Write the main/test program that uses the package, and compile it to check that the specification implements the requirements.
3. Only then, generate the body skeleton (Gnatstub is instrumental for that), fill it, and try the program.

Things to look after in this process:

- Make sure that bodies are not written too early
- Make sure that once the specification is validated (i.e. the main program is compiled against the package specification), the specification is not changed (except possibly for the private part). Students tend to put declarations in the specification that belong to the body. Explain that validating the specification is like signing a contract, you are not allowed to change a contract after it has been signed!

5.4 Concurrency

Students have varying experience with multi-tasking. Those with some experience generally wrote programs at quite a low level, using pthreads, condition variables, interrupt handlers... The concept of logically independent and concurrent tasks is of a higher level, and students have difficulties to mentally figure that several frames of control execute at the same time. This needs a special mental process, analogous to understanding recursivity. Surprisingly enough, the concept of a rendezvous is not easily understood by those with previous experience, who tend to view the accept statement like a kind of interrupt handler that is activated when a user task needs it rather than as a statement executed according to the path of the owning task. Related to this, some have difficulties understanding that **requeue** terminates the current rendezvous or protected call.

6 Exercises

Computers are provided for the exercises, and students are allowed to bring their own laptops if they prefer. However, I always advise them to pair with a colleague for the hands-on sessions. Working in pairs triggers discussions to find the solution to a problem, avoids being blocked by a small error, and in the end, allows the participants to go farther in the exercise and make it more profitable.

Appropriate exercises are very important. Here are some qualities of a good exercise.

- An exercise must be interesting, in order for the student to be motivated in seeing the result. “Hello world” exercises are a waste of time.
- An exercise must have a first step that everybody should normally complete (to avoid frustration), and further developments for those who achieve the first step quickly, in order to exercise more advanced features.
- An exercise should be targeted to demonstrate a particular feature of the language. For example, I’ve seen no student really understand generics from the slides alone, but after a good exercise, they really grasp how they work.
- An exercise should include a number of traps for students to learn and think about how to solve the problem.⁴

Following these principles, the course offers six exercises:

- A first exercise to get started. It shows the general features of the language, requiring the writing of a package and the definition of some types. The exercise can be solved without using the type **Integer**; it is a good opportunity to watch the students and chase uses of **Integer** instead of user defined types!
- An exercise on generics, showing that once a generic works in one (simple) case, it works in all cases.
- An exercise on access types (a simple linked list), showing it is possible to manipulate pointers without any core dump! It also shows that implicit dereference, which looks weird when first explained, makes a very natural notation.
- An exercise on OOP and tagged types. The course of the exercise is such that a procedure that operates on a class-wide type is written *before* the concrete classes that belong to it. This shows that a subprogram can operate on objects whose type has not yet been written, emphasizing the flexibility of the approach⁵.

⁴ Some students are worried that they have difficulties in doing the exercise. I always respond: “If I give you an exercise and you do it without any problem, it only shows that the exercise was not well chosen”.

⁵ At the cost of compile-time safety, of course.

- An exercise on tasking with rendezvous, where many have difficulties understanding that the **accept** statement is executed sequentially, and not as some kind of call-back.
- An exercise on tasking with protected types and requeues.

In addition, the course includes a short presentation of annex E (distributed systems) with a demo where various clients print to a shared or duplicated print server. Those who had no previous experience with distributed systems find the feature very nice, but those who had to tackle with this kind of development are absolutely stunned by the ease of using and reconfiguring the system!

7 (Other) lessons learned

The *real* new thing that most students discover from Ada training is the art of modeling the problem, of thinking in problem terms and not in terms of machine representation. In the end, most of the difficulties are not with the language, but with the lack of knowledge in basic principles software engineering. An essential part of teaching Ada is not the technical details, but the message of software engineering: that programming should move to higher levels of abstraction, that specifications and implementations should be kept separated, and that defining proper types needs more thinking than writing code.

If you explain these principles, and show how Ada was designed to support them, students are easily convinced on the benefits of using Ada. A common question at the end of the course is: “with all these benefits, how comes that Ada is not more widely used?” Part of the answer is that Ada is not easy to approach in a casual manner: it has to be taught, the benefits explained; it is an industrial tool, and using an industrial tool requires training. Nobody would use an excavator without training, while anybody can pick up a shovel. Of course, it’s less powerful if you have a big trench to dig...

That’s why there is still a future for face-to-face training. All the technological details can be learned with internet tools, but passing the spirit of Ada requires discussion, supervised exercises, and a teacher!

The role of the teacher is especially important for exercises, because it is there, that some notions (generics, rendezvous...) are really understood, and especially through errors and the interaction with the teacher that results. Let’s conclude with a proverb that everyone involved in training should keep in golden letters on his desk:

*“I hear and I forget,
I see and I remember,
I do and I understand”*

Ada-Europe 2019 – Newcomer Experience

Maciej Gajdzica

Solwit SA, Azymutalna 11 80-298 Gdańsk; email: mgajdzica@gmail.com

Abstract

Ada-Europe holds an annual conference on reliable software technologies every year in a different city. Luckily this year it took place in Warsaw, so I was able to attend. It was my first Ada related event and I will share with you my thoughts about the conference and the Ada community as a whole, from the perspective of a newcomer.

Keywords: conference, Ada-Europe, Warsaw.

1 Venue

The main part of the conference containing lectures was held on 12-13 June 2019. A day earlier, on the 11th of June, two workshops took place: “Introduction to Ada” and “Controlling IO devices with Ada”. Also, a day after the conference, on the 14th of June, the WG9 had a meeting about the upcoming standard Ada2020, and a workshop on Cyber-Physical Systems was held. I participated only in the main conference.

The venue was really interesting for technical people, as the conference took place in the Polish Institute of Aviation. From the hallway window we could see an aerodynamic tunnel and the lunchroom was decorated with old jet engines with corresponding descriptions. Did you know that a jet engine burns about 1500 liters of fuel per hour?

2 Community

I attended quite a few programming conferences in Poland and Ada-Europe is much different. My first observation was that the average age of participants was higher than on typical Polish conferences. Many people have tens of years of experience in safety-critical software. Therefore, conversations focused mostly on solid software engineering principles, not on newest trends. Also, Ada-Europe is a rather small conference – there were about 100 participants. But everyone who arrived here was really enthusiastic about Ada and many participants devoted their whole career to promoting it. Unfortunately, Ada didn't receive the popularity it deserves during all these years and, for some reason, it doesn't attract many new adepts, even though Rust recently proved that there is room for languages focusing on safety.

During coffee breaks I also had some time to check stands where conference partners presented their products. Most of Ada use cases presented were for military and avionics. That leads to the conclusion that even though Ada is not that popular, it is still unlikely to be totally abandoned. Especially because no other language, even the mentioned earlier Rust, will be able to provide this level of compile

time checks. This statement was confirmed by some people that I talked to, who use Ada in commercial projects. They also claimed that they couldn't recruit Ada developers. They just recruit developers who are willing to learn Ada. But as a company they still benefit from this. The viewpoint of those trained developers is also really interesting. Almost every new recruited developer follows the same pattern of feelings about the language. At first, they are annoyed with compilation errors. It takes about 5-6 weeks to understand that by failing compilation early they save a lot of time on debugging hard to spot bugs later.

The Ada community includes also a strong representation of universities and researchers. Ada is used in various studies and areas of particular interest are scheduling, concurrency and real-time systems. The conference provides an opportunity to show results of their work.

3 Talks

For two days there were a lot of interesting talks. I picked a few that I enjoyed the most to comment here.

3.1 Contract-based Design and Verification Using SPARK 2014

This presentation caught my attention from the very beginning, after I heard something like "This presentation is based on my experience from a military project which is so secret that I can't say a word about it".

The author compared Test Driven Development (TDD) with Design By Contract (DBC). To keep military secrets, a boiling water project example was used. First requirements were formulated and a generic implementation using Model Based Design was created in the SCADE tool.

When using TDD one must create tests based on requirements and add implementations that pass these tests. To satisfy normative regulations regarding traceability, specific requirements must be mapped to tests and code and review must be done manually.

In the DBC approach, contracts were generated from the model. SPARK was very convenient here as it allowed to validate contracts during compilation. In most languages, asserts are checked at runtime so tests executing all possible paths and validating every assert are needed. When using SPARK, unfulfilled contracts will fail compilation and force developer to address this issue immediately. SPARK contracts were generated from SCADE.

As the tool was certified, manual verification for traceability could be omitted, which saved a lot of time. Of course, full code coverage is still required but when certain errors are now caught during compilation, the whole process requires less effort.

One downside is that automatically generated contracts are unreadable for a mere mortal. Conditional logic placed in contracts may be also duplicated later in implementation code. So, this solution also has some disadvantages.

3.2 Verification and Validation of Launcher Flight Software

This one was about work from the European Space Agency on the Ariane 6 rocket. Once again Model Based Design (MBD) and State Machines were used. But this time code was generated from SysML. In the Ariane 6 project there are 4 types of code:

- Generic libraries (e.g. math)
- Code autogenerated from SysML
- Algorithmic code written by programmers
- Other code written by programmers

Code is written in Ada2012. Generators from SysML models are not certified, so results must be validated manually for traceability.

Manual verification was a recurring topic during the whole conference. Many times, this approach is chosen despite it is tedious, time consuming and error prone. The alternative is to certify a custom generator, which is not worth the effort, or using a customizable commercial tool, which is expensive and doesn't guarantee that it will solve project specific issues.

But back to Ariane 6, MBD allows them to easily use code in multiple environments, including in production, test and various simulations. They have also tools ensuring that simulations generate data similar to tests on real hardware.

3.3 Experience from 40 Years of Teaching Ada

Last talk of the first day was given by Jean-Pierre Rosen. He conducted first trainings even before the language was called Ada. It was codenamed "Green" back then, as one of four proposals for the US military. The talk was light and funny. It was a great choice to place it at the end of a day when everyone is tired, and a complicated technical topic would be hard to follow.

Jean-Pierre had some great insights about how people feel when they learn. I also started recently and know most of these from my own experience. At first there is resistance, since Ada requires a different mindset from all the languages I learned before. Many habits from these languages are more like a burden than a help.

The most frustrating thing at first is a nitpicky compiler. We are used to approaches in which compilation is only a first step, followed by a step of fixing runtime errors. We start to value Ada when we realize that after some initial pain no second step is needed. Therefore, the best approach is to develop in small iterations of write-compile-fix.

Also, verbose syntax is something different from other languages, not to mention some unique constructs like Discriminants.

The conclusion was that nowadays the art of solid Software Engineering is vanishing. We focus on quick results instead

of correctness. Therefore, we have Agile, Rapid Development and short time to market, resulting in maintenance hell later.

3.4 A 2020 View of Ada

Second day keynote speaker, Tucker Taft, gave us a sneak peek of the upcoming Ada2020 standard. The main emphasis was put on parallel programming, which was also a topic of the first day keynote on the OpenMP standard.

Another big change will be the addition of some functional programming elements. It may sound silly, but Ada will show some similarities to Python. For example, we will have list comprehension.

It was a great opportunity to learn how the Ada working group proceeds and how new features evolve before they are accepted.

3.5 RCLAda or Bringing Ada to the Robotic Operating System

This was one of the last talks of the second day. I worked on one side project in ROS, so I liked the idea of using Ada in mobile robotics. Of course, this use case is far from the highly reliable safety-critical systems that were focused on almost every presentation at Ada-Europe. But this project could be an opportunity to promote Ada among students. You can follow progress of this project on GitHub [1].

4 Sponsor Presentations

I also mention talks given by sponsor representatives. My previous experience with this kind of presentations was quite bad. They are usually non-technical, going into details of products that I don't use and often biased. But this time I was surprised. They provided general best practices to handle common developer problems, showed interesting statistics and shared feedback from clients.

The one I enjoyed the most was from VectorCast. Their flagship product is intended to speed up unit tests, but I'm not a fan of it to say the least. Nonetheless, their presentation was top notch. It presented common pitfalls when writing tests and contained many words of wisdom. My favourite one was that most developers write tests based on existing code instead of requirements. It's like a university teacher checking the exam based on student answers instead of correct answers.

5 Summary

I really enjoyed Ada-Europe. Discussed topics couldn't be found on any other IT conference in Poland. It's not only because Ada is a niche language, but more because problems that arise when trying to assure the highest safety and reliability levels are unfortunately not a big concern in most non-safety related IT projects. The Ada-Europe conference could be an eye opener for these people and, even if they don't work on such critical systems daily, they would realize the importance of software quality.

References

- [1] <https://github.com/ada-ros/rclada>.



Join Ada-Europe!

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

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

To apply for membership, visit our web page at



<http://www.ada-europe.org/join>

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch