ADA USER JOURNAL

Volume 35
Number 1
March 2014

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

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

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

Proceedings and Special Issues

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

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

News and Product Announcements

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

Reprinted Articles

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

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

Editorial

In this editorial I would like to note to our readers the 19th International Conference on Reliable Software Technologies – Ada-Europe 2014, which will take place 23-27 of June 2014 in the heart of Paris, France. The advance program of the conference, which can be found in the forthcoming events section of the issue, illustrates that it will be a remarkable event, both due to its rich program and beautiful location in central Paris.

On the program of the conference, I would like to highlight the four sessions of technical papers and two sessions of industrial presentations, as well as the special featured keynote talks by Robert Lainé, on the lessons learned in space projects leadership at ESA and EADS; Mohamed Shawky, on futuristic work on intelligent transportation systems; and Alun Foster, to explore the results and objectives of the large Artemis and ECSEL European R&D programmes. Another highlight is the retrospective session on the occasion of the 20th anniversary of GNAT, the open source Ada compiler.

The conference week will also encompass ten tutorials (with topics including parallel programming, developing real-time and mixed criticality systems, high-integrity object oriented programming, Ada 2012 contracts, Model driven engineering, testing, robotics, and SPARK 2014), and three workshops on "Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering"; "Mixed Criticality Systems: Challenges of Mixed Criticality Approaches and Benefits for the Industry"; and "Ada 2012: le point sur le langage (Ada 2012: Assessing the Language)".

On the location, the conference will take place at the ECE School, located near the Tour Eiffel, in the heart of Paris. And the social program includes a conference banquet held aboard a boat, cruising along the Seine, a wonderful opportunity to sightsee some of the most important Parisian monuments. Finally, although not listed in the announcement, it may happen that the new book of John Barnes, Programming in Ada 2012, will be on display in Paris. A full week indeed!

Continuing with the forthcoming events, on the other side of the Atlantic, the SIGAda HILT 2014 conference will be co-located with the SIGPLAN SPLASH conference, in Portland, Oregon, October 18-21, 2014. Registration will allow attending both conferences.

After the usual news digest and calendar and events sections, the technical part of this issue of the Journal also provides a rich set of contents. It starts with an article by Karen Sargsyan, of Academia Sinica, Taiwan, on the use of Coq, Ada and SPARK for bioinformatics applications. After that, Christoph Grein, from Germany, provides an analysis on the use of aspects by GNAT to specify properties of physical units. In the following paper, a group of authors from Sweden discuss the means for lightweight and pragmatic qualification of tools as an alternative to regular certification processes. The final paper, by authors from Universidad Politécnica de Madrid, Spain, provides an overview and analysis of spacecraft on-board software development, instantiated in the UPMSat-2 satellite software.

The issue also continues the publication of articles on the Rationale for SPARK 2014, with an article on the use of Formal Containers, based on contributions by Claire Dross of AdaCore, France. Finally, the Ada Gems section presents a two-part tutorial on Multicore Maze Solving by Pat Rogers, of AdaCore, USA.

Luís Miguel Pinho
Porto
March 2014
Email: AUJ_Editor@Ada-Europe.org

Quarterly News Digest

Jacob Sparre Andersen

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

Contents

Ada Rationale 2012

Errata for Printed Version of Ada 2012 Rationale

– section/page: 2.5/59

the static predicate for subtype Double should be

Double in 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40;

– section/page: 2.5/59

the static predicate for subtype Treble should be

Treble in 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 | 33 | 36 | 39 | 42 | 45 | 48 | 51 | 54 | 57 | 60;

– section/page: 4.6/96

in para starting "The other small change ..."

"subtype give in the profile" should be "subtype given in the profile"

– section/page: 6.3/122

towards end of para starting There are other ...

(rather than **is**) should be (rather than **in**)

section/page: 6.3/124

last line

"one using is" should be "one using in"

– section/page: 6.4/133

top of page

function Reverse should be
function Reverse_List and at end

– section/page: 6.4/133

near bottom of page
"to named access types" should be
"to named general access types"

Declaration of Class_Acc should be
type Class_Acc **is access all** T'Class;
-- *named general access type*

– section/page: 7.2/149

last para
"done be functions" should be
"done by functions"

– section/page: 7.2/151

about two thirds down in list of functions
"function Is_Other ..." should be ..
" function Is_Other_Format ..."

– section/page: 7.5/156

middle of page
"Ada.Wide_Strings.Equal_Case" should be "Ada.Strings.Wide_Equal_Case_"

– section/page: 8.4/171

second displayed fragment of program
"**for** C **in** The_Tree.Iterate(S) **loop**" should be

"**for** C **in** The_Tree.Iterate_Subtree(S) **loop**"

– section/page: 8.4/176

in function Eval, the declaration of L, R: Float needs semicolon thus
L, R, Float;

– section/page: 8.6/184

in with function Get_Priority and Before, no need for space before colon
(my style, so can ignore)

– section/page: 8.6/186

in para starting "As a final example"
"They might included usual ..." should be
"They might include usual ..."

dated 10 Feb 2014

Ada-related Events

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

Ada in Denmark: Birthday of Ada - Talk on Cyclomatic Complexity

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue, 10 Dec 2013 10:43:53 +0100
Subject: Ada in Denmark: Birthday of Ada - Talk on cyclomatic complexity
Newsgroups: comp.lang.ada

Tonight at 17:30 Ada in Denmark will meet at:

Responsum K/S

Farum Gydevej 87

3520 Farum

https://plus.google.com/u/0/events/c5i8f6pga7apck08ss0nuqdicjk

The meeting is also open to non-members. Please let me or Thomas Løcke (+45 60 43 19 92) know if you intend to attend the meeting.

Thomas Pedersen (who is an intern at AdaHeads K/S at the moment) will give a short talk based on McCabe's cyclomatic complexity paper (http://www.literateprogramming.com/mccabe.pdf).

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Wed, 11 Dec 2013 12:32:47 +0100
Subject: Re: Ada in Denmark: Birthday of Ada - Talk on cyclomatic complexity
Newsgroups: comp.lang.ada

A short report from the Ada in Denmark meeting yesterday:

We started the evening with Thomas Pedersen's presentation of the principles in McCabe's cyclomatic complexity measure. Following up on the presentation we discussed the experimental evidence available to identify which (cyclomatic) complexities are acceptable. Per Dalgas was kind enough to provide a "problematic" legacy subprogram in Ada which we found to have a cyclomatic complexity of 9. McCabe himself indicates 10 as an upper limit and references some provided examples of problematic subprograms with complexities of 16 and above (IIRC). I have done a search for Ada subprograms with complexity above 10 in my published Ada projects. All the examples I have found so far were straight transcriptions from old FORTRAN sources.

The discussion continued on the subject of measuring source text quality. Per Dalgas posed the challenge of how we get more software developers to _use_ the available metrics. One option which came up was to run quality metrics automatically on the source texts on public version control repository services (such as Bitbucket, Github and Sourceforge). This has the benefit of being something which initially can be implemented as an independent service and only later be pushed to the actual

source hosting services (if they want it). But would implementing such a measure make a difference? Will the developers worry about it? Will the users of software use it as selection criteria if they get the possibility?

Then SQALE came up as an example of a "combined" source text quality measure (I had the pleasure of attending Jean-Pierre Rosen's talk on SQALE at Ada Europe 2011), but it appears that only few tools exist (and none of them Open Source) and it wasn't obvious which languages the tools can analyse.

In another branch of the discussion we wondered if we could get big buyers/tenderers of software to require SQALE measures with constraints on what are acceptable levels as a part of software delivery contracts.

Doom 3 in Ada at FOSDEM

From: Justin Squirek <jsquirek1@student.gsu.edu>
Date: Sat Feb 1 2014
Subject: Building a cross platform media layer based on Doom 3
URL: https://fosdem.org/2014/schedule/event/doom3_cross_platform/

Resolving API dependencies and Id Tech 4 modding

A short talk on common programming APIs used by games as well as creating simple Doom 3 levels and menus - with examples from current programming projects AdaDoom3 and a Neotokyo tribute modification.

Links:

- Main code repository: https://github.com/AdaDoom3/AdaDoom3
- Doom 3 Modification: https://github.com/AdaDoom3/NeotokyoMod

[It was not only in the Ada DevRoom that Ada applications were presented. —sparre]

[See also “First Person Shooter”, AUJ 34-2, p. 73. —sparre]

From: Justin Squirek <jsquirek1@student.gsu.edu>
Date: Sun Feb 16 2014
Subject: Ada Programming
URL: https://plus.google.com/104228556547212920341/posts/KaJjfqM5wYf

I uploaded my Media layer/AdaDoom3 FOSDEM presentation slides to github if anyone wants to have a look:

https://github.com/AdaDoom3/AdaDoom3/blob/master/FOSDEM%20Presentation.pdf?raw=true

DragonLace at FOSDEM'2014

From: John Marino <dragonlace.cla@marino.st>
Date: Mon Feb 3 2014
Subject: DragonLace at FOSDEM'14
URL: http://www.dragonlace.net/posts/DragonLace_at_FOSDEM__39__14/

Once again the Ada language merited a large room at FOSDEM. On February 1, a series of talks took place in the Ada Devroom, ending with a presentation about the DragonLace project and future plans.

All of presentations were video recorded, and the presentations have been uploaded to the Ada-Belgium site. The DragonLace Presentation is available in PDF and ODP formats. It discusses the latest state of Ports and Pkgsrc support as well as some potential future work.

[See also “New and Updated FreeBSD Ports”, AUJ 34-4, p. 204. —sparre]

FOSDEM Presentations On-line

From: Dirk Craeynest <dirk@cs.kuleuven.be>
Date: Sat, 15 Feb 2014 21:10:42 +0000
Subject: FOSDEM 2014 - Presentations Ada Developer Room on-line
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc

** All presentations available on-line **

Ada Developer Room at FOSDEM 2014

(Ada at the Free and Open Source Software Developers' European Meeting)

Saturday 1 February 2014

Université Libre de Bruxelles (U.L.B.), Solbosch Campus, Room K.4.601

Avenue Franklin D. Roosevelt Laan 50, B-1050 Brussels, Belgium

Organized in cooperation with Ada-Europe

<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/14/140201-fosdem.html>

All presentations from our 5th Ada Developer Room, held at FOSDEM 2014 in Brussels recently, are available on the Ada-Belgium web site.

- “Welcome” by Dirk Craeynest - Ada-Belgium
- “An Introduction to Ada for Beginning and Experienced Programmers” by Jean-Pierre Rosen - Adalog
- “Ada Task Pools: Multithreading Made Easy” by Ludovic Brenta - Debian Project
- “SPARK 2014: Hybrid Verification using Proofs and Tests” by José F. Ruiz - AdaCore
- “Contract Based Programming in Ada 2012” by Jacob Sparre Andersen - JSA Research & Innovation
- “Formal Verification with Ada 2012: a Very Simple Case Study” by Didier Willame - Argonauts-IT
- “Speedup and Quality Up with Ada Tasking (Solving polynomial systems faster and better on multicore computers with PHCpack)” by Jan Verschelde - University of Illinois at Chicago
- “Safer Web Servers with Ada and AWS” by Jean-Pierre Rosen - Adalog
- “Ada in Fedora Linux” by Pavel Zhukov - Fedora Project
- “Ada in Debian Linux” by Ludovic Brenta - Debian Project
- “Ada in *BSD” by John Marino - FreeBSD Project

Presentation abstracts, copies of slides, speakers bios, pointers to relevant information, links to other sites, etc., are all available on the Ada-Belgium site at:

<http://www.cs.kuleuven.be/~dirk/ada-belgium/events/14/140201-fosdem.html>

Shortly, some pictures and video registrations will be posted as well. If you have additional pictures or other material you would like to share, or know someone who does, then please contact me.

Finally, thanks once more to all presenters for their work and collaboration, thanks to the many participants for their interest, and thanks to everyone for another nice experience!

Ada Course in Carlsbad, California

From: Ed Colbert <colbert@abssw.com>
Date: Fri, 21 Feb 2014 23:30:00 -0800
Subject: [Announcing] Public Ada Courses 24-28 March 2014 in Carlsbad CA
Newsgroups: comp.lang.ada

Absolute Software will be holding a public Ada course during the week of 24 March in Carlsbad, CA. You can find a full description and registration form on our web-site, www.abssw.com. Click the Public Courses button in the left margin. (We also offer courses on real-time system design, software architecture-based development, safety-critical development, object- oriented methods, and other object-oriented languages.)

If there is anything you'd like to discuss, please call, write, or send me E-mail.

Video Recordings from FOSDEM

From: The FOSDEM Video Team
Date: Mon Feb 24 2014
Subject: Index of /2014/K4601/Saturday
URL: http://video.fosdem.org/2014/K4601/Saturday/

[Currently available video recordings from the Ada DevRoom at FOSDEM 2014: —sparre]

- Welcome
- Introduction to Ada for Beginning and Experienced Programmers
- Ada Task Pools Multithreading Made Easy
- SPARK 2014 Hybrid Verification using Proofs and Tests
- Contract Based Programming in Ada 2012
- Formal Verification with Ada 2012 a Very Simple Case Study

Ada-Europe 2014 in Paris

From: Ada-France
Date: Tue Feb 25 2014
Subject: Registration
URL: http://ada-europe2014.org/registration1.html

Registration to the conference will open on March 10th. See you then!

[It is soon time to sign up for Ada-Europe 2014. —sparre]

Ada-related Resources

Writing Spreadsheets

From: Serge Mosin <svmosin@gmail.com>
Date: Sat, 16 Nov 2013 20:45:12 -0800
Subject: Ada Spreadsheet output
Newsgroups: comp.lang.ada

I wish to know, if there are some Ada libraries for spreadsheet output, preferably OpenOffice. I mean the ability to create a spreadsheet file and write there/read it from the Ada program. The method should be fast enough, because big amount of data is supposed to be transferred, so launching OpenOffice and controlling output through it is not the solution.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 16 Nov 2013 22:44:14 -0700
Subject: Re: Ada Spreadsheet output
Newsgroups: comp.lang.ada

> [...]

There's Excel Writer for output:

http://excel-writer.sourceforge.net/

I'm not aware of anything for OpenOffice, nor for reading Excel files. This thread from 2004 (http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2004-10/0299.html) suggests using ODBC to read Excel files.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 20 Nov 2013 03:03:45 -0600
Subject: Re: Ada Spreadsheet output
Newsgroups: comp.lang.ada

[...]

A CSV file has characters in cells. Each row of cells is separated by a newline, and within each row, each cell is separated by a comma (or any other delimiting character). http://en.wikipedia.org/wiki/Comma-separated_values

The interpretation of the characters in each cell is up to the spreadsheet program, so the structure can be as complex as needed. In particular, if you want commas in a cell, the spreadsheet has to define a quoting syntax. Usually it's easier to use a different delimiter, such as tab, that is not needed in the cell data.

For example, I just wrote a CSV file in Emacs, by typing the characters:

1,0

2,=A1-B1

3,=A2-B2

Then I opened that file in OpenOffice Calc; the cells starting with "=" were interpreted as formulas, and the spreadsheet display is:

1 0

2 1

3 1

When I examine cell B2, it has the formula =A1-B1. I can then save it in any format OpenOffice supports. However, if I save it as a CSV, the file has numbers, not the formulas. That makes sense, because you might be exporting the results to a plotting program. But it would also make sense to have an option to export the formulas.

So as far as I can see, any formula in an OpenOffice spreadsheet can be imported into OpenOffice via a CSV file.

What else do you need?

The standard OpenOffice file format of ODS supports metadata; cell formatting styles, color, everything else you can set via the toolbars. That data is not representable in the CSV. So if you need to import that data, you'd have to write the ODS format directly. That's a zip of xml files, and there are Ada libraries that do both, so there is code you could build on.

The representation of the single cell B3 in the ODS content.xml file looks like this:

<table:table-cell
table:formula="of:=[.A2]-[.B2]"
office:value-type="float"
office:value="1">

That should be easy to generate with the GNAT XML/Ada library.

From: Maxim Reznik <reznikmm@gmail.com>
Date: Tue, 19 Nov 2013 03:09:46 -0800
Subject: Re: Ada Spreadsheet output
Newsgroups: comp.lang.ada

I saw some support of Open Document Format (aka OpenOffice files) in Matreshka project.

http://forge.ada-ru.org/matreshka/wiki/ODF

I don't know actual status however. You can try download or contact author if you are interested.

Experimental Continuous Integration System for Open Source Projects

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Sun Dec 1 2013
Subject: build.ada-language.com status
URL: http://tero.stronglytyped.org/buildada-languagecom-status.html

Jenkins update broke the distributed builds, so other than Debian 7 builds at http://build.ada-language.com/ are not updating at the moment.

I am waiting for Jenkins fix and also looking for alternative build systems, but that might take a while.

[See also “Experimental Continuous Integration System for Open Source Projects”, AUJ 34-3, p. 137. —sparre]

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue Feb 18 08:25:00 CET 2014
IRC-channel: #Ada
IRC-network: irc.freenode.net

08:25 < tkoskine> sparre: I fixed build.ada-language.com yesterday. (Found finally time to build my own version of build publisher plugin with the fix.)

Repositories of Open Source Software

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue Feb 18 2014
Subject: Repositories of Open Source software
To: Ada User Journal

AdaForge: 8 repositories [1]

Bitbucket: 103 repositories [2]
16 developers [2]

Codelabs: 18 repositories [3]

GitHub: 489 repositories [4]
130 developers [5]

Rosetta Code: 575 examples [6]
26 developers [7]

Sourceforge: 224 repositories [8]

[1] http://forge.ada-ru.org/adaforge [2] http://edb.jacob-sparre.dk/Ada/on_bitbucket

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

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

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

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

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

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

[See also “Repositories of Open Source Software”, AUJ 34-4, p. 198. —sparre]

Ada-related Tools

Deepend

From: Brad Moore <brad.moore@shaw.ca>
Date: Sun Jun 23 2013
Subject: Deepend
URL: http://sourceforge.net/projects/deepend/

Deepend is a storage pool with subpool capabilities for Ada 2005.

- Fixed issues preventing compilation of Ada 2012 version for GNAT GPL 2013.
- Removed workarounds for GNAT compiler bugs for the Ada 2012 version that were fixed in the GNAT GPL 2013 version of the compiler.

[See also “Deepend”, AUJ 33-3, p. 146. —sparre]

OpenGLAda and OpenCLAda

From: Felix Krause <usenet@flyx.org>
Date: Thu, 14 Nov 2013 22:37:34 +0100
Subject: ANN: OpenGLAda 0.3 and OpenCLAda 0.1 released
Newsgroups: comp.lang.ada

These are quite thick Ada bindings for the OpenGL and OpenCL APIs. OpenGLAda also contains additional wrappers for GLFW 2/3, SOIL and FTGL.

The two wrappers are interoperable (you can use the cl_gl extension of OpenCL to transfer data between OpenGLAda and OpenCLAda).

Some online documentation, including overviews of what the wrappers add to the bare C APIs, is available at

- http://flyx.github.io/OpenGLAda/
- http://flyx.github.io/OpenCLAda/

Releases are available as tags of the GitHub repositories:

- https://github.com/flyx/OpenGLAda/tags
- https://github.com/flyx/OpenCLAda/tags

AWS and the TechEmpower Framework Benchmark Project

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 18 Nov 2013 18:59:04 -0800
Subject: AWS Entry into the TechEmpower Framework Benchmark Project
Newsgroups: comp.lang.ada

TechEmpower is Benchmarking different frameworks for web-development (http://www.techempower.com/benchmarks/#section=motivation&hw=i7&test=json).

There was no entry for AWS, or any other Ada showing (I know there's matreshka, and IIRC several Ada/CGI bindings), so I coded one up.

Unfortunately there is no DB-functionality in it right now, as I couldn't get the ODBC to work, nor could I find a working InterBase (or FireBird) binding for Ada.

https://github.com/OneWingedShark/web-framework-test

From: Graham Stark <graham.stark@virtual-worlds.biz>
Date: Thu, 28 Nov 2013 04:32:04 -0800
Subject: Re: AWS Entry into the TechEmpower Framework Benchmark Project
Newsgroups: comp.lang.ada

> [...]

That's interesting. I had a go at the first two database tests using the Posgres stuff in GnatColl and a little database code generator I wrote a couple of years ago (http://virtual-worlds-research.com/downloads/mill). The code is here:

https://github.com/grahamstark/techempower/

I suspect my version is much slower than their peak performers, though I've just tested it locally. They have a peak on the first database test of 105,939 whereas I'm struggling to get above 2,000; I'm guessing that's still under 10,000 on their hardware.

Embedded Web Server

From: Simon Wright <simon@pushface.org>
Date: Thu, 21 Nov 2013 21:02:54 +0000
Subject: Embedded Web Server 20131121
Newsgroups: comp.lang.ada

This minor release of EWS makes no functional changes, but includes support for building on Windows without Cygwin and INSTALL instructions.

https://sourceforge.net/projects/embed-web-srvr/files/ews-20131121/

[See also “Embedded Web Server”, AUJ 34-4, p. 201. —sparre]

Markup Templates Engine

From: Vadim Godunko <vgodunko@gmail.com>
Date: Wed Nov 27 2013
Subject: Matreshka Ada Framework - Markup Templates Engine
URL: http://forge.ada-ru.org/matreshka/wiki/XML/Templates

Markup Templates Engine reads XML template documents and generates XML or HTML5/XHTML5 documents.

[...]

[See also “Matreshka”, AUJ 34-3, p. 139. —sparre]

Qt5Ada

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Fri, 13 Dec 2013 03:29:30 -0800
Subject: Announce: Qt5Ada version 5.2.0 release 13/12/2013 free edition
Newsgroups: comp.lang.ada

Qt5Ada is Ada-2012 port to Qt5 framework based on Qt 5.2.0 final Qt5ada version 5.2.0 open source and qt5c.dll (libqt5c.so) built with Microsoft Visual Studio 2012 in Windows and gcc x86-64 in Linux.

Package tested with GNAT-GPL-2012 Ada compiler in Windows 32bit and 64bit and Linux x86-64 Debian 7.

It supports GUI, SQL, Multimedia, Web, Network, Touch devices, Sensors and many others things.

Qt5Ada for Windows and Linux (Unix) is available from

http://users1.jabry.com/adastudio/index.html

My configuration script to build Qt 5.2 is: configure -opensource -release -nomake tests -opengl desktop -icu -plugin-sql-mysql -plugin-sql-odbc -plugin-sql-oci -prefix "e:/Qt/5.2"

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

[See also “Qt5Ada”, AUJ 34-3, p. 140. —sparre]

GNATColl.SQL Object-Relational Mapping

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu, 19 Dec 2013 15:53:12 +0100
Subject: Experiences with GNATColl.SQL ORM?
Newsgroups: comp.lang.ada

Can somebody report on their experiences with the Object-Relational Mapping (ORM) layer in GNATColl.SQL?

We are using plain GNATColl.SQL at AdaHeads, and I am experimenting with it for some other projects, but I would like to use ORM and not the untyped GNATColl.SQL interface.

I have tried to run the "gnatcoll_db2ada" tool on our data model for Alice, but it fails with an internal error:

A database error occurred, please try again...

Exception name: CONSTRAINT_ERROR

Message: gnatcoll-sql-inspect.adb:174 access check failed

This does not make me very confident of the quality of the ORM layer of GNATColl.SQL. :-(

Are there any alternatives out there? (The requirements are that the persistence backend shouldn't be specific to the tool and that the storage access should be strongly typed.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 19 Dec 2013 16:20:47 +0100
Subject: Re: Experiences with GNATColl.SQL ORM?
Newsgroups: comp.lang.ada

> A database error occurred, please try again...

> Exception name: CONSTRAINT_ERROR

> Message: gnatcoll-sql-inspect.adb:174 access check failed

A strayed accessibility check? Maybe, replacing Access with Unchecked_Access would cure it.

> [...] alternatives [...]

A different approach:

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

Supported back-ends are ODBC and SQLite3. APQ was dropped due to lack of maintenance

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 19 Dec 2013 19:25:58 +0100
Subject: Re: Experiences with GNATColl.SQL ORM?
Newsgroups: comp.lang.ada

> Reading the documentation, it seems like the actual storage is as strings, and not as types equivalent to those used on the Ada side. Is that correct?

Yes, it is a standard OO serialize/deserialize schema. Objects are stored as blobs. Object's types and dependencies are stored independently.

Request: B-tree Library

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 16 Jan 2014 17:12:08 +0100
Subject: Single file resident B-tree library?
Newsgroups: comp.lang.ada

Is there an Ada implementation of (under a commercially-friendly license)?

Something like this:

http://people.csail.mit.edu/jaffer/wb/C-Interface.html#C-Interface

with an ability to scan adjacent keys (ranges of keys). Yet better to be able to attach some data to non-leaf nodes.

P.S. I know that SQLite3 uses B+ trees, but it has an SQL interface, while I need something more light-weight. Berkeley DB does not support scanning, right?

TASH

From: Simon Wright <simon@pushface.org>
Date: Sun, 19 Jan 2014 18:08:18 +0000
Subject: ANN: TASH 8.6-1 20140118
Newsgroups: comp.lang.ada

This release is now available at Sourceforge[1].

Changes in 20140118

A new package Tcl.Async supports writing Tcl variables from Ada. This is especially important if the Ada code isn't running in the same thread as the Tcl interpreter.

You can use the 'trace' facility in Tcl to detect when such a write has taken place.

The build scripts recognise XQuartz in Mac OS X >= Mountain Lion.

Question for users

The thin binding is full of code like:

```
type Tcl_Interp_Rec (<>) is private;
type Tcl_Interp is access all
        Tcl_Interp_Rec;
pragma Convention (C, Tcl_Interp);
Null_Tcl_Interp : constant Tcl_Interp :=
                                    null;
function Is_Null (Ptr : in Tcl_Interp)
        return Boolean;
```

the last 2 lines of which are a holdover from the original C2Ada-generated binding. I'd like to get rid of them. Any problems?

[1] https://sourceforge.net/projects/tcladashell/files/source/20140118/

[See also "Tcl/Tk", AUJ 33-4, p. 237. —sparre]

Matreshka

From: Vadim Godunko <vgodunko@gmail.com>
Date: Tue, 21 Jan 2014 01:00:34 -0800
Subject: ANN: Matreshka 0.6.0
Newsgroups: comp.lang.ada

We are pleased to announce new release of Matreshka framework. New features:

- markup template processor to process XML/XHTML documents
- optimizing HTML5 writer to generate HTML5 documents (mostly for use with template processor)
- support package to help to process XML namespaces in applications
- binding to GCC's intrinsic functions to use SIMD instructions in Ada code without use of assembler code
- GDB plugin to output content of Universal_String in user friendly form
- support for ARM/Linux, FreeBSD, Windows 64-bit
- update to Unicode 6.3.0 and CLDR 24

for complete list of fixed bugs and new features see

http://forge.ada-ru.org/matreshka/wiki/ReleaseNotes/0.6

Matreshka can be downloaded as source code archive or as binary package for some operating systems from

http://forge.ada-ru.org/matreshka/wiki/Download

[See also "Matreshka", AUJ 34-3, p. 139. —sparre]

Turbo Pascal 7 Emulation

From: Pascal <p.p14@orange.fr>
Date: Wed, 29 Jan 2014 20:44:37 +0100
Subject: [gtkada] [ANN] TP7 emulation V3.0 with GTK-Ada.
To: <gtkada@lists.adacore.com>

Hello, here is TP7-Ada based now on GTKAda 3.4.

Other changes are (versus version 2.7):

- implementation of ShowMouse and HideMouse
- bug fix in Delay1.

TP7-Ada is a port of Turbo Pascal libraries in Ada with GTK-Ada support.

Moreover it can be used as a basic multi-purpose library for simple text or graphic stuff with GTK-Ada.

See screen captures on:

http://blady.pagesperso-orange.fr/tp7ada.html

The complete code is here:

http://sourceforge.net/p/p2ada/code/HEAD/tree/extras/tp7ada/current

See also (in French):

http://blady.pagesperso-orange.fr/creations.html#ada_tp7

All TP7 features are not completely functional, see current status:

http://sourceforge.net/p/p2ada/code/HEAD/tree/extras/tp7ada/current/TurboPascal7.0-Ada.html

All Pascal source codes were translated in Ada with P2Ada translator:

http://sourceforge.net/projects/p2ada/

Feel free to send any feedback.

PS for Mac users: XAdaLib 2013 with GTKAda 3.4 binaries have been upload again on SourceForge due to an upload issue.

http://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2013-mavericks/

[See also "Turbo Pascal 7 emulation", AUJ 34-1, p. 7. —sparre]

Emacs Ada Mode

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Tue, 04 Feb 2014 09:01:46 -0600
Subject: Emacs Ada mode 5.0.1 available in Gnu ELPA
Newsgroups: comp.lang.ada

Emacs Ada mode 5.0.1 is now available in Gnu ELPA.

It requires Emacs 24.3; I'm working on backporting to Emacs 24.2 for Debian stable, and possibly 23.4.

This supercedes the Emacs Ada mode 4.0b that is in the Emacs distribution; that will be removed in a future distribution.

To install from Gnu ELPA:

add to ~./emacs:

(package-initialize)

then invoke M-x list-packages, install Ada mode 5.0.1.

To install from source: download from

http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html

This is the long-awaited complete rewrite, supporting almost all Ada 2012 syntax (aspects are not there yet, but I already have one request for them, so they will be soon). It has been alpha-tested by me and several users on the Emacs Ada mode mailing list, so it is ready for general use.

It is based on an OpenToken-generated grammar, which enables more sophisticated navigation features (i.e. move from 'if' to 'then', 'else', 'end if' etc). It's also a _lot_ easier to maintain.

It also includes experimental support for the new GNAT cross-reference tool gnatinspect, which handles C, C++, Ada.

For more info, see the updated Ada mode manual in the package in info format, or at http://stephe-leake.org/emacs/ada-mode/ada-mode.html

Report bugs/requests to the Emacs Ada mode mailing list; see http://host114.hostmonster.com/mailman/listinfo/emacs-ada-mode_stephe-leake.org

VTKAda

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Fri Feb 7 2014
Subject: Announce: VTKADA 6.1
URL: linkedin.com

I'm pleased to announce VTKAda version 6.1 free edition release 07/02/2014.

VTKAda is Ada-2012 port to VTK (Visualization Toolkit by Kitware, Inc) and Qt5 application and UI framework by Nokia VTK version 6.1.0, Qt version 5.2.0 open source and vtkc.dll, vtkc2.dll, qt5c.dll (libvtkc.so, libvtkc2.so, libqt5c.so) were built with Microsoft Visual Studio 2012 in Windows (WIN32) and gcc in Linux x86-64 Package was tested with gnat gpl 2012 Ada compiler in Windows 8 64bit,Debian 7.3 x86-64.

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

Current state of VTKAda is 42064 procedures and function distributed in 643 packages. 135 examples. All QTAda examples are Qt5 applications.

Current state of QTAda is 11925 procedures and function distributed in 324 packages. There are many new packages and examples in this release.

VTKAda you can use without QTAda subsystem QTAda is Ada port to Qt5 framework and can be used as independent system.

VTKAda and QtAda for Windows and Linux (Unix) free edition with prebuilt Qt 5.2 and VTK 6.1.0 are available from VTK 6.1.0 and Qt 5.2.0 prebuilt for win32 and x86-64 https://rapidshare.com/download/share/5CD62F6FFB431DC394A7F47F5CEFF8DD

[See also "VTKAda", AUJ 34-4, p. 201. —sparre]

Ada Utility Library

From: Stephane Carrez <Stephane.Carrez@gmail.com>
Date: Sun Feb 9 2014
Subject: Ada Utility Library 1.7.0 is available
URL: http://blog.vacs.fr/index.php?post/2014/02/09/Ada-Utility-Library-1.7.0-is-available

Ada Utility Library is a collection of utility packages for Ada 2005. A new version is available which provides:

- Added a text and string builder
- Added date helper operations to get the start of day, week or month time
- Support XmlAda 2013
- Added Objects.Datasets to provide list beans (lists of row/column objects)
- Added support for shared library loading
- Support for the creation of Debian packages
- Update Ahven integration to 2.3
- New option -r <test> option for the unit test harness to execute a single test
- Port on FreeBSD

It has been compiled and ported on Linux, Windows, Netbsd, FreeBSD (gcc 4.6, GNAT 2013, gcc 4.7.3). You can download this new version at http://download.vacs.fr/ada-util/ada-util-1.7.0.tar.gz.

[See also "Ada Utility Library", AUJ 34-1, p. 8. —sparre]

TclAdaShell

From: Simon Wright <simon@pushface.org>
Date: Mon, 10 Feb 2014 20:58:11 +0000
Subject: ANN: TclAdaShell 8.6-2
Newsgroups: comp.lang.ada

TclAdaShell 8.6-2 is available:

https://sourceforge.net/projects/tcladashell/files/source/20140210/

Tcl and Tcl.Tk no longer provide "Null_*" constants or "Is_Null (Ptr : in *) return Boolean;" functions (the types concerned are visibly access types, so the standard "null" is available).

Tcl, Tcl.Ada and Tcl.Tk use "not null" in subprogram parameters where applicable. Note that, although this is an Ada 2005 construct, your GNAT project can still specify Ada 95 if required, because of the use of the GNAT-specific "pragma Ada_2005".

Fixed some confusion between Tcl_UniChar and strings of same.

[See also "TclAdaShell 20090611", AUJ 30-3, p. 146. —sparre]

Ahven

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue, 11 Feb 2014 16:45:16 +0200
Subject: ANN: Ahven 2.4
Newsgroups: comp.lang.ada

I released Ahven 2.4 on Sunday (2014-02-09) and it is available at http://sourceforge.net/projects/ahven/files/

It is mostly a maintenance and bug fix release and the biggest changes are:

- A work-around to Ahven.Framework for Apex and ICCAda. Now Apex Ada compiles the body of Ahven.Framework without errors and ICCAda does not produce any warnings.

The compilers did not correctly handle the body of Indefinite_Test_List package

inside Ahven.Framework when Indefinite_Test_List was at the end of ahven-framework.adb. This was fixed by moving the body to the beginning of the file. (No functional changes.)

Special thanks to Atego and Irvine for providing help with the issue. - Various documentation improvements.

- Alternative comfignat-based build system (contrib/comfignat). It is experimental for now and meant mostly for Linux distribution packagers. From Bjorn Persson.

Known issues:

- Fedora Linux systems need libgnat-static package to be installed before Ahven can be compiled.
- On Windows 8.1 you need to use JNT_RTS instead of JTN_RTS_Console as Janus/Ada runtime. Otherwise, Janus/Ada fails to find Ada runtime system for Ahven.

About Ahven:

Ahven is a simple unit test library (or a framework) for Ada programming language. It is loosely modelled after JUnit and some ideas are taken from AUnit.

Ahven is free software distributed under permissive ISC license and should work with any Ada 95, 2005, or 2012 compiler.

http://ahven.stronglytyped.org/

[See also "Ahven", AUJ 34-1, p. 10. —sparre]

OpenToken

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 21 Feb 2014 14:27:56 -0600
Subject: OpenToken 5.0a released
Newsgroups: comp.lang.ada

OpenToken 5.0a is released; see

http://stephe-leake.org/ada/opentoken.html.

There are many bugs related to empty productions fixed in this version.

OpenToken can now accept bison-style input syntax, and generate OpenToken Ada source for declaring the syntax and grammar, or Emacs lisp source. The Emacs lisp source is used for the Ada and gpr grammars in Emacs Ada mode 5.0

Happy parsing!

[See also "Ada 2012 Grammar", AUJ 34-4, p. 202. —sparre]

Ada-related Products

GNAT Programming Studio

From: AdaCore Press Center
Date: Tue Nov 12 2013
Subject: AdaCore Releases Major New Version of GNAT Programming Studio
URL: http://www.adacore.com/press/gps6/

GPS 6.0 Integrated Development Environment brings upgraded and modernized "Look and Feel"

PITTSBURGH, Pa., NEW YORK and PARIS, November 12, 2013 – ACM SIGAda HILT Conference – AdaCore today announced the release of the GPS 6.0 graphical Integrated Development Environment (IDE), a major upgrade with a significantly revised and cleaner user interface that eases program navigation and editing. With this new version of the GNAT Programming Studio, developers can take advantage of more space for editing and a number of design changes that bring program-related information within easy reach. The revised look and feel is supported by a new relational database at the heart of the GPS engine, making code navigation much more efficient. The principles underlying the GPS 6.0 revision help the IDE achieve its main goal: to serve as a customizable platform for multi-language, multi-tool integration, usable by developers at all experience levels.

The improvements to the IDE's look and feel exploit the latest Gtk+/GtkAda graphical toolkit and encompass a reorganized interface (including more economic usage of screen space), a global search facility, additional view capabilities and further support for color tailoring. GPS 6.0 also brings improved performance and new functionality, including language support for SPARK 2014, syntax highlighting and tool tips for Ada 2012 and SPARK 2014 aspects, editor enhancements, and a number of additions to the scripting API. The GPS 6.0 enhancements have received an enthusiastic response from the product's beta sites.

"GPS 6.0 comes from a major engineering effort to improve the product's overall usability," said Nicolas Setton, GPS Product Manager at AdaCore. "We have been listening to what customers have been telling us, and this new version should be more than an IDE, it should also be a pleasure to use."

GPS is provided with the GNAT Pro development toolset on most platforms, for both native and embedded software development, and GPS 6.0 is available to GNAT Pro customers for download through GNAT Tracker.

A GPS 6.0 demo will be available at www.adacore.com/gps-demo. For further information please contact info@adacore.com.

About GNAT Programming Studio (GPS)

GPS is a powerful Integrated Development Environment (IDE) written in Ada using the GtkAda toolkit. GPS's extensive source-code navigation and analysis tools can generate a broad range of useful information, including call graphs, source dependencies, project organization, and complexity metrics. It also supports configuration management through an interface to third-party Version Control Systems, and is available on a variety of platforms. GPS is highly extensible; a simple scripting approach enables additional tool integration. It is also customizable, allowing programmers to specialize various aspects of the program's appearance in the editor for a user-specified look and feel.

CodePeer

From: AdaCore Press Center
Date: Wed Feb 5 2014
Subject: AdaCore Releases Major New Version of CodePeer Static Analysis Tool
URL: http://www.adacore.com/press/codepeer2-3/

Automatic code review and validation tool brings a new level of flexibility and efficiency to Ada software developers

TOULOUSE, PARIS and NEW YORK, February 5, 2014 – ERTS2 Conference – AdaCore today announced the release of CodePeer 2.3, the latest version of its static analysis tool for the automated review and validation of Ada source code. CodePeer assesses potential bugs before program execution to find errors efficiently and early in the development life cycle. It also performs impact and vulnerability analysis when existing code is modified, and, using control-flow, data-flow and other advanced static analysis techniques, the tool detects problems that would otherwise only be found through labor-intensive debugging.

The latest update to CodePeer delivers more precise diagnostic messages and fewer "false positives". It also includes an independent Ada front end, making it even more efficient and flexible. To simplify the development process, CodePeer 2.3 provides better integration with AdaCore's two IDEs: GNAT Programming Studio (GPS) and GNATbench (the GNAT Pro Ada plug-in for Eclipse and Wind River Systems Workbench). Other enhancements include support for floating point overflow on unconstrained types, the ability to supply target configuration files, and improved support for existing codebases in Ada 83. Improved message review capabilities are now available through pragma Annotate, and the tool provides new warnings when a formal parameter could be declared with a more restrictive mode.

CodePeer is fully integrated into the GNAT Pro development environment and comes with a number of complementary static analysis tools common to the technology – a coding standard

verification tool (GNATcheck), a source code metric generator (GNATmetric), a semantic analyzer and a document generator.

"It has been exciting to bring the 2.3 release to our customers, with CodePeer now established as the most advanced and precise static analysis tool available for Ada," said Tucker Taft, AdaCore Vice President and Director of Language Research. "It was especially gratifying to integrate CodePeer with Ada 2012's contract-based programming capabilities; this has really advanced the state of the art in software verification."

About CodePeer

Serving as an efficient and accurate code reviewer, CodePeer identifies constructs that are likely to lead to run-time errors such as buffer overflows, and it flags legal but suspect code, typical of logic errors. Going well beyond the capabilities of typical static analysis tools, CodePeer also produces a detailed analysis of each subprogram, including pre- and post-conditions. Such an analysis makes it easier to find potential bugs and vulnerabilities early: if the implicit specification deduced by CodePeer does not match the component's requirements, a reviewer is alerted immediately to a likely logic error. During system development, CodePeer can help prevent errors from being introduced, and it can also be used as part of a systematic code review process to dramatically increase the efficiency of human review. Furthermore, CodePeer can be used retrospectively on existing code, to detect and remove latent bugs.

[see also AUJ 34-2, p. 70: AdaCore Releases Major New Version of CodePeer Static Analysis Tool]

GNATcoverage

From: AdaCore Press Center
Date: Wed Feb 5 2014
Subject: AdaCore Releases New Version of GNATcoverage Dynamic Analysis Tool
URL: http://www.adacore.com/press/gnatcoverage1-2/

Award-winning, non-intrusive coverage tool supports all levels of safety certification and adds hardware probe functionality

TOULOUSE, PARIS and NEW YORK, February 5, 2014 – ERTS2 Conference – AdaCore today announced the release of GNATcoverage 1.2, the latest version of its source and object code coverage analysis tool. GNATcoverage's innovative technology does not require instrumentation of the executable, and this new product release supports usage with an iSystem hardware probe generating Nexus trace data, as well as usage with Valgrind on Linux.

GNATcoverage 1.2 supports Ada 95, Ada 2005 and many new features in Ada 2012. It can also be used for the upcoming SPARK 2014 revision and includes Beta support for C. Other enhancements include generation of coverage information for generics on a per-instance basis, and improved HTML output (sortable columns, project awareness). The tool is now integrated with the GNAT Pro development environment.

Qualification material is available to support GNATcoverage usage as a verification tool (DO-178B) or a tool at TQL-5 (DO-178C). It can be used as part of the verification process for systems that need to be certified up to Level A, and can thus supply analysis up to Modified Condition/Decision Coverage (MCDC). GNATcoverage can also be used for railway applications that need to comply with EN-50128:2011 (T2).

"This new release of GNATcoverage considerably expands the product's capabilities," said Cyrille Comar, AdaCore Managing Director. "Furthermore, now that it has been established that object branch coverage is not sufficient for claiming MCDC, we can assert that GNATcoverage is the only coverage technology that does complete MCDC without application-level instrumentation."

About GNATcoverage

Originally developed as part of the Couverture research project, GNATcoverage performs coverage analysis on both object code - instruction and branch coverage - and Ada and C language source code - statement, decision, and Modified Condition/Decision Coverage (MCDC). Unlike most current technologies, the tool works without requiring instrumentation of the executable. Instead, it analyzes trace data generated from a program running on either an instrumented version of AdaCore's GNATemulator tool, Valgrind on Linux, or a target platform equipped with a supported hardware probe. GNATcoverage helps software developers assess the breadth of a testing campaign and provides precise answers to the needs of safety-certification processes, such as the DO-178 avionics standard and the EN-50128 railway standard. GNATcoverage is a major example of an Open Source tool dedicated to software certification, and the tool was awarded an Electrons d'Or prize in 2011 by France's Electroniques magazine in recognition of its innovations and predicted impact on the industry.

Ada and Operating Systems

Mac OS X: XAdaLib

From: Pascal Pignard <p.p11@orange.fr>
Date: Fri, 06 Dec 2013 21:06:37 +0100
Subject: [ANN] XAdaLib 2013 binaries for MacOS including GTKAda and more.
Newsgroups: comp.lang.ada

This is XAdaLib 2013 built on MacOS 10.9 Mavericks for X11 including:

- GTK Ada 3.4.2 with GTK+ 3.4.1 complete,
- Glade 3.10.2,
- GnatColl GPL 2013,
- Florist GPL 2013,

to be installed for instance at /usr/local:

$ cd /usr/local

$ sudo tar xzf xadalib-gpl-2013-x11-x86_64-apple-darwin13.0.0-bin.tgz

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

$ PATH=/usr/local/xadalib-2013/bin:$PATH

Update your GPR_PROJECT_PATH to include gtkada.gpr in it:

$ export GPR_PROJECT_PATH=/usr/local/xadalib-2013/lib/gnat:$GPR_PROJECT_PATH

$ export XDG_DATA_DIRS=/usr/local/xadalib-2013/share

Then see documentation and examples in share directory and enjoy.

See the instructions which have produced the libraries on Blady web site:

http://blady.pagesperso-orange.fr/creations.html#gtkada

XAdaLib binaries have been post on Source Forge:

http://sourceforge.net/projects/gnuada/files/GNAT_GPL%20Mac%20OS%20X/2013-mavericks/

Fedora: AVR-Ada

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon Dec 23 2013
Subject: AVR-Ada 1.2.2 RPMs for Fedora 20
URL: http://arduino.ada-language.com/avr-ada-122-rpms-for-fedora-20.html

As a small Christmas gift, AVR-Ada 1.2.2 RPMs for Fedora 20 (i386 and x86_64) are now available in my fedora.ada-language.com repository.

Like always, create file /etc/yum.repos.d/fedora-adalanguage.repo with contents:

[fedora-adalanguage]

name=Tero's Fedora RPM repository for Ada packages

baseurl=http://fedora.ada-language.com/repo/$releasever/$basearch

enabled=1

And run:

sudo yum install avr-gnat avr-ada-lib --nogpgcheck

Notes:

- The used GCC version is still 4.7.2. Fedora 20 ships with avr-gcc 4.8.x, but AVR-Ada is tested mainly with 4.7.x.
- As before, the packaging is done by using gnat 4.7 binaries from Fedora 18.
- The release contains two of my patches, which are not in the official AVR-Ada 1.2.2 release.
  - The first patch reverts AVR.UART behaviour back to AVR-Ada 1.2 (=interrupt mode also works)
- The second patch fixes linking errors with libavrada.a, so that all boards get correct CPU frequencies and other code.
- The RPMs are unofficial in every possible way and they are not endorsed by Fedora or AVR-Ada projects.
- This time I was bit in a hurry, so they are not tested as well as before. If there are bugs, complain to me (tero.koskinen@iki.fi).

[See also "AVR-Ada for Fedora", AUJ 34-3, p. 143. —sparre]

Debian: GHDL

From: Joris van Rantwijk <joris@jorisvr.nl>
Date: Wed, 15 Jan 2014 21:14:46 +0100
Subject: New Debian package for GHDL
To: Nicolas Boulenguez <nicolas.boulenguez@free.fr>
Cc: debian-ada@lists.debian.org, ghdl-discuss@gna.org

[GHDL is a VHDL compiler/simulator using GCC/GNAT technology. —sparre]

There has recently been a new upstream release of GHDL which fixes many bugs and makes it easier to build the software on Debian systems.

I made a Debian package for that release, ghdl-0.31-1, available here:

http://mentors.debian.net/package/ghdl

I believe the package is in good shape and (almost) ready to upload to the Debian archive. It would be great if you could have a look at it and tell me what you think.

Debian: Default Compiler in Debian 9 "Jessie"

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 03 Feb 2014 00:13:39 +0100
Subject: The default Ada compiler for Debian 8 "Jessie"
Newsgroups: gmane.linux.debian.packages.ada

GCC 4.9.0 should be released in a few months from now, possibly in March 2014. It is already available in experimental and Matthias Klose is actively working on it. Apparently[1] he intends to make GCC 4.9 the default compiler for C, C++ and other languages as soon as it reaches unstable, on as many architectures as possible.

The maintainer of Ada in FreeBSD and Dragonlace has stated at FOSDEM[2] that he intends to skip GCC 4.8 altogether for Ada and package GCC 4.9 instead (but note that the default C and C++ compiler on FreeBSD is now clang/LLVM, not GCC).

Debian 8 "Jessie" will be frozen on November 5, 2014 [3], which leaves us 9 months to transition all Ada packages to the next default Ada compiler.

gnat-4.8 has been in Jessie (testing) since November 2013 but gnat-4.9 does not exist at all yet.

We are faced with a tough choice for the next default Ada compiler. If we choose gnat-4.8, then the transition of all packages can start immediately but Jessie ends up with an "old" compiler (4.8.0: March 2013) which is not the default for other languages and which is different from the one in FreeBSD. If we choose gnat-4.9, this will allow better support for Ada 2012 (e.g. contracts and other aspects) and probably a more recent version of PolyORB too.

I have just created the branch org.debian.gnat-4.9 in monotone and I propose the following plan:

- starting right now, everyone interested (and in particular the maintainers I talked to at FOSDEM: you know who you are!) works hard on updating all the Debian patches for gnat-4.9; this is the top priority.
- at the end of March 2013 (two months from now), we review the state of gnat-4.9: is upstream GCC 4.9.0 released? Is it in unstable? Are we satisfied with the quality and stability of gnat-4.9? and we make the final decision as for the Ada compiler for Jessie.
- Immediately after this decision is made, we update all other packages to the chosen new compiler, starting with ASIS and PolyORB.

The obvious risk with this plan is that, if gnat-4.9 turns out not to be viable, we'll have wasted two precious months for the big transition.

Objections? Commitments? Exuberant enthusiasm? Lukewarm support? Please tell me...

[1] https://lists.debian.org/debian-gcc/2013/12/msg00034.html

[2] http://people.cs.kuleuven.be/~dirk.craeynest/ada-belgium/events/14/140201-fosdem/10-ada-bsd.pdf

[3] https://lists.debian.org/debian-devel-announce/2013/10/msg00004.html

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Mon, 03 Feb 2014 11:28:02 +0000
Subject: Re: The default Ada compiler for Debian 8 "Jessie"
Newsgroups: gmane.linux.debian.packages.ada

> [...]

GHDL has been problematic because, being a compiler itself, it depends rather closely on the sources of gcc. I can report that it has been modified to build against gcc-4.9-20140112 (though I think the compiler used to build it was gnat-4.8 rather than my build of 4.9) and the result passed its testsuite.

So barring major changes between that snapshot and gcc4.9 release, GHDL should not impede this transition if you decide to go for 4.9.

(However the current Debian GHDL package, awaiting sponsorship at http://mentors.debian.net/package/ghdl is based on 4.8, and I think it would be better to push this forward and update to 4.9 later rather than wait for Gnat-4.9)

From: Florian Weimer
Date: Thu, 06 Feb 2014 20:58:06 +0100
Subject: Re: The default Ada compiler for Debian 8 "Jessie"
Newsgroups: gmane.linux.debian.packages.ada

> [...]

My own Ada 95 sources do not compile with either GCC 4.8 or GCC trunk. GNAT 4.6 in wheezy appears to be fine. This is just one data point. Not sure what to read into it, and considering that 4.8 and probably 4.9 are similarly afflicted, it doesn't seem to matter anyway.

I need Ada 95 mode because I use limited return types (correctly, I think) to implement multiple inheritance. I suspect that's why you get if you learn the language by yourself, without guidance from experienced users. You tend to rely on features that are rarely used by others.

From: Florian Weimer
Date: Thu, 06 Feb 2014 23:02:37 +0100
Subject: Re: The default Ada compiler for Debian 8 "Jessie"
Newsgroups: gmane.linux.debian.packages.ada

> A better data point would be an actual bug report with a reproducer.

<http://gcc.gnu.org/PR57902>

I had succeeded in reducing the test case, but I had not been able to bisect the change that caused it or isolated the bug further.

Debian: GNAT

From: Nicolas Boulenguez <nicolas.boulenguez@free.fr>
Date: Tue, 11 Feb 2014 01:59:08 +0100
Subject: gnat-4.9
Newsgroups: gmane.linux.debian.packages.ada

Congratulations, head of the gnat-4.9 branch builds on amd64!

[...]

I have tried to build some packages, everything seems ok without refreshing one single patch: asis (2013), dh-ada-library (with all warnings), libxmlada (quite old version though), libgmpada (with all warnings), ada-reference-manual.

Maybe gnat-4.9 should conflict with gnat-4.8, at least until we are bored playing with them and one is selected for unstable.

Maybe gnat-4.9 should be named gnat4.9 instead, because policy 5.6.12 forbids hyphens in native package names.

Only warnings have changed a lot:

* Either -gnatwa should not activate -gnatw.i, or the online documentation should be modified accordingly.

* I think that the new -gnatw.y warning should not be activated by -gnatwa, as it produces a lot of noise for a very rare use case.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sat, 22 Feb 2014 00:59:32 +0100
Subject: gnat-4.9 uploaded to the NEW queue
To: debian-ada@lists.debian.org

I have just uploaded gnat-4.9 (4.9-20140218-1) to the NEW queue; the FTP masters will examine the package and upload it to experimental, hopefully within a few days.

This is revision 3f519073fea55539758f3fcd8223535269911255 on org.debian.gnat-4.9.

Thanks to all who contributed to this; the near future is looking bright with several other packages almost ready for upload :)

FreeBSD: Ahven

From: John Marino <dragonlace.cla@marino.st>
Date: Wed, 12 Feb 2014 04:16:25 -0800
Subject: Re: ANN: Ahven 2.4
Newsgroups: comp.lang.ada

I've updated the Avhen port in FreeBSD to version 2.4:

http://www.freshports.org/devel/ahven

[See also the release announcement for Ahven 2.4 earlier in this issue. —sparre]

FreeBSD: PLplot and Ncurses

From: John Marino <dragonlace.cla@marino.st>
Date: Sun Feb 16 2014
Subject: Ports: PLplot Ada bindings now available
URL: http://www.dragonlace.net/posts/Ports:PLplot_Ada_bindings_now_available/

FreeBSD has PLplot, cross-platform software package for creating scientific plots, at the latest stable version 5.10.0 (as of today). What it did not have is the option to build the Ada bindings although most other languages were available as an option. Rather than update the currently unmaintained PLplot port, I created a new port at math/plplot-ada to build the Ada bindings separately.

In separate news, the Ada bindings to ncurses were completely revamped. Previously the port didn't actually build the library. Now it does and it should work as expected. The port is located at devel/adacurses.

FreeBSD: Ironsides

From: John Marino <dragonlace.cla@marino.st>
Date: Mon Feb 17 12:30:00 CET 2014
IRC-channel: #Ada
IRC-network: irc.freenode.net

12:30 < marino> sparre: dns/ironsides is in FreeBSD ports now

Debian: PolyORB

From: Xavier Grave <xavier.grave@ipno.in2p3.fr>
Date: Fri, 21 Feb 2014 13:57:13 +0100
Subject: polyorb build ok with gnat-4.9
To: <debian-ada@lists.debian.org>

I have a first build version [1] of polyorb with gnat-4.9 (at least !). It's building a very up to date [2] version of upstream. My code source checks seem to indicate that the PCS_version are compatible.

I have tests available mostly on the dsa part, I'll be interested if people can be volunteers to test CORBA part.

The test suite is still disabled and I'll check this as soon as possible.

[1] org.debian.polyorb a60b2e30ffad798f3666803b079f5e254804bf45

[2] com.adacore.polyorb.debian a33ec96a70d7827d73ad160eceee81781cd529cd

References to Publications

Power-saving with AVR-Ada

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue Nov 12 2013
Subject: Saving power with AVR-Ada, part 2: power down mode and watchdog
URL: http://arduino.ada-language.com/saving-power-with-avr-ada-part-2-power-down-mode-and-watchdog.html

After my previous article[1], Olimex people pointed out[2] that their Olimexino-328 board is able to use much less than 4mA if powered through the battery connector.

So, I went and tested their claims and they were correct, indeed. When running Olimexino-328 at 3.3V using battery connector, power down mode instead of power save mode, and watchdog to wake up the board once per minute, I managed to get power usage down to 0.02mA (0.02 milliamps, 20 microamps).

I also made some observations:

- INA219 sensor is pretty accurate when compared to readings from my multimeter

- However, INA219 sensor can measure current only down to 0.1mA, after that I get 0 or negative readings (could be something related to my code)

- The AVR.Watchdog package of AVR-Ada doesn't really support Arduino/atmega328p, so I had to configure the watchdog manually

- At one point, Olimexino-328 was sleeping really deeply and I had to solder ISP header pins to the board because I wasn't able to program the board via serial port

It is somewhat complex to add good watchdog support for atmega328p and also “trigger an interrupt instead of reset” functionality, so I won't be committing my watchdog code to AVR-Ada repos any time soon. Meanwhile, you can get the code from my arduino-blog repository[3], examples/deep-sleep[4] directory.

[See also “Saving Power with AVR-Ada”, AUJ 34-4, p. 205. —sparre]

[1] http://arduino.ada-language.com/saving-power-with-avr-ada.html

[2] http://olimex.wordpress.com/2013/11/05/experimenting-with-low-power-modes-and-arduino/

[3] https://bitbucket.org/tkoskine/arduino-blog/

[4] https://bitbucket.org/tkoskine/arduino-blog/src/tip/examples/deep-sleep/

Parallel Addition

From: Jim Rogers
Date: Sat Jan 4 2014
Subject: Parallel Addition
URL: http://sworthodoxy.blogspot.dk/2014/01/parallel-addition.html

While sequential addition of a set of numbers, such as the elements of an array, is well understood, the implementation of a parallel addition algorithm provides both a new set of challenges and an opportunity to use more than one core on your computer.

The concept behind this parallel addition algorithm is to create several tasks, each of which reads a pair of values from the set of values to be added, performs the addition, then puts its result back into the set of values to be added. When the set of values to be added is reduced to 1 the addition is complete, and the remaining value in the set is the final total.

The following figures graphically show a concept of how the algorithm works. In practice the exact pairing of values to be added may differ. That difference is not important to the result of the program because addition is commutative.

[Includes source text. —sparre]

Case Study for System to Software Integrity Includes SPARK 2014

From: Yannick Moy
Date: Tue Jan 21 2014
Subject: Case Study for System to Software Integrity Includes SPARK 2014
URL: http://www.spark-2014.org/entries/detail/case-study-for-system-to-software-integrity-includes-spark-2014

The NoseGear challenge[1] was proposed at the Workshop on Theorem Proving in Certification[2] as a small yet realistic case of critical system, to demonstrate and compare benefits and limitations of formal methods.

We have extended the scope of the challenge to add a logger and a GUI to the initial computation problem, to make it more realistic. We have developed an architecture of this system in AADL, a model of the computation in Simulink, code for the logger in SPARK and code for the GUI in Ada. Code is also automatically generated from AADL (to Ada) and Simulink (to SPARK), so that the complete concurrent application can be run with a simulator of the physical system. Verification activities include formal verification of the manual and generated SPARK code for absence of run-time errors and verification of properties expressed as contracts. All the artifacts (models, code, verification results) can be accessed from a prototype tool for agile certification, which records automatically traceability links between artifacts.

The paper we will present at ERTS explains the motivation behind this work, and the expected benefits when applied to actual systems.

[1] http://www.cl.cam.ac.uk/~mjcg/FMStandardsWorkshop/NoseGear.html

[2] http://www.cl.cam.ac.uk/~mjcg/FMStandardsWorkshop/

Will My Car be Safe to Drive?

From: Robert Dewar
Date: Mon Feb 3 2014
Subject: Will My Car be Safe to Drive?
URL: http://johndayautomotiveelectronics.com/will-my-car-be-safe-to-drive/

It's no secret that the cars we drive today, and especially those we will drive in the near future, have huge amounts of sophisticated software aboard. By some accounts the number of lines of code in a car can significantly exceed the number of lines of code in a modern commercial jetliner. And as with the jetliner, we are entrusting our safety to the reliability of this software.

[An interesting article about the (lack of) safety in automotive software. —sparre]

Reference Manual in "info" Format

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 24 Feb 2014 03:53:50 -0600
Subject: arm_info 2012.2 released
Newsgroups: comp.lang.ada

Version 2012.2 of arm_info is available at http://stephe-leake.org/ada/arm.html

This contains the latest standard text from AdaIC, which has minor editing changes from the previous version.

There are also changes in the source to help with Debian packaging.

Ada Inside

SPARK CubeSat in Space

From: AdaCore Press Center
Date: Tue Nov 19 2013
Subject: AdaCore and Altran Toolsets Help Launch CubeSat into Orbit
URL: http://www.adacore.com/press/cubesat/

NASA-sponsored satellite from Vermont Technical College uses GNAT Pro and SPARK

NEW YORK, PARIS, and BATH (UK), November 19, 2013 – Today, AdaCore and Altran announced a new space application for the GNAT Pro technology and SPARK language toolset, with the successful launch of Vermont Technical College's Lunar CubeSat. The tiny satellite, measuring only 10 cm x 10 cm x 10 cm and weighing 1.1 kg, was launched into a 500 km earth orbit, where it will remain for about three years to test the systems that will be used for the eventual lunar mission. The CubeSat project is part of NASA's ELaNa IV program (Educational Launch of Nano-satellites).

The CubeSat's navigation and control software was developed in SPARK/Ada using AdaCore's GNAT Programming Studio (GPS) IDE and GNAT Pro compiler and exploiting Altran's SPARK toolset to prove the absence of run-time errors. The software was developed at Vermont Technical College by a team of undergraduate students under the direction of Dr. Peter Chapin. Although they had no previous knowledge of SPARK or Ada, the students came up to speed quickly and were able to take advantage of SPARK's various annotations to produce robust code.

"We specifically chose to write the control program for our CubeSat in SPARK because it offers increased reliability over the C language software used in almost all CubeSats to date," said Prof. Carl Brandon, the project leader from Vermont Technical College. "The success of the fairly complicated software on this ELaNa CubeSat gives us confidence in using SPARK 2014 for the much more complicated and expensive lunar mission."

"We are delighted to see our technologies once again being launched into space," said Robert Dewar, AdaCore President. "You only get one shot for this kind of application, so it is critical to produce safe and totally reliable software. In this case, it is very encouraging to see students without prior experience using SPARK and GNAT Pro together to achieve this goal."

For more information, please visit:

- http://www.cubesatlab.org

[See also "Vermont Tech CubeSat Launch Delayed", AUJ 34-4, p. 198. —sparre]

Muen Separation Kernel

From: AdaCore Press Center
Subject: Muen Separation Kernel Lays Open Source Foundation for High-Assurance Software Components
Date: Tue Dec 10 2013
URL: http://www.adacore.com/press/muen-separation-kernel/

Swiss university develops formally verified Open Source kernel using SPARK language and AdaCore GNAT tools

NEW YORK, PARIS and RAPPERSWIL, Switzerland, December 10, 2013 – The Institute for Internet Technologies and Applications at the University of Applied Science in Rapperswil (Switzerland) and AdaCore today announced a significant expansion of the Open Source software model into the domain of high-assurance systems with the preview release of the Muen Separation Kernel. The Muen Kernel enforces a strict and robust isolation of components to shield security-critical functions from vulnerable software running on the same physical system. To achieve the necessary level of trustworthiness, the Muen team used the SPARK language and toolset to formally prove the absence of run-time errors. Using AdaCore's GNAT development environment to build their software, the team was able to achieve high productivity.

The public preview release of the Muen Separation Kernel in Autumn 2013 is the first major milestone for the ongoing Muen project, whose goal is to produce a trustworthy Open Source foundation for component-based high-assurance systems. This is an area of high potential growth, and indeed Open Source software promises to play an increasing role in the development of safe and secure systems.

"It's an exciting occasion," said Cyrille Comar, Managing Director of AdaCore, "for AdaCore to be participating in the birth of an Open Source community around a separation kernel that can be verified formally using Open Source tools, such as those we develop with our partner Altran. Since this type of software is expensive to produce, community-based development offers an attractive cost-sharing model for the main stakeholders. And openness in the code, and in its security-related verification data, is a key element of the trust that is required for secure software."

The name "Muen" is a Japanese term that means "unrelated" or "without relation", reflecting the main objective for a separation kernel: ensuring the isolation between components. Since a separation kernel enforces isolation, resource control and data flow in a component-based system, any errors in the kernel would be fatal to the security of all components. To prevent such a calamity, the Muen Kernel was written in SPARK, an Ada-based language with a long and successful track record in developing high-assurance systems. The SPARK toolset enabled the Muen team to perform static formal verification of the Kernel and to prove the absence of all run-time errors. In the future, functional correctness proofs will be added to the Kernel by using SPARK in conjunction with an interactive theorem prover.

The Muen developers used SPARK with a zero-footprint runtime – a mode where no runtime environment, and only a minimum of supporting code, is required. This setup is ideal for critical low-level programming, since no unnecessary libraries are introduced into the system.

"The Open Source license of the Muen Separation Kernel, combined with the SPARK and GNAT tools, makes it possible for the community to use Muen as a trusted core component in high-assurance systems," said Prof. Dr. Andreas Steffen, Head of the Institute for Internet Technologies and Applications. "Anyone can inspect and compile the source code and reproduce the formal proofs at any time."

About the Muen Project: "Trustworthy by Design -- Correct by Construction"

The Institute for Internet Technologies and Applications (ITA) at the University of Applied Science Rapperswil (HSR) in Switzerland started the Muen Separation Kernel project to create an Open Source foundation for high-assurance platforms. To achieve trustworthiness exceeding any other Open Source kernel or hypervisor, the absence of runtime errors has been formally proven using the SPARK language and toolset. Through close cooperation with secunet Security Networks AG in Germany during the whole design and implementation process, the Muen Separation Kernel is assured of meeting the requirements of existing and future component-based high-security platforms.

The Git repository for the Kernel is available here:

- http://git.codelabs.ch/?p=muen.git

A snapshot of the Muen repository can be downloaded here:

- http://git.codelabs.ch/?p=muen.git;a=snapshot;h=master;sf=zip

The Muen Separation Kernel is available under the GNU General Public License version 3.

From: Reto Buerki <reet@codelabs.ch>, Adrian-Ken Rueegsegger <ken@codelabs.ch>
Subject: The Muen Separation Kernel
Date: Mon Feb 10 2014
URL: http://muen.codelabs.ch/

Trustworthy by Design – Correct by Construction

The Muen Separation Kernel is the world's first Open Source microkernel that has been formally proven to contain no runtime errors at the source code level. It is developed in Switzerland by the Institute for Internet Technologies and Applications (ITA) at the University of Applied Sciences Rapperswil (HSR). Muen was designed specifically to meet the challenging requirements of high-assurance systems on the Intel x86/64 platform. To ensure Muen is suitable for highly critical systems and advanced national security platforms, HSR closely cooperates with the high-security specialist secunet Security Networks AG in Germany.

A Separation Kernel (SK) is a specialized microkernel that provides an execution environment for components that exclusively communicate according to a given security policy and are otherwise strictly isolated from each other. The covert channel problem — largely ignored by other platforms — is addressed explicitly by these kernels. SKs are generally more static and smaller than dynamic microkernels, which minimizes the possibility of kernel failure, enables the application of formal verification techniques and the mitigation of covert channels. Muen uses Intel's hardware-assisted virtualization technology VT-x as core mechanism to separate components. The kernel executes in VMX root mode, while user components, so called subjects, run in VMX non-root mode.

Note:

- Muen is currently a prototype implementation. We do not yet consider it to be fit for production use.

Features:

- Minimal SK for the Intel x86/64 architecture written in the SPARK language
- Full availability of source code and documentation
- Proof of absence of runtime errors
- Multicore support
- Nested paging (EPT) and memory typing (PAT)
- Fixed cyclic scheduling using Intel VMX preemption timer
- Static assignment of resources according to system policy
- Event mechanism
- Shared memory channels for inter-subject communication
- Minimal Zero-Footprint Run-Time (RTS)
- Support for 64-bit native and 32/64-bit VM subjects

[...]

Deep Blue Capital Selects Ada for Financial System Development

From: AdaCore Press Center
Date: Wed Jan 22 2014
Subject: Deep Blue Capital Selects Ada for Financial System Development
URL: http://www.adacore.com/press/deep-blue-capital-financial-system-development/

Reliable Ada software gives trading firm a competitive edge

NEW YORK and PARIS, January 22, 2014 – AdaCore today announced the adoption of its GNAT Pro Ada Development Environment by Deep Blue Capital (DBC), a propriety trading firm. DBC rotates teams through the world time zones at their Amsterdam-based offices to trade twenty-four hours a day on all of the world's major stock exchanges. DBC employs algorithmic trading systems developed in Ada with AdaCore's GNAT Pro development environment; these systems gather market information and automatically send buy and sell orders with minimal human intervention. DBC, a small company with fewer than twenty employees, can operate globally because of its efficient and reliable software.

The Ada language and AdaCore's GNAT Pro development environment help DBC's developers create systems that can easily and reliably handle the huge influx of price data they receive and the large number of daily financial operations. Their business requires their computers to run continuously, and DBC's use of Ada means that their transaction system is immune to issues like integer overflow that plague systems developed in other languages.

The automated trading system contains more than 1 million lines of code written almost entirely in Ada. It must handle 40,000 price updates a second at the same time as smoothly managing DBC's 10,000 daily transactions, all while remaining perfectly dependable. These volumes and the ceaseless invention of new trading strategies by DBC's researchers requires systems that are easily updatable and adaptable to new technologies. The Ada programming language offers developers a high degree of control so they can create systems capable of handling a large number of operations. As Ada detects many kinds of errors at compile time rather than run time, mistakes are detected and corrected early. This reduces debugging costs.

According to DBC Chief Technology Officer Duncan Sands, creating reliable and efficient software is vital to his company's ability to compete with much larger financial institutions. "Given the fact that we are a small company with limited resources, using Ada with GNAT Pro allows us to create the software we need to compete with the systems of much bigger financial companies," Sands explains. "Our efficiency in controlling the costs of writing reliable software is crucial for our business performance. We just do not have the luxury of spending days tracking down programming errors through heavy debugging sessions. Combined with the fact that our development team's speciality is finance and not software engineering, that our code base exceeds one million lines and that our software up-time requirement is 24 hours a day, selecting Ada was the obvious choice."

Ada in Context

Maximum Number of Tasks?

From: Fritz VonBraun <sf@saf.com>
Date: Tue, 12 Nov 2013 01:53:53 -0800
Subject: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

I was wondering about the maximum number of tasks in Ada but I couldn't find any information. The question is, is a task in Ada technically similar to a thread in Windows under the hood? Threads are restricted by the stack size that each thread has reserved, so in practice the maximum number of threads is about 2000.

The reason I'm asking is that I wonder if Ada provides a more comfortable solution to the thread pool problem. In C++ for example I create a number of threads roughly equal to the number of processor cores and then have a number of jobs that are distributed over the threads and which implement a time sharing system by returning control to the thread which then assigns time to another job.

Would I have to do the same in Ada or are tasks meant to be "micro objects" of which many can be created and the Ada runtime does effectively what my threadpool system does in C++?

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue, 12 Nov 2013 11:59:13 +0100
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [maximum number of tasks]

Probably because it is both compiler, hardware and operating system dependent.

> [task similar to a thread in Windows]

That depends on which compiler you use. Some (most?) versions of GNAT use operating system threads to implement tasks. Janus/Ada implements tasks in its own run-time system.

> [maximum number of threads is about 2000]

I just made a quick test on my laptop. It appears that I can create 32041 tasks before I have to do something special to avoid problems.

The test was done on a Debian 7.2 system with the GNAT 4.6 compiler distributed with Debian.

The test program:

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Task_Demo is

   task type Demo_Task (Index : Positive) is
      entry Stop;
   end Demo_Task;
   type Demo_Task_Reference is access
                                  Demo_Task;
   task body Demo_Task is
   begin
      Put_Line (Positive'Image (Index) & "
                        launched.");
      accept Stop;
      Put_Line (Positive'Image (Index) &
                        " stopping.");
   exception
      when others =>
         Put_Line (Positive'Image (Index) &
               " terminated by an exception.");
   end Demo_Task;

   Collection : array (1 .. 32_041) of
            Demo_Task_Reference;
begin
   for I in Collection'Range loop
      Collection (I) := new Demo_Task
                                  (Index => I);
   end loop;

   delay 1.0;

   for I in Collection'Range loop
      Collection (I).Stop;
   end loop;
end Task_Demo;
```

Reducing the stack size for the individual tasks does not seem to make a difference.

> [...] are tasks meant to be "micro objects" of which many can be created and the Ada runtime does effectively what my threadpool system does in C++?

That depends on your compiler.

From: Georg Bauhaus <bauhaus@maps.arcor.de>
Date: Tue, 12 Nov 2013 13:52:15 +0100
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [...]

If you have jobs that do not require intermittent communication among them, then in particular, I'd be sure to have a look at the Paraffin library. http://paraffin.sourceforge.net/

As an example of a different setup, I have seen a program that had the number of tasks be about 4x that of processors; all ran at the "same" time and the number of tasks was suggested by the program's logic, not by either hardware or OS. Load distribution seemed very well handled (GNAT on GNU/Linux in this case), 4 x #CPU was a sweet spot regarding the number of tasks.

From: Brad Moore <brad.moore@shaw.ca>
Date: Thu, 05 Dec 2013 20:26:50 -0700
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [...]

Paraffin has had several flavours of task pools implemented for some time now.

You can do as you suggest and have a task pool that has the same number of tasks as there are cores in your system, and distribute work across these tasks.

There are three main “flavours” of task pool.

1) No pool at all, just allocate workers on the fly and distribute work across the set of workers

2) A task pool that can be dynamically (or statically) created that contains a bounded (or unbounded) number of workers that can be applied to any number of parallelism opportunities. These task pools allow a worker to migrate to cores, if the OS supports migration (e.g. Windows and Linux)

3) A Ravenscar compliant task pool that is more suitable for real time, where the workers must be statically allocated to cores, and cannot migrate.

To see a demo of these task pools, you could try running the test_parallel_loops or test_parallel_recursion executables that are included with the source for Paraffin.

To see the Ravenscar version of these task pools, you would need to execute the test_ravenscar_parallel_loops and test_ravenscar_parallel_recursion examples.

Rather surprisingly, there is not a significant difference between these three task pool models. In general using a task pool will give a slight edge in performance over creating workers on the fly, but the difference is barely noticeable in these examples.

Also, intermittent communication between the workers is OK, as long as the communication has the necessary safeguards.

For instance you could have several workers performing some lengthy calculation, that briefly writes some value to an IO port. If the IO port was wrapped in a protected object, it may make sense to allow the multiple workers to access this protected object to perform I/O.

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Tue, 12 Nov 2013 05:21:05 -0800
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [...]

I remember that I saw at FOSDEM 2013 in the “Ada Developer Room” a demo that plotted a Mandelbrot set by using a matrix of tasks. A participant asked then “can you do that with 10_000 tasks?” The size of the matrix was changed to 100 x 100 and everything worked smoothly; so, in that case you could create at least 10_000 tasks. The PC was a laptop running some kind of Linux + GNAT, if I remember correctly.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 12 Nov 2013 21:02:46 +0100
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [FOSDEM 2013, Ada Developer Room]

Yes, you remember correctly; I was the one doing that demo :)

That's because the Linux kernel allocates virtual address space to each task but does not allocate any physical memory (RAM or swap) unless and until the task writes to memory (i.e. creates variables on its stack). Even then, Linux only allocates the pages actually written to, not the full 2 MiB (or whatever the default is) per task.

After the demo, I re-ran the program at home and saw it allocate 19.8 GiB of virtual address space (I re-checked just now) and thought: gosh am I lucky I've been running 64-bit Linux since 2006 :)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 12 Nov 2013 21:04:12 +0100
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [...]

Oh and by the way, I will introduce task pools in my demo at FOSDEM 2014 :)

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 12 Nov 2013 08:54:18 -0700
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [threadpool system]

Tasks should reflect inherent concurrency in the problem space, not some aspect of the hardware or OS. I have never encountered any problem following this rule, whether it was 60-70 Ada-83 tasks on a 640 KB DOS system in the 1980s or hundreds of tasks on Linux this year.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 12 Nov 2013 17:17:11 +0100
Subject: Re: Maximum Number Of Tasks?
Newsgroups: comp.lang.ada

> [...] are tasks meant to be “micro objects” of which many can be created and the Ada runtime does effectively what my threadpool system does in C++?

If you have native tasking then tasks are as fat as threads. If you have tasking implemented within one thread (rare), tasks can be “micro”, but then they most likely will be unable to perform I/O concurrently.

> Tasks should reflect inherent concurrency in the problem space, not some aspect of the hardware or OS.

It is not that simple. The problem space may encompass the hardware, e.g. in the case of communication and services. Which is typically the case when worker tasks pool comes in question.

Making Guarantees About Record Components

From: J. Kimball <jkimball4@gmail.com>
Date: Tue, 19 Nov 2013 12:49:26 -0600
Subject: Making guarantees about record components
Newsgroups: comp.lang.ada

I'm trying to guarantee that two record component values map to the same value of another type.

```
type A is (...);
type C is (...);

M : array (A) of C := (...);

type R is record
  A1 : A;
  A2 : A;
end record
  with Dynamic_Predicate =>
          (M (R.A1) = M (R.A2) );
```

Is this the best solution we have as of Ada 2012?

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 19 Nov 2013 15:57:22 -0700
Subject: Re: Making guarantees about record components
Newsgroups: comp.lang.ada

> [...]

If you really want to guarantee that the property always holds, this doesn't do that. Changes to components of variables of type R, and changes to M, may invalidate the predicate but not be detected until later. To really guarantee the property, you'd have to encapsulate M and all instances of R so that all such changes can be checked.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Wed, 20 Nov 2013 03:36:50 -0600
Subject: Re: Making guarantees about record components
Newsgroups: comp.lang.ada

> [...]

The rules for when the Dynamic_Predicate is checked are in LRM 3.2.3 31/3. To me, that says any changes to an object of type R are checked, but not changes to M.

There is no value for M that satisfies this constraint for all possible values of R, so this does not seem like a well-defined problem.

I think you need a private type hiding both R and M to enforce this constraint.

Arduino Due

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 21 Nov 2013 01:33:42 -0600
Subject: Arduino Due
Newsgroups: comp.lang.ada

I've recently purchased an Arduino Due (http://arduino.cc/en/Main/arduinoBoardDue), to build a home robot. The processor is an AT91SAM3X8E, which according to Atmel http://www.atmel.com/devices/SAM3X8E.aspx is an ARM Cortex-M3 84 MHz 32 bit processor, no floating point hardware, along with a bunch of IO stuff.

Has anyone ported GNAT to this? It appears I can use gcc targeted to arm, and specify -mcpu=cortex-m3.

I'm guessing the runtime from AVR-Ada could be useful, depending on how much is in assembler.

The Atmel website provides a C/C++ IDE and a download tool, so I can at least write hello_world.c and try things out, but I'd like to write real code in Ada.

I've ported GNAT to a couple of different processors before, but it's been a while ...

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Thu, 21 Nov 2013 17:58:09 +0200
Subject: Re: Arduino Due
Newsgroups: comp.lang.ada

> [...]

I have also Due, actually has been almost a year already[1], but I haven't had time to port GNAT on it yet. But it is on my todo list along with N+1 other things. :)

> [gcc]

Yes, Due has Atmel's Cortex-m3 class ARM microcontroller and it is

supported by gcc.

> [run-time]

You can probably get the runtime skeleton from AVR-Ada, but most of AVR.* packages are useless. They control peripherals of attiny/atmega/at90 AVR microcontrollers (done mostly in Ada, but most of the register addresses and stuff should be specific to AVRs).

Currently, the best place to start is Lucretia's work at https://github.com/Lucretia/ , especially TAMP:

https://github.com/Lucretia/tamp

> [...]

I haven't checked AT91SAM3X8E datasheet in detail, but unless Atmel has recycled their UART/I2C/SPI/etc peripheral logic from AVRs to ARM, you have pretty hard road a head. (You need to either create bindings to C functions or implement all peripheral handling from scratch.)

Btw, if you would use Arduino Uno and AVR-Ada, you could have robot coded in about 30 lines of code[2]. ;)

[1] http://arduino.ada-language.com/arduino-due.html

[2] http://arduino.ada-language.com/remote-controlled-robot-using-xbees-and-ada.html

From: MatthiasR <MatthiasR@invalid.invalid>
Date: Sun, 24 Nov 2013 14:12:35 +0100
Subject: Re: Arduino Due
Newsgroups: comp.lang.ada

I don't know about a ready-to-use solution *) for Cortex-M3, but besides the already mentioned project from 'Lucretia' there are some more starting points:

http://sourceforge.net/projects/arm-ada/

- for LPC21xx (ARM7); with Ravenscar runtime

- as far as I know based on GNAT for Mindstorms

https://github.com/telrob/stm32-ada

- for STM32F4 (Cortex M4F); with Ravenscar runtime

- I have made some tests on a STM32F4-Discovery board; simple test programs with multiple tasks are working

*) technically speaking, there is one, but I assume it's out of question for your project:

- GNAT Pro for ARM supports Cortex M3, M4F and R4F, 'Zero Footprint' and Ravenscar runtimes are provided.

The source distribution of GNAT GPL 2013 contains most parts of the ARM support, but it is not complete. Some parts of the bareboard runtimes are located in the 'zfp-support' package. This package was publicly released only as part of the sources for GNAT GPL for Mindstorms and GNAT GPL for AVR. There was neither a 2013 release for Mindstorms nor for AVR, thus the most recent release of this package was in 2012. And back then, there was no support for ARM...

Passing Large Objects as Arguments

From: Fritz Von Braun <sf@saf.com>
Date: Sat, 23 Nov 2013 23:20:53 -0800
Subject: How To Pass Large Object Arguments
Newsgroups: comp.lang.ada

I am fairly new to Ada and I am wondering how I should pass large parameters to subprograms like arrays or records that contain other components like vectors or lists.

I did a lot of reading but wasn't able to find a definite answer. The general consensus I got from Barne's book and various blogs and whitepapers from Universities was that in theory IN parameters are copied but the compiler manufacturer is free to implement a reference to the original object and so on. So basically what I found out there is no concrete rule that says "parameter of that size or greater are passed by reference internally".

So my question is, is there a de facto standard at least? What does GNAT do in such cases? (In all honesty, my programs will never run on anything but GNAT, so other compilers don't really matter to me). I am considering passing objects that I think are too big for a copy operation through an access parameter, but that would basically contradict the principle of problem orientation instead of machine orientation. I would really rather be able to handle these situations without having to worry about the underlying mechanism myself.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sun, 24 Nov 2013 12:12:46 +0100
Subject: Re: How To Pass Large Object Arguments
Newsgroups: comp.lang.ada

> [...]

You should not pass parameters by copy or by reference; this is the job of the compiler. And you should not presume to know better than the optimizer in the compiler which method is faster.

So, pass parameters "in" or "in out".

> I did a lot of reading but wasn't able to find a definite answer.

The definitive resource is the Ada Reference Manual (i.e. the ISO standard that defines the language), which is Free, unlike for some other languages...

http://www.adaic.org/resources/add_content/standards/12rm/html/RM-6-2.html

GNAT normally passes arrays (including Strings) by reference but very short arrays might be passed by copy in registers, which is *faster* than by reference.

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Sun, 24 Nov 2013 07:45:31 -0500
Subject: Re: How To Pass Large Object Arguments
Newsgroups: comp.lang.ada

> [...]

Let the compiler worry about it.

You only need to step in if you can show (for example with profiling) that your program's performance is inadequate AND the problem is due to a "foolish" choice of parameter passing mechanism on the part of the compiler.

Certain types are definitely passed by reference, e.g., limited types that can't be copied. For types that could be passed either way I think you can be confident that any sane compiler will do "the right thing" and pass large objects by reference.

From: Georg Bauhaus <bauhaus@maps.arcor.de>
Date: Mon, 25 Nov 2013 11:59:33 +0100
Subject: Re: How To Pass Large Object Arguments
Newsgroups: comp.lang.ada

> [...]

Exactly. The language rules in LRM 6.2 (see Ludovic's message) make the compiler choose among the possibilities so established. In addition, some rules are AS-IF rules, so optimizers can manage parameter passing as they see fit. They do, drawing upon the compiler writers' knowledge of the architecture:

If a primitive operation of a "small" tagged type has Inline applied to it, then, for example, GNAT's optimizer may drop all reference to the object when translating Object.<primitive operation>.

```
function Val (Object : OO_Type)
         return Some_Integer;
  pragma Inline (Val);

function Val (Object : in T)
         return Integer is
begin
  return Object.Data;
end Val;
```

is one example. Its translation, at -gnatn -O2, shows that record components need not be made publicly visible to address worries about mechanism.

This feature of the language, i.e. making by-copy/by-reference and in/out separate concepts, removes the need for access parameters almost everywhere. And also thinking about them if not problem oriented ;-)

From: Adam Beneschan <adam@irvine.com>
Date: Mon, 25 Nov 2013 08:53:56 -0800
Subject: Re: How To Pass Large Object Arguments
Newsgroups: comp.lang.ada

> [...]

The RM has some rules about how certain types are to be passed. Elementary types are always passed by copy; those are types that essentially aren't broken down into subcomponents, i.e. numbers, enumerations, access types. This is true even for IN OUT parameters; the value will be passed by copy, and a new value will be copied back after the procedure or function returns. Tagged types, tasks, protected types, other limited types, and any record or array containing one of those, are always passed by reference. This is true even for IN parameters. If the "vectors" or "lists" you're referring to are types in one of the Ada.Containers packages, then they will be passed by reference since Ada.Containers.Vectors.Vector is defined to be a tagged type, and I think that's true for all the other containers.

But for records and arrays that don't fall into one of those categories, it's up to the compiler. And the compiler's decision may depend on the target processor. One of our compilers (for a particular RISC-ish target) would pass any record up to four 32-bit words by copy, in registers. However, for a Pentium, which has very few registers, an implementation like this wouldn't make sense, and there's no point in copying a record to the stack if it isn't required by the language.

So for record and array types that aren't specified by the RM, you shouldn't worry about the parameter passing mechanism, and let the compiler decide what it thinks the best way is. You should also write code in a way that assumes either one or the other mechanism could be used. That is, if you call Foo(Param => X) where X's type is some record type, and somewhere while Foo is running, something happens that causes a field in X to be modified, Foo itself may or may not notice that that field has changed if it accesses Param.Field. (And that's true even if X is passed by reference, since the compiler could generate code that "knows" Param.Field won't change, since it can't tell whether the actual record will change behind its back.)

"Go Ada"

[Something like this kind of an Ada source repository has been discussed for a while on the #Ada IRC channel on Freenode. Some related posts to comp.lang.ada are quoted here. Note: This is still vapour-ware. —sparre]

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon, 2 Dec 2013 19:10:28 +0200
Subject: Re: CPAN style Ada repository
Newsgroups: comp.lang.ada

> [...] Jenkins server for building and testing Ada projects. That might be a good place to work from. No need to pick specific version control repositories or anything. Just publish and demonstrate working build scripts for your Ada projects.

It was me. The Jenkins server(s) is up at http://build.ada-language.com/

I have access to three different compilers, GNAT, Janus/Ada, and Irvine ICCAda, so I have setup them to compile various Ada projects automatically in multiple environments.

However, unfortunately, the distributed build results are not updated/reported currently because of Jenkins bug https://issues.jenkins-ci.org/browse/JENKINS-20067

Once that is fixed or once I find time to switch to another CI system, the build reports will be updating again.

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Wed, 11 Dec 2013 21:49:34 +0100
Subject: Re: CPAN style Ada repository
Newsgroups: comp.lang.ada

> Would there be interest in a Perl CPAN style Ada repository?

This might seem like a good idea, until one starts to realize the implications.

I assume that you would want this repository to be usable on many different operating systems, and maybe with different compilers (because if you were planning to target only GNAT and Debian for example, then you'd simply make Debian packages instead of proposing a new repository). The Ada language itself is quite portable between operating systems and compilers, but how to get the Ada code compiled and installed is quite another story. There is no standard for how to invoke a compiler or how to link to libraries, no standard set of compiler options and so on. Different operating systems have different commands for making directories and copying files, vastly different filesystem layouts, and even differences in pathname syntax. GNAT project files do only parts of the job, and are specific to GNAT as far as I know.

CPAN has it easy by comparison. There is only one Perl interpreter (probably because the language is such a hideous mess that it's impossible to write a compatible second implementation), so they don't need to worry about different compilers.

I recommend packaging for one of the existing distributions instead. Come join us in Fedora, Debian or some other distribution of your choice. Version control systems, bug trackers, build servers, mirrors and packaging standards are already there for you (at least in Fedora), and the packages will be just as readily available to users as any other package.

It may seem like duplicated effort to package the same software multiple times for different distributions, but it's actually not so bad. Packaging for one operating system is easier than packaging for many of them at once, so it's several smaller efforts instead of one big effort. I took part in the Gnuada project at Sourceforge for a while. There we tried to make RPM packages that could be built for both Suse and Fedora. Only two target platforms, very similar and based on the same package manager, and even that was enough to cause problems.

One thing that would help considerably, and that would be surmountable, would be if developers of free Ada software could agree on some conventions for how makefiles should be written. Free projects usually have build systems made of makefiles and Gnat project files, but most of them are too inflexible to adapt to different filesystem layouts, support staging or allow compiler options to be customized. Packaging is slow when packagers have to figure out how each makefile works and often patch makefiles to get them to meet packaging standards. We could get much more libraries and

programs packaged if developers would follow some conventions. Such conventions would need to work for mixed-language projects as well as pure Ada projects, and for both Gnatmake and GPRbuild.

I recommend using the Make variable names from the GNU Coding Standards (https://www.gnu.org/prep/standards/html_node/Makefile-Conventions.html) and extending that with Ada-specific variables in the same style.

Developers, please support LDFLAGS for linker options, and support CFLAGS if there is C code in your project. Stick to this naming scheme and use "ADAFLAGS" for Ada compiler options, "GNATBINDFLAGS" for gnatbind options, and so on.

Support the GNU standard directory variables so that your software can be installed in different systems with different filesystem layouts: prefix, exec_prefix, bindir, libdir, libexecdir and all the others as appropriate for the types of files that your project installs. Extend with "gprdir" for GNAT project files and "alidir" for ALI files. Install ALI files in a library-specific subdirectory of alidir, just like source files go in a subdirectory of includedir. Support DESTDIR so that packagers can install to a staging directory and don't have to build packages as root.

Writing such a flexible makefile is of course nontrivial work that you don't want to do over and over. You can avoid most of the work by using Comfignat (https://www.rombobjörn.se/Comfignat/). Comfignat gives you all of the above, except that limitations in Gnatmake's and GPRbuild's command line syntax prevent it from automatically supporting LDFLAGS and GNATBINDFLAGS when libraries are built.

This approach covers only those operating systems that are Unix-like enough to have Make and basic commands such as cp and mkdir, but that should include OS X, and hopefully even Windows with Cygwin or MinGW is Unix-like enough, so it's a decent set of target platforms. (And none of the above prevents you from also supporting some other platform by other means.)

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Thu, 12 Dec 2013 09:23:05 +0100
Subject: Re: CPAN style Ada repository
Newsgroups: comp.lang.ada

> Would there be interest in a Perl CPAN style Ada repository?

Something like it, yes.

I have earlier discussed the subject with Thomas Løcke and Kim Rostgaard Christensen in terms of the tools and infrastructure provided with the Go programming language from Google.

Step one must be to decide what the purpose of the project is. Some possibilities:

a) Make it easier for newcomers to Ada to get started.

b) Make it easier to find and install Open Source libraries and applications written in Ada.

c) Document how (and how well) Open Source libraries and applications written in Ada build and run on various platforms and with various compilers.

d) Advertise the existence of other Ada compilers than GNAT.

e) Survive, grow, and encompass all published Open Source Ada source texts.

I know that not all contributors to this thread weigh these purposes equally, but I think they all have value for the Ada community as a whole.

Based on these objectives I propose:

1) Make a "first download" package, which provides (or downloads or validates the existence of) a compiler and a client (developer) tool.

(Like for Go. A step towards objective a.)

2) Besides the "first download", the system provides "projects", which may be libraries, applications or a mix of both. (A "library" is basically a project without any non-test executables.) The collection of projects can be queried and downloaded through the client tool.

(A step towards objectives a and b.)

3) All projects are required to have some built-in tests. As a minimum there should be test applications which ensure that all compilation units in the project sources are compiled.

(A step towards objective c.)

4) Support multiple compilers. Tero Koskinen has both GNAT, Irvine ICCAda and Janus/Ada on his <http://build.ada-language.com/> site, so it should be possible.

(A step towards objectives c and d.)

5) Make it easy for developers to submit new projects to the system.

(A step towards objective e.)

Refining the proposals above:

6) Each project needs some build rules. As we want to support multiple compilers (proposal 4), it makes sense to have a very simple (proposal 5) build rule format, which then can be compiled to build rules for the various supported compilers.

The minimum requirements for the build rules might be as simple as three lists:

- Which projects this project depends on.

- Which applications are to be generated by this project.

- Which test programs are included in the project.

7) If an up-stream project is parameterised (or uses different sources for different hosts/compilers/targets), our view of it is multiple unparameterised projects. This will allow for simpler build rules (proposal 5 and 6) and at the same time simplify testing (objective c and proposal 3)

The following is a collection of various ideas for the project on a more practical level. These ideas are not (yet) tied properly to the objectives listed above.

General:

- Source-only projects - Skip the whole problem of generating dynamic libraries. Not many systems will have more than one application running using large parts of the same Ada library.

- Use ISO dates plus a single latin letter as version identifiers for projects. This will allow about one version per hour per project. Is this an unreasonable limit?

- Project naming: As Ada identifiers with the extra constraint that they can't end in "_" & Possible_Version_Identifier or "_" & Possible_Version_Identifier & "_test". (Other constraints?) This is to allow GPR files to reference either the "head" version of a project or a specific revision.

- Project parameters:

  + Build rules.

  + Version control link.

  + Provides: API ID's.

  + Dependencies: Both other projects in the system, API ID's _and_ operating system specific dependencies.

  + (plus revision information)

Client (developer tool):

- Keep track of which projects the developer has fetched explicitly (and which specific versions have been requested, where that is relevant)

- Keep track of the dependencies of the installed projects, and make sure they are fulfilled.

- When updating; only pull the newest versions of previously requested projects, when the developer hasn't asked for a specific version or has asked for "head".

- The Go command-line tool has (at least) the following operations: get, build, test, run, install. I'm not sure I care about "run", but "drop" (sort-of "uninstall") and "select-compiler" (as we want to support more than one) would be nice additions.

Service:

- Generate new project versions from upstream commits automatically.
- Test new project versions automatically as they are generated.
- Only release project versions which pass the tests on at least one host-compiler-target combination.
- It would be nice if it automatically could test which revisions of the dependencies of a project result in a working (test-passing) system.
- Should we host copies of the sources of the projects as a part of the service? J. Kimball doesn't like to do it, but I like the idea of fetching sources and build rules from a single location.

PS: I like a name like "go Ada" better than "something-CPAN".

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Fri, 13 Dec 2013 19:38:18 +0100
Subject: Re: CPAN style Ada repository
Newsgroups: comp.lang.ada

> I would leave "install" to the packagers; upstream Makefiles should just build in place.

I disagree with that for the following reasons:

- The directory variables aren't used only for copying files in the installation step. Some of these pathnames sometimes need to be embedded in programs so that they'll know where to find data files or other programs.
- Some of the directory variables also need to be embedded in libraries' usage project files. Packagers will have to patch usage projects if the upstream build system doesn't configure them correctly, and patches are much more of a maintenance burden than command line parameters are.
- Libraries need some source files installed, but usually not all of the source files. At least some specifications are always needed, but sometimes bodies are also needed, if they contain generic units or inline subprograms. Manually keeping track of which files are needed is difficult, especially for a packager who isn't also an upstream developer, but the GNAT builders know which files are needed and can install them for you.
- Packagers aren't the only ones who will build your software. Users who download the source tarball and install locally won't like to dig files out of the source tree manually and edit project files.

> [...] sounds like an Ada-style alternative to automake, which is very welcome.

Yes, that's a good comparison. The purpose of Comfignat is similar to that of Automake, but the way it works is somewhat different. You import a generic makefile instead of generating a makefile, and it delegates dependency tracking to GPRbuild or Gnatmake.

Reason for Ada.Strings.Bounded not Being Pure

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 5 Dec 2013 14:17:05 -0600
Subject: Re: Reason for 'Ada.Strings.Bounded' not being declared 'pragma Pure' ?
Newsgroups: comp.lang.ada

We re-analyzed all of the existing packages for Ada 2005, and changed the categorization of some of them. The details can be found in AI95-0362-1 (http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ais/ai-00362.txt).

There is a listing of every predefined package in that AI. Here's the entry for Bounded strings:

Ada.Strings.Bounded -- A.4.4; Preelaborate

This package contains no state, no dependence on non-pure units, no other items that prevent the package from being pure, and does not declare any types that would be a problem for Annex E, so it could be declared pure.

But it's large and complex, and many of the operations are not conceptually pure (they do in-place updates), so no change is recommended.

This admittedly does not seem very satisfying. We didn't redo this exercise for Ada 2012, the only change we made was to make Stream_IO preelaborated so that loggers and the like can be written. (It's not practical to make the full Text_IO preelaborated [the obvious approach is not task-safe], and there was no agreement on the contents of a preelaborable subset.) I suppose you could send a request to reconsider this to Ada-Comment (but it would probably have to wait until the next Standard, whenever that is).

Someone asked about Ada.Tags. The entry for it says:

Ada.Tags -- 3.9; not categorized

Package Tags has state, so it cannot be pure. That state is generally either set up at link-time (before elaboration) or during the elaboration of tagged types (that is, during the elaboration of other units). In either case, no complex state need be initialized at elaboration time. Thus, this package can be Preelaborated.

(The "state" that is talked about here is the table of internal tag <=> external tag mappings. Distributing that could be a significant overhead.) Making it Pure is not practical.

From: Brad Moore <brad.moore@shaw.ca>
Date: Thu, 05 Dec 2013 20:03:09 -0700
Subject: Re: Reason for 'Ada.Strings.Bounded' not being declared 'pragma Pure' ?
Newsgroups: comp.lang.ada

> [...]

Actually, this wasn't the only related change in Ada 2012. We also allowed Remote_Types packages and Remote_Call_Interface packages to depend on preelaborated packages, if that dependency is via a private with clause.

See http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05/ai05-0206-1.txt

This means that you can build abstractions that use Ada.Bounded_String (Or Ada.Tags for that matter), so long as they are not used in the visible part of a Remote_Types or Remote_Call_Interface package. eg.

```
private with Ada.Strings.Bounded;

package RT is
  pragma Remote_Types;
  type W is private;

  procedure Set (Item  : in out W;
                 Value : in     String);
  function Get (Item : W) return String;

private
  package My_String is new
    Ada.Strings.Bounded.
          Generic_Bounded_Length
          (Max => 100);
  type W is
    record
      D : My_String.Bounded_String;
    end record;

  function Get (Item : W) return String is
          (My_String.To_String (Item.D));
end RT;

package body RT is
  procedure Set (Item  : in out W;
                 Value : in     String) is
  begin
    My_String.Set_Bounded_String
          (Target => Item.D,
           Source => Value);
  end Set;
end RT;
```

Freezing Rules

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 17 Jan 2014 17:23:55 -0600
Subject: Re: 'Protected' abstract subprograms
Newsgroups: comp.lang.ada

Robert A Duff wrote:

> The freezing rules make my brain hurt. Even though I had a hand in writing them! ;-)
>

> A better-designed language would not have anything like freezing rules.

That's a fascinating assertion from the guy that's responsible for most the AARM notes describing the freezing rules. Especially as you guys pretty much redesigned that area in Ada 95 -- you essentially created a whole new language design for it. It makes me wonder how a language could be better designed and not "have anything like freezing rules".

After all, a compiler has to have a known representation for types and objects at some point, certainly before code generation. If one is going to support some sort of separate compilation (especially *safe* separate compilation), freezing rules or something like them seems mandatory. [There is an argument to be made that a truly modern language doesn't need to support separate compilation at the language level, given that it is now practical to delay compilation until bind time. But that seems awfully radical and would seem to put an upper limit on the sizes of programs that could be written in the language. I don't think that's what you meant.]

Ada's freezing rules are far more detailed than absolutely necessary, but the obvious way to get simplify them would be to require that all types are declared before all objects (a-la Pascal) -- and we have evidence that's too inflexible. There don't seem to be any obvious way to simplify them if objects can be declared anywhere.

So I have to wonder what sort of "well-designed language" you have in mind.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Sun, 19 Jan 2014 16:07:41 -0500
Subject: Re: 'Protected' abstract subprograms
Newsgroups: comp.lang.ada

> [...]

Not really. Ada 83 had "forcing occurrences" regarding rep clauses. And I think a bunch of similar rules regarding premature use of private types. IIRC, we decided that these two sets of rules should be combined. Also, the rules were buggy, and we wanted to fix them.

The actual rules in Ada 95 are almost the same as in Ada 83. They don't look the same, because the wording was changed a lot. But the things that are legal and illegal in Ada 83 didn't change much in Ada 95.

In other words: Don't mix up "the Ada language" with "the words we use to describe the Ada Language in the RM". The latter can change without changing the former. And in this case, the latter changed a lot while the former changed a little.

>... It makes me wonder how a language could be better designed and not "have anything like freezing rules".

Well, I'm too lazy to give all the details, but here's one key point:

It is obvious[*] that module specs should not be elaborated. They should be purely a compile-time description of the interface, and should not exist at all at run time.

[*] I'm just kidding about "obvious". It took me years to figure that out. But having done so, it's obvious (to me).

Another point: Something like Ada's aspect clauses are better than pragmas and separate syntax for rep clauses. That's because aspect clauses are physically attached to the declaration, so there's less of an issue about when things are evaluated. Also, you don't have to refer to the thing by name; you're just saying "this thing has so an so properties". Every time you refer to something by name, you put a (slight) burden on the person reading the code, who has to match up uses with declarations.

> After all, a compiler has to have a known representation for types and

> objects at some point, [...]

No, that's not what I meant.

> Ada's freezing rules are far more detailed than absolutely necessary, but

> the obvious way to get simplify them would be to require that all types are

> declared before all objects (a-la Pascal) [...]

I hate that aspect of Pascal. I wouldn't call Pascal a well-designed language, although it is much better than many others in many respects.

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 20 Jan 2014 19:21:54 -0600
Subject: Re: 'Protected' abstract subprograms
Newsgroups: comp.lang.ada

> [...]

Right. Aspect clauses eliminate quite a bit of the need for freezing rules. (Although we ended up using them to describe the semantics of aspect clauses, that was mainly historical in nature -- it would have been better to wait until the end of the unit for those determinations, but that would have not allowed various things legal in Ada.)

I believe your point that a purely compile-time description would eliminate freezing. I'm not convinced that such a restriction would really be usable -- I suppose it depends on what could actually be described that way. (Personally, I find Ada interfaces useless; I much prefer package specifications for abstraction. I'm not sure if that carries over to your idea.)

Anyway, not particularly relevant for Ada, since we're surely not reducing what is allowed in a package specification.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Tue, 21 Jan 2014 09:35:42 -0500
Subject: Re: 'Protected' abstract subprograms
Newsgroups: comp.lang.ada

> Of course, I know this, but I don't really think that this is relevant. You ("you" meaning Ada 9x team here since I don't know for sure who did what)

I did most of the work on chapter 13. I did the initial work on freezing rules, and I think I'm responsible for the term "freezing". We modified them over and over, trying to fix bugs, and trying to clarify. At some point, I got frustrated, and told Tucker I'm sick of those rules, you (Tucker) please work on them, and I'll finish up chapter 3 (which Tucker had been working on at the time). So Tucker is responsible for the final wording of the freezing rules in Ada 95.

> could have changed the language more (since you guys had already decided to change the wording drastically), but you didn't. I would say that was mainly because it wasn't really possible to change the language further without changing it's philosphosy.

Not philosophy, so much as compatibility.

>...After all, you did make a number of cases illegal that were legal in Ada 83.

Yes, but pretty obscure cases. I don't remember the details, and I'm too lazy to look them up, but I recall cases in Ada 83 where ARG had ruled "X is legal" for some feature X. But if you do X it's guaranteed to raise an exception during elaboration (so for a library package, the program is guaranteed to fail every time). Also, in order to generate correct code for X, the compiler has to KNOW that it's going to raise an exception; otherwise it would generate code that would follow dangling pointers and the like.

Clearly, making such an X illegal is an acceptable incompatibility.

> Right. Aspect clauses eliminate quite a bit of the need for freezing rules.

Yeah, too bad they weren't invented in the late 1970's, in time for Ada 83.

> I believe your point that a purely compile-time description would eliminate freezing. I'm not convinced that such a restriction would really be usable -- I suppose it depends on what could actually be described that way.

I'm convinced. ;-) In fact, my language design allows some useful things that Ada does not. For example, in Ada:

```
package Sequences is
   type Sequence is private;
   function Make_Sequence
     (Length: Natural) return Sequence;
   Empty_Sequence : constant :=
         Make_Sequence (Length => 0);
   -- Wrong!
   ...
private
   ...
end Sequences;
```

That won't work, and it's annoying. In my language, it works fine, because Make_Sequence is called during elaboration of the BODY of Sequences.

But that's not Ada, and it's not even possible to compatibly change Ada in this way, so I should stop talking about off-topic stuff. ;-)

> [...]

In any case, you're right that SOME sort of rules are needed. You need to prevent circular things like X is of type T, and the Size of T depends on the value of X. I just don't think it needs to be anywhere near as complicated as the freezing rules.

Difference in Sizes of Integer Types

From: user3261820 <http://stackoverflow.com/users/3261820/user3261820>
Date: Sun Feb 2 2014
Subject: Ada types size difference
URL: http://stackoverflow.com/questions/21506182/ada-types-size-difference

I have this Ada program:

```
with Ada.Text_IO, Ada.Integer_Text_IO;
use  Ada.Text_IO, Ada.Integer_Text_IO;

procedure Test is
  type MY_TYPE is new Integer
                          range 1 .. 20;
  subtype MY_TYPE2 is MY_TYPE
                          range 5 .. 6;
  C : MY_TYPE  := 10;
  F : MY_TYPE2 := 6;
begin
  Put (Integer (C'Size));
end Test;
```

and when I run it I get 32. If I replace

```
type MY_TYPE is new Integer
                          range 1..20;
```

with

```
type MY_TYPE is range 1..20;
```

I get 8. What is the difference between the two declarations?

From: Brian Drummond <brian@shapes.demon.co.uk>
Date: Sun Feb 2 2014
URL: http://stackoverflow.com/questions/21506182/ada-types-size-difference

You are allowing the compiler to choose the sizes for these different type declarations, and it is picking the size of your INTEGER type according to the size of its base type (INTEGER).

You have control over the sizes of these types : if you rewrite the first declaration as

```
type MY_TYPE is new Integer range 1..20;
for MYTYPE'SIZE use 8;
```

you should get an 8-bit MY_TYPE.

```
for MYTYPE'SIZE use 5;
```

ought to pack MYTYPE into 5 bits (as I understand it, a compiler is permitted to reject this with an explicit error, or generate correct code, but NOT to accept it and generate garbage.)

Why would you want to pack MYTYPE into 5 bits? One reason is if it's used as a component of a record : that leaves room for 3 more components in a single byte, as long as they are booleans and their SIZE attribute is 1!

This may look like extreme packing, but it's actually quite common in embedded programming, where that record type matches the bits in a peripheral or I/O port. You would also specify the bit-level layout within the record, as in:

```
type Prescale is new Integer range 1..20;
for Prescale'SIZE use 8;

type Timer_Ctrl_Reg is record
  Scale: Prescale;
  Up: Boolean;
  Repeat : Boolean;
  Int_En : Boolean;
end record;

for Timer_Ctrl_Reg use record
  Scale at 0 range 0 .. 4;
  Up at 0 range 5 .. 5;
  Repeat at 0 range 6 .. 6;
  Int_En at 0 range 7 .. 7;
end record;
```

at specifies the offset from the record base in "storage units" usually bytes or words : range specifies the bit positions within the storage unit.

No more dodgy bit masking and extraction to worry about!

On the other hand,

```
for MYTYPE'SIZE use 4;
```

ought to fail, as MYTYPE has more than 16 discrete values.

From: Keith Thompson <Keith.S.Thompson@gmail.com>
Date: Sun Feb 2 2014
URL: http://stackoverflow.com/questions/21506182/ada-types-size-difference

This:

```
type MY_TYPE is new Integer range 1..20;
```

explicitly inherits MY_TYPE from Integer, which apparently is 32 bits on your system.

This:

```
type MY_TYPE is range 1..20;
```

leaves it up to the compiler to decide how to represent MY_TYPE. The result is implementation-specific; apparently your compiler chooses to implement it as an 8-bit integer type.

Aborting a Process That Uses a Protected Object That Requeues

From: ArthurTheLearner <http://stackoverflow.com/users/2643554/arthurthelearner>
Date: Thu Feb 20 2014
Subject: Aborting a process in Ada that uses a protected object that requeues
URL: http://stackoverflow.com/questions/21907632/

I'm experiencing some troubles with my program.

I have a process that calls a function (Take_Job) that is supposed to remain blocked until a time (MINIMUM_WAIT) passes. If it doesn't happen that way, a message informing of this situation will appear.

```
for Printer_ID in Type_Printer_ID loop
  select
    delay MINIMUM_WAIT
    Pragma_Assert (True, "");
  then abort
    Take_Job (Controller,
              Printer_ID,
              Max_Tonner,
              Job,
              Change_Tonner);
    Pragma_Assert (False, "Testing of
              Take_Job hasn't been successful.
              It should have remained blocked.");
  end select;
end loop;
```

The function Take_Job calls to an entry in a protected object:

```
procedure Take_Job
          (R: in out Controller_Type;
           Printer: in Type_Printer_ID;
           Siz: in Typo_Volume;
           Job : out Typo_Job;
           Excep_Tonner : out Boolean) is
begin
  R.Take_Job (Printer, Siz, Job,
                    Excep_Tonner);
end Take_Job;
```

Where "R" is the protected object.

The following code is the entry of the protected object. Actually, the "when" condition is True because I need to check some stuff with the parameters of the entry. Since Ada doesn't allow me to do that, I copy the parameters inside the protected object and call to a "delayed entry", then in the "delayed entry" I will

make sure that the condition is met before proceeding.

```
entry Take_Job
            (Printer_ID : in Type_Printer_ID;
            Remaining : in  Type_Volume;
            Job : out Type_Job;
            exceptionTonner : out Boolean)
when True is
begin
  Copy_Remaining (Printer_ID) :=
                        Remaining;
  requeue Take_Job_Delayed (Printer_ID);
end Take_Job;
```

Let's see the "delayed entry" code:

```
entry Take_Job_Delayed
      (for J in Type_Printer_ID)
          (Printer_ID : in Type_Printer_ID;
            Remaining : in Type_Volume;
            Job : out Type_Job;
            exceptionTonner : out Boolean)
when False is -- I've done this on purpose
begin
  null; -- Actually, there would be a lot of
       -- code here
end Take_Job_Delayed;
```

Let's say that my goal is to pass the MINIMUM_WAIT and run the "Pragma_Assert(True, "")". If I put the "when" condition of Take_Job to "False", then everything works fine. Take_Job is never accepted and the Pragma_Assert will be executed. If I set it to "True" and the "when" condition of Take_Job_Delayed to "False", I don't get the same effect and the process gets blocked and neither of the Pragma_Asserts will be executed.

Why? It looks like the problem is in the "requeue" or somewhere near that, but why is this happening?

From: Simon Wright <simon@pushface.org>
Date: Thu Feb 20 2014
URL: http://stackoverflow.com/questions/21907632/

You need to do the requeue with abort;

```
entry Take_Job_Delayed
      (for J in Type_Printer_ID)
          (Printer_ID : in Type_Printer_ID;
            Remaining : in Type_Volume;
            Job : out Type_Job;
            exceptionTonner : out Boolean)
when True is
begin
  Copy_Remaining(Printer_ID) := Remaining;
  requeue Take_Job_Delayed(Printer_ID)
with abort;
end Take_Job;
```

because otherwise the opportunity to abort the entry call has been lost. There are details in ARM 9.5.4[1], and a more understandable explanation in Burns & Wellings, “Concurrency in Ada”.

[1] http://www.adaic.org/resources/add_content/standards/12rm/html/RM-9-5-4.html

Rationale for Encapsulation of Dynamically Dispatching Operations

From: coredump <http://stackoverflow.com/users/124319/coredump>
Date: Mon Feb 24 2014
Subject: Rationale behind Ada encapsulation of dynamically dispatching operations (primitives)
URL: http://stackoverflow.com/questions/21991312/

In Ada, Primitive operations of a type T can only be defined in the package where T is defined. For example, if a Vehicles package defines Car and Bike tagged record, both inheriting a common Vehicle abstract tagged type, then all operations than can dispatch on the class-wide Vehicle'Class type must be defined in this Vehicles package.

Let's say that you do not want to add primitive operations: you do not have the permission to edit the source file, or you do not want to clutter the package with unrelated features.

Then, you cannot define operations in other packages that implicitly dispatches on type Vehicle'Class. For example, you may want to serialize vehicles (define a Vehicles_XML package with a To_Xml dispatching function) or display them as UI elements (define a Vehicles_GTK package with Get_Label, Get_Icon, ... dispatching functions), etc. The only way to perform dynamic dispatch is to write the code explicitly; for example, inside Vehicle_XML:

```
if V in Car'Class then
  return Car_XML (Car (V));
else
  if V in Bike'Class then
    return Bike_XML (Bike (V));
  else
    raise Constraint_Error
      with "Vehicle_XML is only defined for
            Car and Bike."
end if;
```

(And a Visitor pattern defined in Vehicles and used elsewhere would work, of course, but that still requires the same kind of explicit dispatching code.)

My question is then:

Is there a reason why operations dynamically dispatching on T are restricted to be defined in the defining package of T?

Is this intentional? Is there some historical reasons behind this?

From: ajb <http://stackoverflow.com/users/2464386/ajb>
Date: Mon Feb 24 2014
Subject: Rationale behind Ada encapsulation of dynamically dispatching operations (primitives)
URL: http://stackoverflow.com/questions/21991312/

I can think of several reasons:

(1) Your example has Car and Bike defined in the same package, both derived from Vehicles. However, that's not the "normal" use case, in my experience; it's more common to define each derived type in its own package. (Which I think is close to how "classes" are used in other compiled languages.) And note also that it's not uncommon to define new derived types afterwards. That's one of the whole points of object-oriented programming, to facilitate reuse; and it's a good thing if, when designing a new feature, you can find some existing type that you can derive from, and reuse its features.

So suppose you have your Vehicles package that defines Vehicle, Car, and Bike. Now in some other package V2, you want to define a new dispatching operation on a Vehicle. For this to work, you have to provide the overriding operations for Car and Bike, with their bodies; and assuming you are not allowed to modify Vehicles, then the language designers have to decide where the bodies of the new operation have to be. Presumably, you'd have to write them in V2. (One consequence is that the body that you write in V2 would not have access to the private part of Vehicles, and therefore it couldn't access implementation details of Car or Bike; so you could only write the body of that operation if terms of already-defined operations.) So then the question is: does V2 need to provide operations for all types that are derived from Vehicle? What about types derived from Vehicle that don't become part of the final program (maybe they're derived to be used in someone else's project)? What about types derived from Vehicle that haven't yet been defined (see preceding paragraph)? In theory, I suppose this could be made to work by checking everything at link time. However, that would be a major paradigm change for the language. It's not something that could be easily. (It's pretty common, by the way, for programmers to think "it would be nice to add feature X to a language, and it shouldn't be too hard because X is simple to talk about", without realizing just what a vast impact such a "simple" feature would have.)

(2) A practical reason has to do with how dispatching is implemented. Typically, it's done with a vector of procedure/function pointers. (I don't know for sure what the exact implementation is in all cases, but I think this is basically the case for every Ada compiler as well as for C++ and Java compilers, and probably C#.) What this means is that when you define a tagged type (or a class, in other languages), the compiler will set up a vector of pointers, and based on how many operations are defined for the type, say N, it will reserve

slots 1..N in the vector for the addresses of the subprograms. If a type is derived from that type and defines overriding subprograms, the derived type gets its own vector, where slots 1..N will be pointers to the actual overriding subprograms. Then, when calling a dispatching subprogram, a program can look up the address in some known slot index assigned to that subprogram, and it will jump to the correct address depending on the object's actual type. If a derived type defines new primitive subprograms, new slots are assigned N+1..N[2], and types derived from that could define new subprograms that get slots N[2]+1..N[3], and so on.

Adding new dispatching subprograms to Vehicle would interfere with this. Since new types have been derived from Vehicle, you can't insert a new area into the vector after N, because code has already been generated that assumes the slots starting at N+1 have been assigned to new operations derived for derived types. And since we may not know all the types that have been derived from Vehicle and we don't know what other types will be derived from Vehicle in the future and how many new operations will be defined for them, it's hard to pick some other location in the vector that could be used for the new operations. Again, this could be done if all of the slot assignment were deferred until link time, but that would be a major paradigm change, again.

To be honest, I can think of other ways to make this work, by adding new operations not in the "main" dispatch vector but in an auxiliary one; dispatching would probably require a search for the correct vector (perhaps using an ID assigned to the package that defines the new operations). Also, adding interface types to Ada 2005 has already complicated the simple vector implementation somewhat. But I do think this (i.e. it doesn't fit into the model) is one reason why the ability to add new dispatching operations like you suggest isn't present in Ada (or in any other compiled language that I know of).

From: Jacob Sparre Andersen <jacob@jacob-sparre.dk>
Date: Tue Feb 25 2014
Subject: Rationale behind Ada encapsulation of dynamically dispatching operations (primitives)
URL: http://stackoverflow.com/questions/21991312/

Without having checked the rationale for Ada 95 (where tagged types were introduced), I am pretty sure the freezing rules for tagged types are derived from the simple requirement that all objects in T'Class should have all the dispatching operations of type T.

To fulfill that requirement, you have to freeze type and say that no more dispatching operations can be added to type T once you:

- Derive a type from T, or
- Are at the end of the package specification where T was declared.

If you didn't do that, you could have a type derived from type T (i.e. in T'Class), which hadn't inherited all the dispatching operations of type T. If you passed an object of that type as a T'Class parameter to a subprogram, which knew of one more dispatching operation on type T, a call to that operation would have to fail. - We wouldn't want that to happen.

Conference Calendar

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

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

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

2014

April 05-13 17th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2014), Grenoble, France. Events include: CC, International Conference on Compiler Construction; ESOP, European Symposium on Programming; FASE, Fundamental Approaches to Software Engineering; FOSSACS, Foundations of Software Science and Computation Structures; POST, Principles of Security and Trust; TACAS, Tools and Algorithms for the Construction and Analysis of Systems.

- April 07-11 20th **International Conference on Tools and Algorithms for the Construction and Analysis of Systems** (TACAS'2014). Topics include: specification and verification techniques; analytical techniques for real-time systems; analytical techniques for safety, security, or dependability; static and dynamic program analysis; abstraction techniques for modeling and verification; system construction and transformation techniques; tool environments and tool architectures; applications and case studies; etc.
- April 07-11 17th **International Conference on Fundamental Approaches to Software Engineering** (FASE'2014). Topics include: software engineering as an engineering discipline; specification, design, and implementation of particular classes of systems (embedded, distributed, ...); software quality (validation and verification of software using theorem proving, model checking, testing, analysis, refinement methods, metrics, ...); model-driven development and model transformation (design and semantics of domain-specific languages, consistency and transformation of models, ...); software evolution (refactoring, reverse and re-engineering, ...); etc.
- ☺ April 12 **Programming Language Approaches to Concurrency and communication-cEntric Software** (PLACES'2014). Topics include: the general area of programming language approaches to concurrency, communication and distribution, such as design and implementation of programming languages with first class support for concurrency and communication; concurrent data types, objects and actors; verification and program analysis methods for concurrent and distributed software; high-level programming abstractions addressing security concerns in concurrent and distributed programming; multi- and many-core programming models, including methods for harnessing GPUs and other accelerators; integration of sequential and concurrent programming techniques; programming language approaches to web services; etc.
- April 12 11th **International Workshop on Formal Engineering approaches to Software Components and Architectures** (FESCA'2014). Topics include: modelling formalisms, temporal properties and their formal verification, interface compliance and contractual use of components, static and dynamic analysis, industrial case studies and experience reports, etc.

April 07-10 23rd **Australasian Software Engineering Conference** (ASWEC'2014), Sydney, Australia. Topics include: dependable and secure computing; domain-specific models and languages, and model driven development; engineering/operating large-scale distributed systems; formal methods; legacy systems, software maintenance and reverse engineering; modularisation techniques; open source software development; programming languages and techniques; quality assurance; real-time and embedded

software; software analysis; software architecture, design and patterns; software processes and quality; software risk management; software reuse and product lines; software security, safety and reliability; software verification and validation; standards; etc.

April 13-16 — 22nd **High Performance Computing Symposium** (HPC'2014), Tampa, Florida, USA. Topics include: high performance/large scale application case studies, multicore and many-core computing, distributed computing, tools and environments for coupling parallel codes, high performance software tools, etc.

April 22-26 — 13th **International Conference on Modularity** (Modularity'2014), Lugano, Switzerland. Topics include: varieties of modularity (generative programming, aspect orientation, software product lines, components; ...); programming languages (support for modularity related abstraction in: language design; verification, contracts, and static program analysis; compilation, interpretation, and runtime support; formal languages; ...); software design and engineering (evolution, empirical studies of existing software, economics, testing and verification, composition, methodologies, ...); tools (refactoring, evolution and reverse engineering, support for new language constructs, ...); applications (distributed and concurrent systems, middleware, cyber-physical systems, ...); complex systems; etc.

April 23-25 — 27th **Conference on Software Engineering Education and Training** (CSEET'2014), Klagenfurt, Austria.

April 23-25 — XVII **Ibero-American Conference on Software Engineering** (CIbSE'2014), Pucón, Chile. Topics include: formal methods applied to software engineering; languages, methods, processes, and tools; model-based engineering; proof, verification, and validation; quality, measurement, and assessment of products and processes; reverse engineering and software system modernization; software development paradigms; software evolution and maintenance; software product families and variability; software reuse; reports on benefits derived from using specific software technologies; quality measurement; experience management; systematic reviews and evidence-based software engineering; etc.

Apr 29 - May 05 — 6th **NASA Formal Methods Symposium** (NFM'2014), NASA Johnson Space Center, Houston, Texas, USA. Topics include: identifying challenges and providing solutions to achieving assurance in mission- and safety-critical systems; static analysis; model-based development; applications of formal methods to aerospace systems; correct-by-design and design for verification techniques; techniques and algorithms for scaling formal methods, e.g. abstraction and symbolic methods, compositional techniques, parallel and distributed techniques; application of formal methods to emerging technologies; etc.

May 12-16 — 19th **International Symposium on Formal Methods** (FM'2014), Singapore. Topics include: interdisciplinary formal methods (techniques, tools and experiences demonstrating formal methods in interdisciplinary frameworks); formal methods in practice (industrial applications of formal methods, experience with introducing formal methods in industry, tool usage reports, etc); tools for formal methods (advances in automated verification and model-checking, integration of tools, environments for formal methods, etc); role of formal methods in software and systems engineering (development processes with formal methods, usage guidelines for formal methods, method integration, qualitative or quantitative improvements); theoretical foundations (all aspects of theory related to specification, verification, refinement, and static and dynamic analysis).

- May 13 — 3rd **International Workshop on Engineering Safety and Security Systems** (ESSS'2014). Topics include: methods, techniques and tools for system safety and security; methods, techniques and tools for analysis, certification, and debugging of complex safety and security systems; case studies and experience reports on the use of formal methods for analyzing safety and security systems; etc.

May 13-16 — 10th **European Dependable Computing Conference** (EDCC'2014), Newcastle upon Tyne, UK. Topics include: hardware and software architecture of dependable systems, safety critical systems, embedded and real-time systems, impact of manufacturing technology on dependability, testing and validation methods, privacy and security of systems and networks, etc.

☺ May 19-23 — 28th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2014), Phoenix, Arizona, USA. Topics include: parallel and distributed algorithms, applications of parallel and distributed computing, parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, parallel programming paradigms, programming environments and tools, etc.

☺ May 19 19th **International Workshop on High-Level Parallel Programming Models and Supportive Environments** (HIPS'2014). Topics include: all areas of parallel applications, language design, compilers, run-time systems, and programming tools; such as New programming languages and constructs for exploiting parallelism and locality; Experience with and improvements for existing parallel languages and run-time environments; Parallel compilers, programming tools, and environments; Programming environments for heterogeneous multicore systems; etc.

May 19 4th NSF/TCPP **Workshop on Parallel and Distributed Computing Education** (EduPar-14). Topics include: experience with incorporating Parallel and Distributed Computing (PDC) topics into core CS/CE courses; pedagogical tools, programming environments, and languages for PDC; etc.

☺ May 20 **International Workshop on Programming Models, Languages and Compilers** (PLC'2014). Topics include: programming models (thread and task based models, data parallel models, stream programming), programming environments for heterogeneous systems, compiler optimizations, runtime systems for multicore processors, applications and benchmarks, etc.

May 26-29 8th ACM **International Conference on Distributed Event-Based Systems** (DEBS'2014), Mumbai, India. Topics include: software systems, distributed systems, dependability, programming languages, security and software engineering, real-time analytics, embedded systems, enterprise application integration, etc. Deadline for submissions: April 5, 2014 (Doctoral Symposium papers, posters, demos).

☺ May 31- Jun 06 36th **International Conference on Software Engineering (**ICSE'2014), Hyderabad, India. Deadline for early registration: April 14, 2014.

May 31-Jun 01 11th **Working Conference on Mining Software Repositories** (MSR'2014). Topics include: mining of repositories across multiple projects; characterization, classification, and prediction of software defects based on analysis of software repositories; techniques to model reliability and defect occurrences; search techniques to assist developers in finding suitable components and code fragments for reuse; empirical studies on extracting data from repositories of large long-lived and/or industrial projects; mining execution traces and logs; etc.

June 01-07 **Software Engineering Education and Training Track** (SEET'2014). Topics include: new best practices for SEET, continuing education in the face of rapid technological change, ensuring graduated students meet new industry needs through the understanding of development practices for different environments, etc.

☺ June 03-05 **DAta Systems In Aerospace** (DASIA'2014), Warsaw, Poland.

June 03-06 9th **International Federated Conferences on Distributed Computing Techniques** (DisCoTec'2014), Berlin, Germany. Includes the COORDINATION, DAIS, and FMOODS & FORTE conferences. Deadline for early registration: May 5, 2014.

June 03-06 14th IFIP **International Conference on Distributed Applications and Interoperable Systems** (DAIS'2014). Topics include: all aspects of distributed applications and systems, throughout their lifecycle; design, architecture, implementation and operation of distributed computing systems, their supporting middleware, appropriate software engineering methods and tools, as well as experimental studies and practical reports; language-based approaches; parallelization; domain-specific languages; design patterns and methods; etc.

June 09-11 ACM SIGPLAN **Conference on Programming Language Design and Implementation** (PLDI'2014), Edinburgh, UK. Topics include: programming languages, their design, implementation, development, and use; innovative and creative approaches to compile-time and runtime technology, novel language designs and features, and results from implementations; language designs and extensions; static and dynamic analysis of programs; domain-specific languages and tools; type systems and program logics; checking or improving the security or correctness of programs; memory management; parallelism, both implicit and explicit; debugging techniques and tools; etc.

☺ June 12-13 ACM SIGPLAN/SIGBED **Conference on Languages, Compilers, and Tools for Embedded Systems** (LCTES'2014). Topics include: programming language challenges (features to exploit multicore architectures; features for distributed and real-time control

embedded systems; language capabilities for specification, composition, and construction of embedded systems; language features and techniques to enhance reliability, verifiability, and security; virtual machines, concurrency, inter-processor synchronization, and memory management; ...); compiler challenges (interaction between embedded architectures, operating systems, and compilers; support for enhanced programmer productivity; support for enhanced debugging, profiling, and exception/interrupt handling; ...); tools for analysis, specification, design, and implementation (distributed real-time control, system integration and testing, run-time system support for embedded systems, support for system security and system-level reliability, ...); etc.

June 16-20 26th **International Conference on Advanced Information Systems Engineering** (CAiSE'2014), Thessaloniki, Greece. Theme: "Information Systems Engineering in Times of Crisis". Topics include: methods, techniques and tools for IS engineering (models and software reuse; adaptation, evolution and flexibility issues; languages and models; variability and configuration; security; ...); innovative platforms, architectures and technologies for IS (model-driven architecture; component based development; distributed and open architecture; ...); etc.

June 23-25 26th ACM **Symposium on Parallelism in Algorithms and Architectures** (SPAA'2014), Prague, Czech Republic. Topics include: parallel and distributed algorithms; multi-core architectures; compilers and tools for concurrent programming; synergy of parallelism in algorithms, programming, and architecture; etc.

June 23-25 19th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2014), Uppsala, Sweden.

♦ June 23-27 19th **International Conference on Reliable Software Technologies - Ada-Europe'2014**, Paris, France. Sponsored by Ada-Europe, in cooperation with ACM SIGAda, SIGBED, SIGPLAN. Deadline for early registration: May 31, 2014.

☺ June 24-27 13th **International Symposium on Parallel and Distributed Computing** (ISPDC'2014), Porquerolles Island, France. Topics include: methods and tools for parallel and distributed programming, tools and environments for parallel program design/analysis, parallel programming paradigms and APIs, distributed software components, multi-agent systems, security and dependability, real-time distributed and parallel Systems, etc.

Jun 30- Jul 02 8th IEEE **International Conference on Software Security and Reliability** (SERE'2014), San Francisco, USA. Theme: "Software Quality Assurance". Topics include: security, reliability, availability, and safety of software systems; fault tolerance for software reliability improvement; validation, verification, and testing; software vulnerabilities; benchmark and empirical studies; etc.

☺ July 08-11 26th **Euromicro Conference on Real-Time Systems** (ECRTS'2014), Madrid, Spain. Topics include: all aspects of real-time systems, such as applications, hardware/software co-design, multicore and manycore architectures for real-time and safety, operating systems, run-time environments, software architectures, programming languages and compiler support, component-based approaches, distribution technologies, modelling and formal methods for design and analysis, safety, reliability, security and survivability; mixed critical systems, etc.

July 08 10th **International Workshop on Operating Systems Platforms for Embedded Real-Time Applications** (OSPERT 2014).

July 08 5th **International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems** (WATERS 2014).

July 15-19 33rd **Annual** ACM SIGACT-SIGOPS **Symposium on Principles of Distributed Computing** (PODC'2014), Paris, France.

☺ July 18-20 **GNU Tools Cauldron** 2014, Cambridge, UK. Topics include: gathering of GNU tools developers, to discuss current/future work, coordinate efforts, exchange reports on ongoing efforts, discuss development plans for the next 12 months, developer tutorials and any other related discussions.

July 18-22 26th **International Conference on Computer Aided Verification** (CAV'2014), Vienna, Austria. Topics include: theory and practice of computer-aided formal analysis methods for hardware and software systems.

July 21-25 — 38th **Annual International Computer Software and Applications Conference** (COMPSAC'2014), Västerås, Sweden. Topics include: software engineering, security and privacy, quality assurance and assessment, embedded and cyber-physical environments, etc. Deadline for submissions: April 8, 2014 (fast abstracts, posters, doctoral symposium papers).

July 21-25 — 10th **European Conference on Modelling Foundations and Applications** (ECMFA'2014), York, UK. Topics include: domain specific modelling languages and language workbenches; model reasoning, testing and validation; model transformation, code generation and reverse engineering; Model-Based Engineering (MBE) environments and tool chains; MBE for large and complex industrial systems; MBE for safety-critical systems; comparative studies of MBE methods and tools; etc.

July 21-25 — 4th **International Workshop on New Algorithms and Programming Models for the Manycore Era** (APMM'2014), Bologna, Italy. Topics include: parallelisation with appropriate programming models and tool support for multi-core and hybrid platforms; software engineering, code optimisation, and code generation strategies for parallel systems with multi-core processors; etc.

☺ Jul 28 - Aug 08 — 28th **European Conference on Object-Oriented Programming** (ECOOP'2014), Uppsala, Sweden. Topics include: all areas of object technology and related software development technologies, such as concurrent and parallel systems, distributed computing, programming environments, versioning, refactoring, software evolution, language definition and design, language implementation, compiler construction, design methods and design patterns, aspects, components, modularity, program analysis, type systems, specification, verification, security, real-time systems, etc.

August 04-07 — 19th IEEE **International Conference on Engineering of Complex Computer Systems** (ICECCS'2014), Tianjin, China. Topics include: verification and validation, security of complex systems, model-driven development, reverse engineering and refactoring, design by contract, agile methods, safety-critical & fault-tolerant architectures, real-time and embedded systems, tools and tool integration, industrial case studies, etc.

August 14-17 — **Symposium on Dependable Software Engineering: Theories, Tools and Applications** (SETTA'2014), Nanjing, China. Topics include: formal software engineering methods; formal aspects of engineering approaches to software and system quality; integration of formal methods into software engineering practice; formal methods for embedded, real-time, hybrid, and cyber-physical systems; formal aspects of security, safety, reliability, robustness, and fault-tolerance; model checking, theorem proving, and decision procedures; contract-based engineering of components, systems, and systems of systems; formal and engineering aspects of software evolution and maintenance; scalable approaches to formal system analysis and design; applications of formal methods and industrial experience reports; etc.

☺ August 25-29 — 20th **International European Conference on Parallel Computing** (Euro-Par'2014), Porto, Portugal. Topics include: all aspects of parallel and distributed computing, such as support tools and environments, scheduling, high-performance compilers, distributed systems and algorithms, parallel and distributed programming, multicore and manycore programming, theory and algorithms for parallel computation, etc.

August 27-29 — 40th **Euromicro Conference on Software Engineering and Advanced Applications** (SEAA'2014), Verona, Italy. Topics include: information technology for software-intensive systems.

August 29-31 — 9th **International Conference on Software Engineering and Applications** (ICSOFT-EA'2014), Vienna, Austria. Topics include: software integration, software testing and maintenance, model-driven engineering, software quality, software and information security, formal methods, programming languages, parallel and high performance computing, software metrics, agile methodologies, risk management, quality assurance, certification, etc. Deadline for submissions: May 21, 2014 (position papers).

September 01-03 — 8th **International Symposium on Theoretical Aspects of Software Engineering** (TASE'2014), Changsha, China. Topics include: theoretical aspects of software engineering, such as specification and verification, program analysis, model-driven engineering, aspect and object orientation, embedded and real-time systems, component-based software engineering, software safety, security and reliability, reverse engineering and software maintenance, etc.

September 01-05 — 12th **International Conference on Software Engineering and Formal Methods** (SEFM'2014), Grenoble, France. Topics include: abstraction and refinement; programming languages, program

analysis and type theory; formal methods for real-time, hybrid and embedded systems; formal methods for safety-critical, fault-tolerant and secure systems; software verification and validation; formal aspects of software evolution and maintenance; light-weight and scalable formal methods; tool integration; applications of formal methods, industrial case studies and technology transfer; education and formal methods; etc.

☺ Sep 09-12 43rd **Annual International Conference on Parallel Processing** (ICPP'2014), Minneapolis, MN, USA. Topics include: all aspects of parallel and distributed computing, such as applications, architectures, compilers, programming models, etc.

September 14-15 7th **International Conference on Software Language Engineering** (SLE'2014), Vasteras, Sweden. Topics include: techniques for software language reuse, evolution and managing variation (syntactic/semantic) within language families; engineering domain-specific languages (for modeling, simulating, generation, description, checking); novel applications and/or empirical studies on any aspect of SLE (development, use, deployment, and maintenance of software languages); etc. Deadline for submissions: May 16, 2014 (abstracts), May 23, 2014 (full papers).

September 15-19 8th ACM/IEEE **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2014), Turin, Italy. Topics include: qualitative methods, replication of empirical studies, empirical studies of software processes and products, industrial experience and case studies, evaluation and comparison of techniques and models, reports on the benefits / costs associated with using certain technologies, empirically-based decision making, quality measurement and assurance, software project experience and knowledge management, etc. Deadline for submissions: May 19, 2014 (short papers, posters).

September 22-25 14th **International Conference on Runtime Verification** (RV'2014), Toronto, Canada. Topics include: monitoring and analysis of software and hardware system executions. Application areas include: safety/mission-critical systems, enterprise and systems software, autonomous and reactive control systems, health management and diagnosis systems, and system security and privacy. Deadline for submissions: April 8, 2014 (abstracts), April 15, 2014 (full papers).

September 24-26 14th **Workshop on Automated Verification of Critical Systems** (AVoCS'2014), Twente, the Netherlands. Topics include: model checking, specification and refinement, verification of software and hardware, specification and verification of fault tolerance and resilience, real-time systems, dependable systems, verified system development, industrial applications, etc. Deadline for submissions: June 16, 2014 (abstract), June 23, 2014 (full papers), August 7, 2014 (research ideas).

October 12-16 9th **International Conference on Software Engineering Advances** (ICSEA'2014), Nice, France. Topics include: advances in fundamentals for software development; advanced mechanisms for software development; advanced design tools for developing software; software security, privacy, safeness; specialized software advanced applications; open source software; agile software techniques; software deployment and maintenance; software engineering techniques, metrics, and formalisms; software economics, adoption, and education; improving productivity in research on software engineering; etc. Deadline for submissions: May 16, 2014.

♦ Oct 18-21 ACM SIGAda **Annual International Conference on High Integrity Language Technology** (HILT'2014), Portland, Oregon, USA. Sponsored by ACM SIGAda, in cooperation with Ada-Europe and the Ada Resource Association (approvals pending). Deadline for submissions: June 7, 2014 (technical articles, extended abstracts, experience reports, panel sessions, workshops, tutorials), July 3, 2014 (industrial presentations).

☺ October 20-24 ACM **Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2014), Portland, Oregon, USA. Deadline for submissions: June 8, 2014 (Dynamic Languages Symposium). Deadline for early registration: September 19, 2014.

November 03-07 16th **International Conference on Formal Engineering Methods** (ICFEM'2014), Luxembourg, Luxembourg. Topics include: abstraction and refinement; program analysis; software verification; formal methods for software safety, security, reliability and dependability; tool development, integration and experiments involving verified systems; formal methods used in certifying products under international standards; formal model-based development and code generation; etc. Deadline for submissions: April 11, 2014 (abstracts), April 18, 2014 (papers).

November 04-06 14th **International Conference on Software Process Improvement and Capability dEtermination** (SPICE'2014), Vilnius, Lithuania. Topics include: process assessment, improvement and risk determination in areas of application such as automotive systems and software, aerospace systems and software, medical device systems and software, safety-related systems and software, financial institutions and banks, small and very small enterprises, etc. Deadline for submissions: June 13, 2014 (tutorials), June 20, 2014 (full papers, extended abstracts). Deadline for early registration: September 1, 2014.

November 16-22 22nd ACM SIGSOFT **International Symposium on the Foundations of Software Engineering** (FSE'2014), Hong Kong, China. Topics include: architecture and design; components, services, and middleware; distributed, parallel, and concurrent software; embedded and real-time software; formal methods; model-driven software engineering; program analysis; reverse engineering; safety-critical systems; scientific computing; software engineering education; software evolution and maintenance; software reliability and quality; specification and verification; tools and development environments; etc.

December 08-12 15th ACM/IFIP/USENIX **International Middleware Conference** (Middleware'2014), Bordeaux, France. Topics include: design, implementation, deployment, and evaluation of distributed system platforms and architectures for computing, storage, and communication environments, including reliability and fault-tolerance; scalability and performance; programming frameworks, parallel programming, and design methodologies for middleware; methodologies and tools for middleware design, implementation, verification, and evaluation; etc. Deadline for submissions: May 9, 2014 (abstracts), May 16, 2014 (papers).

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

December 10-12 15th **International Conference on Product Focused Software Development and Process Improvement** (PROFES'2014), Helsinki, Finland. Topics include: software engineering techniques, methods, and technologies for product-focused software development and process improvement as well as their practical application in an industrial setting. Deadline for submissions: June 18, 2014 (full papers), June 25, 2014 (short papers), November 3, 2014 (posters).

December 17-20 21st IEEE **International Conference on High Performance Computing** (HiPC'2014), Goa, India. Topics include: parallel and distributed algorithms/systems, parallel languages and programming environments, hybrid parallel programming with GPUs and accelerators, scheduling, resilient/fault-tolerant algorithms and systems, scientific/engineering/commercial applications, compiler technologies for high-performance computing, software support, etc. Deadline for submissions: May 16, 2014 (papers), September 16, 2014 (student symposium submissions). Deadline for early registration: November 14, 2014.

2015

April 11-19 18th **European Joint Conferences on Theory and Practice of Software** (ETAPS'2015), London, UK. Events include: CC (International Conference on Compiler Construction), ESOP (European Symposium on Programming), FASE (Fundamental Approaches to Software Engineering), FOSSACS (Foundations of Software Science and Computation Structures), POST (Principles of Security and Trust), TACAS (Tools and Algorithms for the Construction and Analysis of Systems).

December 10 200th birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

The 19th International Conference on Reliable Software Technologies - Ada-Europe 2014 is an exciting event with an outstanding technical program, keynote talks, an exhibition from Tuesday to Thursday, and a rich program of workshops and tutorials on Monday and Friday.

The conference is hosted by **ECE**, a French engineering school located near the Tour Eiffel, right in the heart of Paris, with convenient connections to all places of interest, and lots of facilities around. An event not to be missed!

For full details and up-to-date information, see the conference website:

http://www.ada-europe.org/conference2014

Conference program at a glance

Monday	Tuesday	Wednesday		Thursday	Friday
	Opening/Welcome				
3 tutorial tracks + CPS workshop	Keynote talk	Keynote talk	Ada-France workshop	Keynote talk	3 tutorial tracks + MCS workshop
... continued ...	Technical papers	Technical papers	... continued ...	Technical papers	... continued ...
3 tutorial tracks + CPS workshop	Vendor session	Industrial track *Ada in Aerospace*		Industrial track *Ada in Railway*	3 tutorial tracks + MCS workshop
... continued ...	Presentations	GNAT retrospective		Technical papers	... continued ...
	Ada-Europe General Assembly	Cruise and conference banquet Best paper award		Best presentation award Closing session	

Keynote talks

On the three central days of the conference week, a keynote will be delivered as the opening event to address hot topics of relevance in the conference scope. The keynote speakers include: **Robert Lainé**, to talk about *Lessons learned and easily forgotten*, drawing from his many years of experience in space projects leadership at the European Space Agency and EADS Astrium; **Mohamed Shawky**, from Université de Technologie Compiègne (France), to present his futuristic work on *Intelligent Transportation Systems*; and **Alun Foster**, Acting Executive Director and Programme Manager of the ARTEMIS JU, to expose *From ARTEMIS to ECSEL: growing a large eco-system for high-dependability systems*, about the results achieved in ARTEMIS and the objectives of the new ECSEL program.

Exhibition

The exhibition will open on Tuesday morning, and run until the last session on Thursday. It will take place in the conference venue; coffee breaks will be served in the exhibition space. Don't let your

company miss this opportunity of engaging with Ada, reliable software, real-time, and embedded community experts from some of Europe's largest and well-known companies and institutes; contact exhibit@ada-europe2014.org.

Tutorials

Improve the benefits of coming to the conference further by attending our tutorials, all given by famous experts.

Monday				Friday		
T1	**T3**	**T5**	AM	**T7**	**T9**	**T10**
T. Taft	**I. Broster and A. Coombes**	**B. Brosgol**		**W. Bail**	**L. Asplund**	**R. Chapman and Y. Moy**
Proving Safety of Parallel/Multi-Threaded Programs	*Debugging Real-time Systems*	*High-Integrity Object-Oriented Programming with Ada 2012*		*Technical Basis of Model Driven Engineering*	*Robotics Programming*	*Introduction to Verification with SPARK 2014*
T2	**T4**	**T6**	PM	**T8**		
T. Taft	**A. Alonso, A. Crespo and J. Martin**	**J. Sparre-Andersen**		**W. Bail**		
Multicore Programming using Divide-and-Conquer and Work Stealing	*Developing Mixed-Criticality Systems with GNAT/ORK and Xtratum*	*Ada 2012 (Sub)type and Subprogram Contracts in Practice*		*An Overview of Software Testing with an Emphasis on Statistical Testing*		

Social program

The conference banquet will be held aboard a ship, cruising along the Seine! Enjoy excellent food while passing by the world's famous Tour Eiffel, Louvre, Musée d'Orsay, Notre-Dame, and more in the night's light!

Workshops

In addition to various co-located events, three major workshops are taking place in connection with the conference:

- Monday June 23rd: Workshop on *Challenges and new Approaches for Dependable and Cyber-Physical Systems Engineering*, organized by **CEA** and **Thales**.
- Wednesday June 25th: **Ada-France** day: *Ada 2012: le point sur le langage* (Ada 2012: Assessing the Language), a special session in French for software managers who want to learn about the current state of Ada.
- Friday June 27th: Workshop on *Mixed Criticality Systems: Challenges of Mixed Criticality Approaches and Benefits for the Industry*, organized by **ECE**.

GNAT retrospective

The 2014 Ada-Europe conference marks the 20th anniversary of GNAT as a supported open-source Ada compiler. This started a new era for the distribution and the promotion of the Ada language. A retrospective will look back at these important 20 years.

ACM SIGAda Annual International Conference
High Integrity Language Technology HILT 2014
Call for Technical Contributions

Developing and Certifying Critical Software

P Portland Marriott Downtown Waterfront Hotel
Portland, Oregon, USA
October 18-21, 2014

Sponsored by ACM SIGAda in cooperation with
Ada-Europe and the Ada Resource Association
Contact: SIGAda.HILT2014@acm.org www.sigada.org/conf/hilt2014

NOTE

HILT 2014 will take place on the four days immediately preceding — and in the same hotel as — the 2014 ACM SIGPLAN conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH). This "co-location" will make it possible for registrants to attend both conferences.

SUMMARY

High integrity software must not only meet correctness and performance criteria but also satisfy stringent safety and/or security demands, typically entailing certification against a relevant standard. A significant factor affecting whether and how such requirements are met is the chosen language technology and its supporting tools: not just the programming language(s) but also languages for expressing specifications, program properties, domain models, and other attributes of the software or overall system. HILT 2014 will provide a forum for experts from academia/research, industry, and government to present the latest findings in designing, implementing, and using language technology for high integrity software. **We are soliciting technical papers, experience reports, and tutorial proposals on a broad range of relevant topics.**

POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

- New developments in formal methods
- Multicore and high integrity systems
- Object-Oriented Programming in high integrity systems
- High-integrity languages (e.g., SPARK)
- Use of high reliability profiles such as Ravenscar
- Use of language subsets (e.g., MISRA C, MISRA C++)
- Software safety standards (e.g., DO-178B and DO-178C)
- Typed/Proof-Carrying Intermediate Languages
- Contract-based programming (e.g., Ada 2012)
- Model-based development for critical systems
- Specification languages (e.g., Z)
- Annotation languages (e.g., JML)
- Teaching high integrity development
- Case studies of high integrity systems
- Real-time networking/quality of service guarantees
- Analysis, testing, and validation
- Static and dynamic analysis of code
- System Architecture and Design including Service-Oriented Architecture and Agile Development
- Information Assurance
- Security and the Common Criteria / Common Evaluation Methodology
- Architecture design languages (e.g., AADL)
- Fault tolerance and recovery

KINDS OF TECHNICAL CONTRIBUTIONS

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM *Ada Letters*. The Proceedings will be entered into the widely consulted ACM Digital Library accessible online to university campuses, ACM's more than 100,000 members, and the wider software community.

EXTENDED ABSTRACTS discuss current work for which early submission of a full paper may be premature. If your abstract is accepted, a full paper is required and will appear in the proceedings. Extended abstracts will be double-blind refereed. In 5 pages or less, clearly state the work's contribution, its relationship with previous work (with bibliographic references), results to date, and future directions.

EXPERIENCE REPORTS present timely results and "lessons learned". Submit a 1-2 page description of the project and the key points of interest. Descriptions will be published in the final program or proceedings, but a paper will not be required.

PANEL SESSIONS gather groups of experts on particular topics. Panelists present their views and then exchange views with each other and the audience. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation and, if selected, a subsequent abstract for a 30-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for ACM *Ada Letters*.

WORKSHOPS are focused sessions that allow knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. Workshop proposals, up to 5 pages in length, will be selected based on their applicability to the conference and potential for attracting participants.

TUTORIALS can address a broad spectrum of topics relevant to the conference theme. Submissions will be evaluated based on applicability, suitability for presentation in tutorial format, and presenter's expertise. Tutorial proposals should include the expected level of experience of participants, an abstract or outline, the qualifications of the instructor(s), and the length of the tutorial (half day or full day).

***HOW TO SUBMIT*: Except for Tutorial proposals use www.easychair.org/conferences/?conf=hilt2014**

Submission	*Deadline*	*Use Easy Chair Link Above*
Technical articles, extended abstracts, experience reports, panel session proposals, or workshop proposals	**June 7, 2014**	For more info contact: **Tucker Taft**, Program Chair `taft@adacore.com`
Industrial presentation proposals	**July 3, 2014** (overview)	
Send **Tutorial** proposals to	**June 7, 2014**	**John McCormick**, Tutorials Chair `mccormick@cs.uni.edu`

At least one author is required to register and make a presentation at the conference.

FURTHER INFORMATION

CONFERENCE GRANTS FOR EDUCATORS: The ACM SIGAda Conference Grants program is designed to help educators introduce, strengthen, and expand the use of Ada and related technologies in school, college, and university curricula. The Conference welcomes a grant application from anyone whose goals meet this description. The benefits include full conference registration with proceedings and registration costs for conference tutorials/workshops. Partial travel funding is also available from AdaCore to faculty and students from GNAT Academic Program member institutions, which can be combined with conference grants. For more details visit the conference web site or contact **Prof. Michael B. Feldman** (`MFeldman@gwu.edu`)

OUTSTANDING STUDENT PAPER AWARD: An award will be given to the student author(s) of the paper selected by the program committee as the outstanding student contribution to the conference.

SPONSORS AND EXHIBITORS: Please contact **Greg Gicca** (`gicca@verocel.com`) to learn the benefits of becoming a sponsor and/or exhibitor at HILT 2014.

IMPORTANT INFORMATION FOR NON-US SUBMITTERS: International registrants should be particularly aware and careful about visa requirements, and should plan travel well in advance. Visit the conference website for detailed information pertaining to visas.

ANY QUESTIONS?

Please send email to SIGAda.HILT2014@acm.org or Conference Chair (**Michael Feldman**, mfeldman@gwu.edu), or Program Chair (**Tucker Taft**, taft@adacore.com).

Reliable Software in Bioinformatics: Sequence Alignment with Coq, Ada and SPARK

Karen Sargsyan

Institute of Biomedical Sciences, Academia Sinica, 128 Sec. Academia Rd., Taipei, Taiwan; Tel:+886-2-27899043; email: karsar@ibms.sinica.edu.tw

Abstract

Bioinformatics is becoming an indispensable tool for personalized medicine. Software is the most important factor to enable bioinformatics in practical applications and the reliability of such software has not been previously discussed. In this paper we share our strategy to ensure the reliability of the software for biological sequence alignment, which is the most widely used application in bioinformatics. We present the first findings of our project, which incorporates Coq, Ada and SPARK tools, and illustrate the reliability issues specific to bioinformatics.

Keywords: Bioinformatics, Coq, Ada, SPARK

1 Introduction

Bioinformatics is a computer science applied to biological problems, as a result, it is a subject to high expectations in the current era of computational technologies and advances in biology. Research in bioinformatics results in software, which means to provide solutions for biomedical problems. It is therefore of interest to discuss the reliability requirements which must apply to such software and to our knowledge, this is not currently taking place within the bioinformatics community. However, we do need highly reliable software in this field and we need to assess the requirements of its reliability.

The rapid development of bioinformatics during the last few decades brings huge advancements for the biomedical field, as well as new technical challenges for researchers. In the near future, doctors will have access to genetic data on which to base and tailor their medical treatment. To outline the advances in this field, we refer to the announcement of the full genome sequencing which costs less than $1000 [8]. In addition, a complete clinical assessment of a patient incorporating a personal genome [4] and the 1000 Genomes Project to sequence 1000 individuals [1] serve as further examples. Although these achievements merely appear to be of scientific value, recent developments benefit medicine directly. Thousands of DNA variants are associated with diseases and traits [14]. Chemotherapy medications (imatinib and trastuzumab) to treat specific cancers [11, 15] and a targeted pharmacogenetic dosing algorithm for warfarin [16, 17] demonstrate the potential of personalized medicine. Checking for susceptible genotypes for abacavir, carbamazepine and clozapine [13, 9, 7] reduces adverse effects. All of these examples demonstrate a rapid development in the field of personalized medicine. Reliability is important if life or health is dependent on the software in use. We expect the latter will become the case for bioinformatics software in near future. Here, we pay attention to reliability issues, which specifically apply, to bioinformatics, whilst avoiding a discussion of measures that any reliable software development has to apply.

2 Typical sources of errors in Bioinformatics

One of the biggest challenges in bioinformatics is the amount of data. It is not practical to store all results and make similar runs on the same datasets. These problems are still awaiting solutions and an increased speed of algorithms is desired. As a consequence, safety of the software is not a high priority for scientists in the short-term. However, we raise the importance of applying visible integration of bioinformatics with medicine and the time required to develop reliable software.

Errors come from different sources within bioinformatics. The error rate of available sequencing technologies results in significant challenges for applications. We still have to verify a novel DNA variant identification by placing it into its genomic context due to the high false positive rate. Verification itself is dependent on genome size and is time consuming. Therefore, it is not always a simple task to distinguish software errors from those which arise from data during experimental verification.

A programming language of choice can contribute to error accumulation in the code, if the language is not equipped with safety assurance tools or their usage is neglected. As illustrated in [10], popular choices of programming languages in bioinformatics, are C, C++, C#, Java, Perl and Python. Unfortunately, the only subjects of [10] are speed of execution and memory usage for popular algorithms in bioinformatics, with no analysis of software reliability provided. Moreover, R language for statistical software is omitted, although much is done in R to make statistical analysis reliable. In our survey we did not find the application of verification tools (provers, statistical analysis of the code, etc.), except for unit tests (which do not always exist) in conjunction with those languages for the case of bioinformatics software. The survey was conducted on published research articles and open source software, which are representative for the current stage of bioinformatics development. Furthermore, there are implementations of the new algorithms and verified/tested algorithms on a

small number of datasets due to the absence of additional data. In severe cases, software is provided for the sole demonstration of the algorithm and verification of the results is provided in the paper, but with no further warranty provided for its accuracy. Despite these facts, several software tools have been developed over recent decades and the results of these applications prove their safety in scientific settings.

Application of the existing non-bioinformatics software for the analysis of biological data where no special software exists, may still lead to erroneous results. For example: Excel (Microsoft Corp., Redmond, WA) altered irreversibly gene names, which looked like dates [18].

3 Sequence Alignment

The general term of bioinformatics includes sequence analysis, genome annotation, gene expression analysis and other subfields. The wide variety of approaches also includes sequential, structural (structures of the molecules are analyzed), network analysis, multi-agent modeling and others. Sequence alignment, which is singled out as a mature and applied part of bioinformatics, will be the main focus for the rest of this paper. Also, problems of software reliability mentioned in this specific context are general enough to be discovered in other parts of bioinformatics. Hence, instructions to solve issues, which are illustrated in this specific context are also valuable in other subfields. However, scenarios existing in other approaches, different from the sequence analysis, will require additional attention in the future.

The sequence (of protein, RNA or DNA) provides information about genes encoding proteins, RNA genes, regulatory sequences, structural motifs, and repetitive sequences. Sequence alignment tries to match up biological sequences to others and evolutionary arguments justify sequence alignment by stating that the similar sequences have a comparable function and/or molecular structure. Therefore using sequence alignment we may answer such questions as:

- Which species have a protein that is evolutionarily related to a certain known protein?
- Which other genes encode proteins that exhibit given structures?

Programs for sequence alignment, such as Basic Local Alignment Search Tool (BLAST), search sequences from more than 260,000 organisms, containing over 190 billion nucleotides daily [5]. The sequencing process, which is the power of contemporary genetics, incorporates a variant of sequence alignment itself. Some algorithms contain sequence alignment as subroutines. Therefore, the reliability of sequence alignment software has an impact on bioinformatics.

4 A formal model of Sequence Alignment with Coq

Here we concentrate mostly on the Smith-Waterman (SW) local alignment algorithm. This algorithm may be briefly described as:

$$H(i,0) = 0, 0 \le i \le m$$
$$H(0,j) = 0, 0 \le j \le n$$

If $a_i = b_j \Rightarrow w(a_i, b_j) = w(match)$ or if

$a_i \neq b_j \Rightarrow w(a_i, b_j) = w(mismatch)$

$$H(i,j) = \max \begin{cases} 0 \\ H(i-1, j-1) + w(a_i, b_j) \\ H(i-1, j) + w(a_i, -) \\ H(i, j-1) + w(-, b_j) \end{cases}$$

Where *a,b* are the strings over alphabet corresponding to nucleotides in case of DNA/RNA or amino-acids for proteins, *m,n* are the lengths of the sequences, '-' denotes gap, *H(i,j)* - is the maximum similarity score and *w(x,y)* is the scoring scheme. After obtaining *H(i,j)*, one starts with the highest value in it and moves backwards to one of positions (i − 1,j), (i, j − 1), and (i − 1, j − 1), which has the highest value, until (0, 0) is reached. So (i − 1,j), (i, j − 1) moves correspond to adding gaps to the second and the first sequence accordingly.

The motivation for using local alignments is the existence of a reliable statistical model. Another incentive is difficulty in obtaining correct alignments in regions of low similarity for distantly related biological sequences. Whether local alignment as described above is a useful tool depends on how it agrees with biological experiment. Therefore, one may consider it as a formal model, which is the subject of experimental verification. In the context of software reliability, we need to note that the direct implementation of the algorithm will have long execution time, which is not fast enough for the majority of real-time applications. To solve this problem, more advanced implementations of SW are suggested [12, 2]. Their aim is to provide results similar to SW, with a faster performance in order of times. Because of the importance of SW, implementations using FPGA, CUDA and SIMD architectures exist. The next optimization of the algorithm is more complicated as it becomes harder to ensure the implementation is equivalent to the outcome of SW. This is where the proof assistants, such as Coq (http://coq.inria.fr), may be helpful. Availability of libraries, documentation and an organized community for Coq make it a natural choice for our project. Although, other alternatives to Coq may also work. Our approach is to implement a Coq module corresponding to SW and use it to prove equivalence to SW for each new optimized implementation. Coq also allows extraction of the proof in the form of a certified functional program.

These are arguments we would like to bring in justification of this strategy. It is always better to ensure the implementation used for medical purposes adheres to a well-known formal specification or a model. In scientific settings, it is clear which formal model is used and whether it sufficiently models reality, without guessing if the errors of implementation impact on observed disagreement. In the case of such errors one either ignores the correct model or the errors mask the disagreement.

The complex biological problems have different models to achieve the same goals, with application under different conditions. For sequence alignment, the most widely used substitution of SW is BLAST [3]. BLAST is regarded as less precise than SW, producing inferior alignment under some settings, but having a practical execution speed. As its description is lengthy, we do not discuss it here. However, it is of interest to present a module specifying BLAST in Coq. This is valuable, since many optimizations of BLAST exist (CS-BLAST, CUDA-BLASTP, Tera-BLAST, etc.), similar to SW. Moreover, in such a setup, judgments about SW and BLAST are subject of exact proofs. To illustrate our ideas the author provides Coq codes for SW specification and an example of simple certified SW alignment program under the GPLv3 license (forbars.github.io). The listing of the code is not given here, as it is subject to further refinement.

It is possible to describe SW alignment in a language of paths on the graphs. Alternatively, one might choose another general mathematical framework in which SW alignment is a particular case. In our approach we do not assume any knowledge on the future form of formalized bioinformatics. Our first specifications are not of an abstract nature and our work follows an exploratory approach. In the author's view, it is risky to assume what the mathematics of bioinformatics will look like beforehand.

5 Reliable Sequence Alignment with Ada and SPARK

Although model specifications implemented in Coq are valuable in general, it is more interesting to apply specifications in a more practical realm. Our choice of language for a sequence alignment package implementation is Ada 2012 and SPARK 2014. Decades of development with attention on safety of code, compilation on a wide variety of platforms (including embedded applications) and acceptable speed of execution (in comparison to C) make these a natural option. It is not an easy task to modify a functional program extracted from Coq to the requirements of a particular (embedded and real-time) hardware. Therefore, our specifications in Coq are intended to formalize bioinformatics and do not deal with specific requirements, such as integer number representation in a provided system. SPARK, being a subset of Ada, allows proof of the correctness of subroutines before compilation. In addition, its subset is actively formalized in Coq [6], which makes it possible in the future to compare programs written in SPARK against specifications in Coq. Proving SPARK code with even fewer strict specifications is important, as testing of the bioinformatics tool involves large datasets and this makes it harder to prepare valid unit tests and find errors.

Ada contracts serve in a similar way, however, it is not always an option, as they check pre- and post- conditions during runtime. It is less desirable for medical software if a precompiled verification is available. Here, we avoid discussion of the improvements that are possible due to the access of suitable tools and methods in writing safe applications for general software case. It is worth noting, the practices which exist for safe code have to be transferred to the software in bioinformatics. Rather, we present a specific issue in sequence alignment with demonstration of natural capabilities of Ada to provide a solution.

Different alignment algorithms often yield different results. This fact is widely ignored in practice. As a result of increased sizes of sequence datasets, faster alignment tools are needed. The trend is to make assumptions about the nature of frequent sequences and apply simplifications, which result in faster alignment, but this leads to a loss of quality. Methods incorporating sequence alignment as a subroutine are usually verified in experiments with a specific alignment in mind. Therefore, it is not immediately known how they behave if an alignment procedure is replaced with an alternative one. Moreover, those algorithms are dependent on parameters, such as weight matrix (BLOSSOM65 in BLAST). Weight matrices contribute to the difference between outputs, as their customized versions may be in use. Furthermore, tools for the sequence alignment are not always designed to provide output in the same format, as a consequence, several packages implement unification of interfaces for sequence alignment methods (an experiment in BioPython as an example). This makes it easy to interchange alignment tools in the code. However, as far as alignment algorithms are not equivalent, it is necessary to choose a suitable output and reject the production from the untested alignment with application in mind. This is where strong typing of Ada becomes important. Ada types are different in case their names differ, without accounting for the fact that the implementations are the same. As an example:

```
type Aligned is … ;
type SW is new Aligned;
type BLAST is new Aligned;
```

Here, we see two different versions of Aligned, which differ only in their name. However, functions and procedures that require the result of SW alignment will not accept the output of other forms of alignment. Thus, we associate separate types with outputs of different alignments. Strict distinction of the outputs, as proposed here, is not implemented or suggested in other packages/tools to our knowledge. It forces us to pay more attention to proven facts and methods in bioinformatics during software implementation and to avoid ignorance of the differences between sequence alignment algorithms.

6 Conclusion and Further Research

It has become even more apparent that bioinformatics software will play an important role in personalized medicine. However, current trends in software development for bioinformatics tend to give priority to the speed of the algorithms, together with complicated optimizations of existing implementations and accessibility, such as a service via cloud computing. Without denying the importance of all those topics, we raise the question of the implementation reliability of the core algorithms in bioinformatics and consider its importance for the future. Our plans include providing Coq modules as formal specifications for the important algorithms in bioinformatics and the corresponding tools for developing bioinformatics software in Ada and SPARK. All specifications in Coq and some examples of SPARK and Ada code are published or are subject of publication under the GPL license (forbars.github.io).

References

[1] 1000 Genomes Project Consortium (2010), *A map of human genome variation from population-scale sequencing. Nature*, 467:1061-1073.

[2] S. F. Altschul, B. W. Erickson (1986), *Optimal sequence alignment using affine gap costs*, Bulletin of Mathematical Biology 48: 603–616.

[3] S. F. Altschul et al. (1990), *Basic local alignment search tool*, Journal of Molecular Biology 215 (3): 403–410.

[4] E. A. Ashley, et al. (2010), *Clinical assessment incorporating a personal genome*, Lancet, 375:1525-1535.

[5] D. A. Benson, et al. (2008), *GenBank*, Nucleic Acids Res. 36(Database issue): D25–D30.

[6] P. Courtieu et al. (2013), *Towards the formalization of SPARK 2014 semantics with explicit run-time checks using coq*, HILT '13 Proceedings of the 2013 ACM SIGAda annual conference on High integrity language technology, pp: 21-22.

[7] M. Dettling, et al. (2007), *Clozapine-induced agranulocytosis in schizophrenic Caucasians: confirming clues for associations with human leukocyte class I and II antigens*, Pharmacogenomics J. 7:325-332.

[8] R. Drmanac, et al. (2010), *Human genome sequencing using unchained base reads on self-assembling DNA nanoarrays*, Science, 327:78-81.

[9] P. B. Ferrell, H. L. McLeod (2008), *Carbamazepine, HLA-B*1502 and risk of Stevens-Johnson syndrome and toxic epidermal necrolysis: US FDA recommendations*, Pharmacogenomics, 9:1543-1546.

[10] M. Fourment, M. R. Gillings (2008), *A comparison of common programming languages used in bioinformatics*, BMC Bioinformatics. 9: 82 – 91.

[11] C. Gambacorti-Passerini (2008), *Part I: Milestones in personalised medicine–imatinib*, Lancet Oncol., 9: 600.

[12] O. Gotoh (1982), *An improved algorithm for matching biological sequences*, Journal of molecular biology 62(3):705–708.

[13] S. Hetherington, et al. (2002), *Genetic variations in HLA-B region and hypersensitivity reactions to abacavir*, Lancet, 359:1121-1122.

[14] L. A. Hindorff, et al. (2009), *Potential etiologic and functional implications of genome-wide association loci for human diseases and traits*, Proc. Natl Acad. Sci. USA, 106: 9362-9367.

[15] C. A. Hudis (2007), *Trastuzumab–mechanism of action and use in clinical practice*, N. Engl. J. Med. 357:39-51.

[16] International Warfarin Pharmacogenetics Consortium et al (2009), *Estimation of the warfarin dose with clinical and pharmacogenetic data*, N. Engl. J. Med. 360:753-764.

[17] H. Sagreiya, et al. (2010), *Extending and evaluating a warfarin dosing algorithm that includes CYP4F2 and pooled rare variants of CYP2C9*, Pharmacogenet. Genomics. 20:407-413.

[18] B. Z. Zeeberg, et al. (2004), *Mistaken Identifiers: Gene name errors can be introduced inadvertently when using Excel in bioinformatics*, BMC Bioinformatics, 5:80-86.

Physical Units with GNAT

Christoph K W Grein
email: christ-usch.grein@t-online.de

Abstract

There has often been a demand to be able to compute with physical items where dimensional correctness is checked. However, methods working at compile-time suffered from the combinatorial explosion of the number of operations required for mixing units and thus could be used with a set of only very few units like e.g. distance, time, and speed. The full SI system with seven base dimensions evaded all such attempts. On the other hand, methods working at run-time were not really applicable because of the memory and calculation overhead.

Ada with its newest generation of 2012 has introduced so-called aspect clauses to allow, among others, specifying additional type properties like type invariants. AdaCore's GNAT uses these aspects in an implementation-specific way to handle physical units at compile-time. This paper presents an overview of the achievements and shortcomings of this method.

With some modification, AdaCore's invention, in the author's opinion, might be apt to standardization in a future Ada generation.

Keywords: physical units, aspect clause, Ada enhancement.

1 Introduction

In the Ada Europe conference in Toulouse 2003, the present author, with co-authors Dmitry Kazakov and Fraser Wilson, gave an overview of methods used to handle physical units in Ada [3, 4]. Unfortunately, the Ada 95 issue 324 [5] dealing with a proposal by the *Ada Rapporteur Group* had not been able to be included in the proceedings because the final submission date for papers had just expired when the ARG proposal was published, so it was mentioned only orally. The conclusion to be drawn from all those attempts was that neither compile-time nor run-time methods were satisfactory for general use.

With the Ada 2012 aspects, things might turn out different. AdaCore's [1] GNAT compiler handles physical dimensions with implementation defined aspects at compile-time in such an ingenious way that it might be apt to be standardized with the next Ada generation (whenever this might be). However, for this to occur, proper demand from the Ada community must be shown to the ARG and the method must prove itself free from pitfalls. Otherwise, ARG would view any such request with utmost reluctance.

This paper presents the author's personal view on the achievements and shortcomings of the GNAT method (as of GNAT GPL 2013 [2]). As you will see, the notation is very natural; any combination of units is possible without the dreaded combinatorial explosion. Some problems have already been solved since the method's first release a few years ago due to user input. It is the author's hope that this paper will induce further discussions among physicists and help to optimize the method so that its chances of standardization will be increased.

2 Shortcomings of hitherto used methods

Compile-time methods using separate types for each dimension and overloading for operators mixing types are well-known to suffer from the combinatorial explosion of the number of operators needed. Thus also the ARG proposal [5] was doomed to fail, which was heavily based on a very clever use of generics. Hence those methods are only applied for a small set of dimensions.

Run-time methods, on the other hand, store dimension information for each item in additional components and thus suffer from the vast additional time and space demands for storing and calculating them. It is unknown to the present author whether these methods have found any application at all.

3 GNAT's use of aspects

GNAT uses an implementation-specific language extension of the new Ada 2012 aspects to define a type and appropriate subtypes for any physical dimension in such a way that dimensional correctness can be checked at compile-time.

The type to be used for physical items is defined with the GNAT-specific aspect Dimension_System, a kind of record aggregate, specifying the seven base dimensions together with the base unit names and symbols. The symbols may be either characters or strings. (The dimension symbols are used for error messages in case of dimensional errors only.) This is done in a package called System.Dim.MKS (see next page). Conceptually, the Dimension_System aggregate declares a record with components

```
record is
   Meter, Kilogram, Second, Ampere,
   Kelvin, Mole, Candela: Fraction;
end record;
```

which the compiler invisibly affixes to any object of the type MKS_Type during compile-time, where the type of each record component is a fraction, i.e. either an integer or a rational number; the Unit_Symbol might be seen as a shortcut for an aggregate like e.g.

```
'm' := (Meter => 1, others =>0):
```

```
package System.Dim.MKS is
type MKS_Type is new Long_Long_Float with
    Dimension_System =>
    ((Unit_Name => Meter   , Unit_Symbol => 'm'  ,
                             Dim_Symbol => 'L'),
     (Unit_Name => Kilogram, Unit_Symbol => "kg" ,
                             Dim_Symbol => 'M'),
     (Unit_Name => Second  , Unit_Symbol => 's'  ,
                             Dim_Symbol => 'T'),
     (Unit_Name => Ampere  , Unit_Symbol => 'A'  ,
                             Dim_Symbol => 'I'),
     (Unit_Name => Kelvin  , Unit_Symbol => 'K'  ,
                             Dim_Symbol => "Θ"),
     (Unit_Name => Mole    , Unit_Symbol => "mol",
                             Dim_Symbol => 'N'),
     (Unit_Name => Candela , Unit_Symbol => "cd" ,
                             Dim_Symbol => 'J'));
```

Other such types may be defined, using at most seven base dimensions, but less are tolerated like for instance the outdated Gaussian CGS with only three dimensions (centimeter, gram, second; the electric and magnetic items use combinations of fractional powers thereof):

```
type CGS_Gauss is new Long_Long_Float with
    Dimension_System =>
   ((Unit_Name =>Centimeter, Unit_Symbol => "cm",
                             Dim_Symbol => 'L'),
    (Unit_Name => Gram     , Unit_Symbol => 'g' ,
                             Dim_Symbol => 'M'),
    (Unit_Name => Second   , Unit_Symbol => 's' ,
                             Dim_Symbol => 'T'));
```

From this type, GNAT creates subtypes via another aspect Dimension, again in the form of a kind of record aggregate:

```
subtype Length is MKS_Type with
  Dimension => (Symbol => 'm',
                Meter =>1,
                others =>0);
```

Here, no connection is present to any of the dimensions defined before, although the aggregate component's name Meter seems to indicate so, because no check is made that the symbol does not conflict with the unit symbol defined above. Any nonsense is possible like Symbol => 's' or even Symbol => "XYZ".

```
subtype Speed is MKS_Type with
   Dimension => (Symbol => "m/s",
                 Meter  => 1,
                 Second =>-1,
                 others => 0);
```

Again no check is performed that the symbol is compatible with the exponents as long as it is composed from basic symbols. Silly lapses like Symbol => "m/s**2" will remain undetected.

Of course, new names may be defined for further subtypes' dimension symbols. We even can use fractional powers:

```
subtype Charge is CGS_Gauss with
   Dimension => (Symbol     => "esu",
                 Centimeter =>3/2,
                 Gram       =>1/2,
                 Second     =>-1,
                 others     =>0);
```

For sure, no check can be performed here that this is correct, as the symbol "esu" has no connection to the unit symbols. This in fact is an implicit declaration of the unit g**(1/2)*cm**(3/2)/s.

In this way, GNAT defines all other SI units with names and symbols by further subtypes:

```
subtype Pressure is MKS_Type with
  Dimension => (Symbol   => "Pa",
                Meter    => -1,
                Kilogram =>  1,
                Second   => -2,
                others   =>  0);

subtype Thermodynamic_Temperature is
   MKS_Type with  Dimension =>
                 (Symbol   => 'K',
                  Kelvin   =>1,
                  others   => 0);

subtype Celsius_Temperature is
   MKS_Type with   Dimension =>
                  (Symbol   => "°C",
                   Kelvin   =>1,
                   others   =>0);
```

The declaration of Celsius_Temperature in the author's opinion is a bad mistake, since temperatures in Kelvin are not simply compatible with those in Celsius.

In effect, the GNAT method is very similar to one of those described in the Toulouse Ada Europe conference 2003 [3], except that the type is not private and the dimension record is present only during compile-time.

4 Notation

The package System.Dim.MKS goes on to declare constants for all named SI units with names reflecting the symbols in order to be able to write values with units:

```
m : constant Length := 1.0;
s : constant Time   := 1.0;
g : constant Mass   := 1.0e-3;
A : constant Electric_Current := 1.0;
Si: constant Electric_Conductance :=  1.0;
dC: constant Celsius_Temperature :=  273.15;

Dist := 5.0 * m;
Dist := 5.0 * M;
```

Whether this is a good idea is questionable, because symbols (and prefixes, see below) are case sensitive whereas Ada is not.

So please note here that the correct symbol S(upper case) for *Siemens* is impossible (and therefore replaced by the invention Si) because of the s (lower case) for *Second* defined before. Also note that in the last line, M for *Meter* in upper case is legal, but actually in the wrong casing,

misleading the reader to think of some other item like a mass.

And GNAT's declaration of dC looks like a severe error in the author's opinion since 5°C is anything but 5.0 * dC! Most probably this constant is meant as the conversion factor between Kelvin and Centigrade, but the latter should not have been declared as a subtype in the first place.

On the other hand, short names should be avoided in any case in software. In physics literature, items are written in italics, units in straight face, so that 5g is 5 grams, but 5*g* could be 5 times the earth acceleration. With short names, mistakes are probable if not inevitable.

A better idea could be to do without these declarations and use the original symbol names instead, since this would easily solve the casing problem, e.g.:

```
Dist := 5.0 * 'm';
Dist := 5.0 * 'M';  -- illegal
Pres := 2.1 * "Pa";
```

Here, the symbol M would be wrong and illegal since there is no such symbol.

An even better idea could be to define a *dimensioned literal*, i.e. a new kind of numeric literal with unit suffixes (in a similar way as C defines literals with suffixes describing the length like 1L for a long integer):

```
Time_of_Travel := 5.0's';
Conductance := 4.2'S';
```

Prefixes

Prefixes pose the same case sensitivity problem – mS is Milli-Siemens, Ms is Mega-Second. You often find such wrong casings in software, and the author, being a physicist, finds this abhorrent.

GNAT defines prefixes only for a few of the base units (meter, kilogram, second, ampere), and only for some powers (milli to mega) of all those defined for the SI system (from 10^{-24} to 10^{+24}). Case sensitivity hits back here – names different from the SI ones have to be used for some prefixes like Meg instead of Mg.

```
mg : constant Mass := 1.0E-06;  -- milli
Meg: constant Mass := 1.0E+03;  -- mega
```

To define prefixes in this way for all named units and all powers is of course feasible, but introduces names that will never be used because of inappropriate size (like GF, gigafarad, whereas capacities generally lie in the pico- respectively nanofarad range; or kT, kilotesla, a magnetic field strength which would tear apart any matter).

A better proposal could be: Again use dimensioned literals like 5.0"ms", and the casing problem is solved. The author has no proposal how these prefixed units could be defined. Ideas are welcome.

As a preliminary conclusion, we see that, apart from some problematic cases, the notation is very natural.

```
Grav: constant Accelaration := 9.81*m/s**2;
  -- 9.81"m/s**2";  -- author's proposal
T: Time;
D: Length;

D := 0.5*Grav*T**2;
```

As was mentioned above, even fractional powers are provided, so that the author's pet equation, the Schottky-Langmuir equation, may be solved for any item, the current density *j*, the voltage *U*, the distance *d*.

$$j = \frac{4}{9}\varepsilon_0\sqrt{\frac{2e_0}{m_0}}\frac{U^{\frac{3}{2}}}{d^2}$$

When you declare constants of unknown dimension like intermediate values, the dimension is taken from the initial value as has always been the case with indefinite subtypes:

```
Material_Const: constant MKS_Type :=
      4.0/9.0 * Eps0 * (2.0 * E0/M0)**(1/2);
```

Unfortunately, GNAT does not allow this for variables.

Let us deal with some more fractional exponents.

```
Dist: constant Length := (8.0*cm)**(1/3+2/3);
```

This fails with dimension mismatch because of preference of integer division in the exponent (1/3+2/3=0+0). However, this works:

```
Eight_cm: constant Length := (8.0*cm)**((1+2)/(5-2));
```

Admittedly, who would write such nonsense, but the rational arithmetics package (there is no documentation) seems inconsistent.

You have to be very careful with fractional powers because of the preference of integer division. The reason for this behaviour lies in the very base of Ada and is partly unavoidable:

```
 8.0**(1/3) = 1.0  -- (a)
 8.0**(1/3)*cm  = 2.0*cm  -- (b)
(8.0*cm)**(1/3) = 2.0*cm**(1/3)  -- (c)
```

Case (a) is "classical" Ada: 1/3=0; (b) and (c) use fractional arithmetic with the GNAT invention because the item is dimensioned. You also need a dimensioned value to give the expected result for expression (a) (by the way: what *is* expected here?):

```
One: constant MKS_Type := 1.0;
8.0**(1/3)*One = 2.0  -- GNAT invention
8.0**(1/3) = 1.0  -- classical Ada
```

Here, the constant One also has *dimension* 1, i.e. in normal parlance it is "dimensionless". This behaviour might lead to very difficult to find problems, to say the least.

Exponents must be known at compile time, so X**N may be written as long as N is a static integer constant, but the following is illegal when X is not dimensionless:

```
for N in A_Range loop
   … X**N …  -- illegal
end loop;
```

This is not really a limitation since variable exponents turn up normally only in power series, and these can be reformulated so that the dimension is extracted to a common factor.

Fractional constants cannot be written at all since GNAT does not disclose their type:

```
One_Third: constant ?:= 1/3;
```

5 Mathematical functions

We have seen that fractional powers are possible. This means that the exponentiation operator is triply overloaded:

```
function "**"  -- Standard
  (Left: MKS_Type'Base; Right: Integer)
  return MKS_Type'Base;
function "**"  -- GNAT invention
   (Left: MKS_Type'Base;
    Right: Rational)
   return MKS_Type'Base;
function "**"  -- gen.elem.functions
  (Left: MKS_Type'Base;
   Right: MKS_Type'Base)
  return MKS_Type'Base;
```

where the first two have dimensioned arguments and return another dimensioned value, whereas the last requests all parameters dimensionless and also returns a pure number.

(The second declaration is in fact a lie, there is no type named *Rational*, but a function with such a profile must exist somewhere, albeit hidden. And it is absolutely not clear how GNAT manages to resolve the overloading of an expression like a**(1/3), since 1/3=0 if the literals in the fraction are of type Integer; see the cases (a) and (b) above in the previous section. So of which type are the literals if 1/3 is a *Rational* and not an Integer.)

This leads us to the question which dimensions are allowed for arguments of mathematical functions. From physics, we know the answer: They must all be dimensionless except for the square root (i.e. the rational exponent 1/2) and some of the trigonometric functions, e.g.:

```
function Sin (X, Cycle: MKS_Type'Base)
      return MKS_Type'Base;
```

Both arguments here must have the same dimension, the result is dimensionless, a pure number. The corresponding rule holds for the inverse function:

```
function Arcsin (X, Cycle: MKS_Type'Base)
      return MKS_Type'Base;
```

must request X dimensionless and return a value that is dimensioned like Cycle.

Similar rules apply to the arctangent as the reverse of the tangent:

```
function Arctan (Y: MKS_Type'Base;
                 X: MKS_Type'Base := 1.0;
                 Cycle: MKS_Type'Base])
      return MKS_Type'Base;
```

must request X and Y to have the same dimension (the quotient Y/X must be dimensionless) and the return value must be dimensioned like Cycle.

On the other hand, an expression like Exp(5.0*m) or Sin(42.0*kg) is complete nonsense.

GNAT requires all mathematical functions except SQRT of an instantiation of the package Ada.Numerics.Generic_Elementary_Functions for a dimensioned type to have dimensionless parameters.

6 Vectors and records

An instantiation of package Ada.Numerics.Generic_Real_Arrays provides arrays and matrices, which may serve as vectors and tensors. Providing a dimension is possible, but is partly ignored:

```
subtype Axis is Integer range 1 .. 3;
subtype Vector is Real_Vector (Axis);
  A: Vector := (1=> 1.0, 2 => 0.0, 3 => -9.8) * cm/s**2;
  D: Vector := (Axis => 0.0) * m**2;
  T: Mass   := 10.0*kg;
  D := A * T**2 / 2.0;
```

The result of this equation with nonsense units is computed numerically correct, i.e. the factor 10^{-2} (because of the unit cm) is taken into account in the acceleration vector A. The result, when output (see IO below), is without unit indication. It seems that GNAT takes the units into account when computing the values, but then ignores dimensions.

While a matrix and a vector may have a dimension as a whole, individual components with different dimensions are not allowed. This is in best order, since using the method for linear algebra (see below) is more than can be expected.

On the other hand, records may serve for instance as a collection of particle properties, so each component may indeed have a different dimension like mass, charge, location, speed, etc.

GNAT allows those multidimensional components:

```
type Particle is record
   M: Mass;
   Q: Electric_Charge;
   R: Vector := (Axis => 0.0) * m;
  -- Darn, this conflicts with M!
   V: Vector := (Axis => 0.0) * m/s;
   -- Same conflict.
end record;
```

We see here another reason why short names especially for units are evil.

7 Input and output

There is a generic package System.Dim.Float_IO, which however is a plain lie – only output exists, however with unit indication; input of dimensioned items is still an open issue. Also the output facility leaves a lot of wishes open.

```
Q: Electric_Charge := 40.0 * C;
R: Length := 10.0 * cm;
Put (Q**2/R**2, Aft => 2, Exp => 0);
```

This results in

```
160000.00 m**(-2).s**2.A**2
```

The opinions about the dot as a unit separator may vary. However, the fact that there is a blank character between the number and the unit makes it difficult to read the value back in. How can a potential Get operation discriminate between a pure value (with dimension 1) and a dimensioned value when there is no indication how far to read? Also the dot separator makes the integer number 2 in a sequence like s**2.A look like a floating point number 2.0 (remember that upon input, the decimal digits after the dot may be omitted).

The specification of Put is

```
procedure Put
  (Item  : Num_Dim_Float;
   Fore  : Field  := Default_Fore;
   Aft   : Field  := Default_Aft;
   Exp   : Field  := Default_Exp;
   Symbol: String := "");
```

There is some description given in the package specification how the Symbol could be used, but when used the compiler complains (as of GNAT GPL 2013):

```
Symbol parameter should not be provided
reserved for compiler use only
```

In former compiler versions, the symbol could be any string, which, when given, replaced the unit output. There was not any check that the string was appropriate, so any nonsense could be supplied.

What is expected when the symbol string is given, is at least a check that the symbol be appropriate or else an exception be raised. Far better would be an adaptation to the magnitude requested. So for instance the expected output for

```
Put (12.0*m, Symbol => "km");
```

must be something like 0.012 km.

The compiler developers are well aware of the missing input facility. On a personal note to the author they said they were waiting for user requirements.

Compare [3] for a better solution of IO.

8 Type conversions

Since GNAT's dimensional types are numeric types, the Ada type conversion is available, e.g. with the two types shown in this paper, we could write:

```
H: MKS_Type := CGS_Gauss (1.0 * Oe);
```

This is utter nonsense! The H-field is measured in Oersted in the Gaussian system, in A/m in SI. The correct conversion is

$$1\ \mathrm{Oe} = \frac{1000}{4\pi}\ \mathrm{A/m}$$

A type conversion like this cannot handle the unit conversion, so the best would be that this be illegal.

GNAT allows such conversions!

Of course, for being able to provide correctly dimensioned conversions, some form of the Ada type conversion must be available. Ideas are welcome again. One possible way would be to allow type conversions only for dimensionless values, so that the original dimension would first have to be stripped, the value type-converted, last the new dimension added together with the necessary conversion factors.

9 Linear algebra

There is one further application which in the author's view need not be handled: linear algebra. This means that physical dimensions need not be included when linear equations are solved like e.g. for linear partial differential equations. Thus, vectors (like velocity or force) (represented as arrays with three components) and tensors (3 by 3 matrices) just have one physical dimension, whereas in linear algebra, arrays and matrices may have any number of components and each component may have a different dimension. This is, in the author's opinion, way beyond what this method can (and should be able to) handle.

10 Conclusion

GNAT's use of Ada's new aspects, despite its present shortcomings, for physical dimensions is indeed ingenious and deserves attention and thoughtfulness by the Ada community. It has been much improved over the years, and most of the problems mentioned in this paper can easily be solved. Also a few improvement proposals have been presented.

Thus the author again wants to express his hope that widespread use of this method will persuade Ada programmers to further improve the method by communicating their findings to AdaCore and eventually ask the Ada Rapporteur Group to consider incorporation into the next Ada standard.

References

[1] AdaCore, http://www.adacore.com/

[2] GNAT GPL 2013, http://libre.adacore.com/tools/gnat-gpl-edition/

[3] C. Grein, D. A. Kazakov and F. Wilson, *A Survey of Physical Unit Handling Techniques in Ada,* In Jean-Pierre Rosen, Alfred Strohmeier (Eds), Reliable Software Technologies - Ada-Europe 2003, Lecture Notes in Computer Science, Vol. 2655, Springer-Verlag.

[4] C. Grein, *Handling Physical Dimensions in Ada*, http://www.christ-usch-grein.homepage.t-online.de/Ada/Dimension.html

[5] AI95-00324-01 *Physical Units Checking* http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ais/ai-00324.txt?rev=1.3.

Tool Qualification for Safety Related Systems

Mathias Ekman

Bombardier Transportation Sweden AB, Västerås, Sweden; email: mathias.ekman@se.transport.bombardier.com

Henrik Thane

Safety Integrity AB, Västerås, Sweden; email: henrik.thane@safetyintegrity.se

Daniel Sundmark

Mälardalen University, Västerås, Sweden; email: daniel.sundmark@mdh.se

Stig Larsson

Effective Change AB, Västerås, Sweden; email: stig.larsson@effectivechange.se

Abstract

Tools used in the development of safety related software applications need to be qualified as safe. That is, the tools cannot be allowed to introduce hazardous faults into the application, e.g., a compiler shall not generate dangerous code due to failure of the compiler. In many cases laws and regulations require the product development of safety related applications to comply with industry sector specific safety standards. Examples of such standards include EN50129/50128 for railway applications, ISO/EN13849 for machines with moving parts, DO-178B/C for avionics, or ISO26262 for cars. These standards require the use of a rigorous development and maintenance process. The standards are also mainly intended to be used when developing systems from scratch. However, most development and test tools are not developed from scratch according to the rigorous processes of these standards. In order to address this issue, some of the standards provide means for qualifying existing tools as a more lightweight and pragmatic alternative to a regular certification process. In this paper we analyze the concept of these qualification approaches. The result of the analysis in our contribution includes a set of approaches that can be applied individually or as a combination in order to reduce the effort needed for qualifying tools. As a running example we use one of the most flexible but at the same time dangerous, even prohibited, maintenance techniques available: dynamic instrumentation of executing code. With this example, we describe how exceptions in these standards can be utilized in order to qualify a dynamic instrumentation tool with a minimal effort, without following the process of tool certification as defined by the standards.

Keywords: tool qualification; certification; functional safety; software instrumentation; dynamic instrumentation.

1 Introduction

Many of the products we use every day have safety related software in them, controlling vital and potentially dangerous functions. The most obvious examples are functionality in modern cars like airbags, ABS systems, stability control systems, radar controlled cruise controls, and automatic braking systems. Similarly, for railway, like trains and subways, computer software controls the doors, propulsion, brakes, traffic signaling, etc. For these systems to be deemed safe, the functionality has to be deemed *free from unreasonable risk, i.e., free from dangerous failures* as defined in the best-practice functional safety standard IEC61508:2010 [1]. The IEC61508 is a generic standard, covering the entire safety life cycle of a safety related system. This standard, even though you can use it on its own, is a template for the implementation of industry-specific functional safety standards, which may also be harmonized with relevant legislation. Example implementations include EN50129/50128 [2] for railway applications, and ISO26262 [3] for cars. Another significant standard, albeit not descendant from IEC61508, is the avionics standard DO-178B/C [4]. All these standards make use of their own, in some cases - the same, classification into how dangerous systems are and what kind of integrity is required of the safety protection mechanisms in order to reduce the identified risks down to a tolerable, safe, level. For example, IEC61508 defines SIL – Safety Integrity Levels 1 to 4[1], while ISO26262 defines ASIL – Automotive Safety Integrity Levels A to D. In the remainder of this paper we will use SIL as synonymous to any type of safety integrity level allocation to hazards or mitigations as they may be defined in any of the standards.

The functional safety standards do not only define requirements on the development and maintenance of safety related systems, but also on tools used in the development and maintenance of such systems. The standards may, for example, require the usage of a certified

[1] In IEC61508, SIL 4 denotes the highest risk, and the highest requirement on the mitigation. ASIL D is correspondingly the highest risk classification in ISO26262.

compiler or a certified test tool with a SIL that is on par with safety related system's required SIL.

In this paper, we review alternative approaches of qualify tools to be used for safety related systems, and apply as a running example a complex and potentially dangerous dynamic software instrumentation tool for exemplifying and evaluating some of these approaches. We investigate these approaches by posing the following research questions:

- **RQ1:** What are the alternatives for tool qualification in different standards?
- **RQ2:** What are the differences in terms of qualification effort?
- **RQ3:** What are the risks associated with different approaches to tool qualification?

We will begin by briefly describing the tool that we are using as a running example.

2 Dynamic instrumentation

As a running example in this paper we use one of the most flexible but at the same time dangerous, even prohibited, maintenance techniques available: dynamic instrumentation of executing code. Here is a brief introduction to the basic concepts and techniques.

When testing, debugging or running diagnostics on a system it is essential to be able to observe the system behaviour: typically, inputs, outputs and internal execution activities. A common means to increase the observability is to make use of some type of online-monitoring mechanism, like software instrumentation [8]. Static software instrumentation requires code to be prepared prior execution and is limited to only allow for activation of prepared instrumentation code, making it difficult to add or modify code after deployment. A more flexible approach is dynamic software instrumentation, which is a technique that allows for information extraction software to be downloaded, and patched into a running system, so that run-time information can be collected from the real-world execution. This approach has several advantages over static instrumentation, due to its flexibility to modify instrumentation code and alter the instrumentation points during runtime without having to restart the actual execution of target. Since there is no need to restart the system to extract execution information or data for debugging or verification purposes, rare fault conditions that are difficult to reproduce off-target can be analyzed with the actual real-time environmental data available.

2.1 The running example

We have previously published two methods for dynamic instrumentation based on binary modification [8,10]. Implemented in our tool, these methods automate the entire process of inserting and activating code for monitoring purposes. Our tool makes it possible to instrument a running system without preparing the original source code, and relying on dynamic linking of object files. Experiments have shown a low probe effect since only few instructions are needed for invoking the actual instrumentation code. Also, when the instrumentation code is disabled, there is no dormant code that needs to be executed, resulting in an even lower probe effect. Instrumentation points do not need to be prepared, allowing for a high flexibility of where to insert the instrumentation code. See author's previous publications [8,9] for details about these methods.

An overview description of the process of using our tool is provided in Figure 1. Basically, the user adds instrumentation code (2.a) to the original source code (1.a) of the running application. The user then compiles the new code and run our tool (3.a). The tool identifies what have changed between the old executable code (1.c - running on the target) and the instrumented code (2.b). The tool then allows for downloading a patch to the running target (3.b-c) and activates it. The tool allows for extracting logs from the running target (3.b), and then allows the user to disable the patch in order to restore the system to its original state.

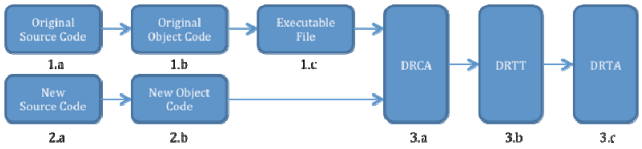


Figure 1 - Dynamic Instrumentation Process[2].

When using dynamic software instrumentation tools in safety-critical systems, there is a risk that bugs in the code may lead to hazardous failures. Examples of such bugs include invalid code or data generated as output from the tool, which may lead to memory corruptions on the target. Consequently, if a dynamic software instrumentation tool is used, then the tool must be deemed to be safe according to an applicable standard. In some of the standards, e.g., EN50129/50128 [2,17], and IEC61508 [1], dynamic reconfiguration, like in-run-time instrumentation, is not recommended for use, or in other words it is prohibited for higher SILs, unless it can be proven to be safe. In the following sections, we will investigate the research questions as stated in 1, in order to understand the possibilities to qualify our tool.

3 Qualification of tools

There are basically three approaches to qualify tools according to standards like EN50128 and ISO26262:

1. Develop the safety case from scratch according to the standard with the rigor needed for achieving the target SIL.
2. Follow procedures defined in the standard for qualification of tools not developed according to the standard; "Tool Qualification".
3. Design a protection harness that shelters the safety related system from dangerous tool outputs.

[2] DRCA=Dynamic Relink Code Analyzer, DRTT=Dynamic Relink Transfer Tool, DRTA=Dynamic Relink Target Agent.

Of the above approaches (1) is most expensive in terms of effort since it implies the full use of a standard and the retroactive usage of the standard to an already existing tool, while (3) is only limited to prove SIL integrity of the tool's protection system that should have less complexity than the tool itself, thus requiring significantly less effort. The alternative of qualifying existing tools (2) can be seen as a back door, were the full weight of the standards can be avoided while maintaining the safety of the system. This approach includes proving that previously used tools are legible for the "proven in use" concept as described in the standards requiring proofs to be supplied that for example the hardware and runtime-environment is unmodified, in conjunction with providing a history of successful execution records. In the following sections, we elaborate on these three approaches, to reduce the effort needed for qualifying tools, with a focus on dynamic instrumentation as a running example.

4 Approach I: Develop the safety case from scratch

This approach requires either tools to be developed from scratch, or that the entire safety case is constructed after the product has been developed and released. This is typically the approach that requires most effort in terms of qualification effort. But, even for minor projects, like in our case with the dynamic instrumentation example, a relatively large amount of planning, specification, reviewing, testing and, documentation is needed to argue and substantiate that the target SIL has been achieved, which is in our case according to SIL4/ASIL-D. The number of requirements that must be fulfilled for this approach according to the EN50128 standard is about 390, and for the ISO26262 standard it is about 370. Even though this approach implies an extensive qualification effort, it also allows for the most stringent and safest approach in terms of risks to expose safety hazards.

5 Approach II: Tool qualification

In this chapter, we review tool qualification according to the railway standard EN50128 and the automotive standard ISO26262, with the aim to qualify our dynamic instrumentation tool according to these standards. The sections for each standard are divided into *tool classification, and tool qualification process*.

Essentially, the tool qualification approach addresses two basic questions:

- *What harm can the tool-set do to the system?*
- *Is there a way to detect or even prevent that a fault in the tool-set leads to a hazard?*

5.1 Tool classification

EN50128 classifies tools into *on-line* and *off-line* tools, with different requirements for use cases and hazard analysis for each category. There are three tool categories defined for off-line support tools in EN50128 (3.1):

- T1: "generates no outputs which can directly or indirectly contribute to the executable code (including data) of the software"
- T2: "supports the test or verification of the design or executable code, where errors in the tool can fail to reveal defects but cannot directly create errors in the executable software"
- T3: "generates outputs which can directly or indirectly contribute to the executable code (including data) of the safety-related system"

Examples of tools belonging to class T1 are editor and configuration management tools, as these do no generate output that can contribute to the software execution. A verification tool that fails to detect an error introduces a risk to the system, is classified as class T2. Examples of tools belonging to class T2 are test coverage tools, and static analysis tools. Class T3 include source code compilers and compilers that incorporate executable run-time packaging into executable code, like in dynamic instrumentation tools, but also tools used for configuration of parameters or variables.

EN50128 allows tools to be qualified as proven in use according to EN50128: 6.7.4.4.a; "a suitable combination of history of successful use in similar environments and for similar applications." This means that arguments must be provided that the runtime environment is identical, but also that the actual application is similar. This is only valid within an organization where it has been previously used.

ISO26262 has a slightly different approach, since it classifies software tools into different tool confidence levels, namely TCL1, TCL2 and TCL3. A software tool is classified in ISO26262 based on its possible use cases, but classification is also based on an analysis if erroneous tool output leads to violation of safety requirements, or if the tool fails to detect or prevent these kinds of errors.

The tool classification level (TCL) combined with the Automotive Safety Integrity Level (ASIL) of the safety-related software developed using the software tool, determines the selection of the appropriate tool qualification methods. There is also a possibility to qualify a tool that has been proven in use according to ISO26262-8, clause 14. The validity of the proven in use argument is evaluated based on factors including; number of hours previously in use, field data analysis relevant to safety-related events and runtime environment conformance – including hardware. For a proven in use status to be obtained, reports that indicates the frequency of incidents during a specific period of usage must be provided. An incident in this case is defined as a failure that is potential to lead to the violation of a safety goal, see table 1. For example, for ASIL-D status to be obtained there must be provided usage reports that indicate a maximum of 10^{-9} safety violating failures per hour.

ASIL	Observable incident rate
D	$< 10^{-9}$/ h
C	$< 10^{-8}$/ h
B	$< 10^{-8}$/ h
A	$< 10^{-7}$/ h

Table 1 – Observable incident rate.

ISO26262 defines a tool classification approach where the TCL can be determined based on its tool impact level (TI) and tool error detection (TD) levels (see table 2). The TI level is defined as the possibility that a malfunction in the tool can introduce or fail to detect errors in a safety-related item, or element being developed. The possible impact is analyzed, and if the tool can satisfy all safety requirements, impact level TI 1 shall be selected, for all other cases TI 2 shall be selected. After selecting TI, the TD level shall be selected. This level determines the degree of confidence that the tool prevents or detects erroneous output data. TD levels range between TD1 and TD3, where TD3 is the lowest confidence in error prevention or detection, and TD1 is the highest. In case when several use cases results in different TCL levels, the highest level that implies highest requirements shall be selected.

		Tool Error Detection		
		TD 1	TD 2	TD 3
Tool Impact	TI 1	TCL 1	TCL 1	TCL 1
	TI 2	TCL 1	TCL 2	TCL 3

Table 2 – Determination of TCL levels.

Running example – dynamic instrumentation

In our case, we apply the *off-line* parts of the dynamic instrumentation toolset, excluding the *on-line* parts of the toolset running on the target, since it requires a different approach for qualification and will be considered as future work. The EN50128 standard classifies offline tools into three categories, where our dynamic instrumentation tool is to be considered as a class T3 tool (highest requirements), since it generates outputs similar to a source code compiler, which can directly or indirectly contribute to the executable code, including data.

ISO26262 has a different approach where no differentiation is made between offline and online tools. The standard classifies tools according to the reliability of the behaviour, tool confidence level combined with the expected ASIL. The input for this judgment is an evaluation of the tool impact level (TI) in conjunction with the tool error detection level (TD). Since the tool should fulfill all safety requirements according to ASIL-D in our case, any malfunction shall be avoided or detected, as required by TI1. The tool is developed according to ASIL-D, the level with the highest degree of confidence, which allows the selection of TD1. Based on the selections of TI1 and TD1, TCL1 should be selected according to table 2.

5.2 Tool qualification process

EN50128 states that any potential failure of the toolset must be detected by technical or organizational measures outside the tool. Evidences must be provided that tool failures do not affect the toolset output in a safety related manner (EN50128: 6.7.1). Also, for a tool in class T3 it is required that the output from the tool conforms to the specification of the output or that failures in the output are detected. Tools in categories T2 and T3 shall include identification of potential failures that can affect the tool output, and actions should be specified to avoid such failures. Evidences must be provided that the tool output conforms to the specification, or that failures in the output are detected. Evidence can also be based on (EN50128: 6.7.4.4):

- a suitable combination of successful history use in similar environments and applications
- tool validation (as specified in EN50128: 6.7.4.5)
- diverse redundant code (allows detection and control of failures)
- compliance with safety integrity levels derived from risk analysis
- other appropriate methods for avoiding or handling tool failures

In case conformance evidence according to EN50128: 6.7.4.4 is not available, effective methods should be applied to control failures of the software that the tool imposes. For example a non-trusted compiler can be justified if a combination of tests, checks and analysis, which are capable of ensuring the correctness of the code, and that, it is consistent to the target safety integrity level.

ISO26262 states that confidence is needed that the software tool effectively achieves the following goals;

- (1) Minimizing the risk of systematic faults due to software tool malfunctioning
- (2) Adequate development process used, if activities or tasks required by the standard rely on the correct functioning of the software tool used

Software tools needs to be certified for each project. When a software tool is to be used, the environmental and functional constraints and its operating conditions must be determined. The qualification process requires a planning of the software usage according to the following:

- Identification and version number of the software tool
- Configuration of the tool (for example compiler switches)
- Use cases of the tool (user interaction)
- Software environment
- Maximum ASIL from all the safety requirements
- Based on confidence level: Qualification Methods

In addition, the tool qualification process requires identification of possible use cases. When this is performed, goals 1 and 2 above are considered, i.e. the possibility that a malfunction of the tool (behavior or output) leads to a condition not fulfilling the safety-requirements. The probability to detect or prevent such failures shall be evaluated. When these steps are performed, the required TCL is considered. As stated above, TCL 3 is the most stringent level, and TCL 1 is the lowest.

Tools with TCL 1 are not required to apply for tool qualification. Tools with TCL 2-3 require tool qualification. The TCL and the ASIL[3] level of the safety-related software being developed is the base for the selection of allowed tool qualification methods, as listed in ISO26262-8, see table 3 and 4.

	Qualification Methods for tools classified as TCL3	A	B	C	D
1a	Increased confidence from use in accordance with 11.4.7	+ +	+ +	+	+
1b	Evaluation of the tool development process in accordance with 11.4.8	+ +	+ +	+	+
1c	Validation of the software tool in accordance with 11.4.9	+	+	+ +	+ +
1d	Development in accordance with a safety standard	+	+	+ +	+ +

Table 3 – Qualification methods TCL3 in ISO26262-8.

	Qualification Methods for tools classified as TCL2	A	B	C	D
1a	Increased confidence from use in accordance with 11.4.7	+ +	+ +	+ +	+
1b	Evaluation of the tool development process in accordance with 11.4.8	+ +	+ +	+ +	+
1c	Validation of the software tool in accordance with 11.4.9	+	+	+	+ +
1d	Development in accordance with a safety standard	+	+	+	+ +

Table 4 – Qualification methods TCL2 in ISO26262-8.

Running example – dynamic instrumentation

The approach of applying tool qualification of dynamic instrumentation tools begins with follow procedures defined in the standard for qualification of tools not developed according to the standard. This approach is applicable for off-line parts of tools like dynamic instrumentation techniques, since it contains code generator parts that may otherwise be difficult to develop according to a standard. This is a prominent approach also for existing tools that are not developed from scratch.

[3] ++ - Highly recommended. + - Recommended. ASIL A-D.

EN50128: 6.7 describes the process of qualifying tools; a non-trusted compiler, as in our case the code generator part of the tool, can be justified if a combination of tests, checks and analysis are available, which are capable of ensuring the correctness of the code, and that, it is consistent to the target safety integrity level. ISO26262: 11.4.9 describes the process in a similar manner, validation measures, malfunction and erroneous output should be detected, but also reaction of the tool to anomalous operating conditions shall be examined. For EN50128, the tool qualification process emphasizes on that evidences must be provided that potential failures of the tool do not affect the output in a safety related manner, such that failures are not detected. Since dynamic instrumentation tools are classified as T3, evidences must be provided that the tool output conforms to the specification, or that failures in the output are detected. Evidences may be automated; in our case a possible approach is to automate the correctness check of the output from the tool, before the code is activated on target. Another approach would be to formally prove that the output conforms to the specification, for example by invoking a model checking tool analyzer. Other options include proven in use (successful history), but this option is more likely to be suitable for tools like compilers with a relative static output result from a certain input. This option may not be suitable for dynamic instrumentation tools because of its rather limited usage compared to for example a widely used commercial compiler. Validation is another option. Design deviations of the tool are identified, by facilitating coverage tests, static code analyzers etc. but require extensive testing and, depending on the implementation complexity of the dynamic instrumentation tool, may be an overwhelming task in terms of effort. Another option available, which is mentioned in the standard as “other appropriate methods”, is to introduce a protection harness like a safety shell that detects all failures in the output (see section 6).

In ISO26262, qualification methods may include validation methods for detecting and preventing software faults, like the introduction of code coverage tools, static code analyzers and model checkers. One approach for our running example is to reduce the TCL to TCL1, by implementing a tool error detection system. By using this approach, all failures in the output will be detected for the dynamic instrumentation toolset. This means that there is no need to follow the actual qualification process as stated in the standard, since tools classified as TCL1 do not need any qualification methods.

In both ISO2626 and EN50128, tools might be qualified according to ‘increased confidence from use’ versus ‘proven in use’. ISO26262 defines a rigorous separate section for this (ISO26262: 11.4.7), and specifies different levels of recommendations regarding confidence from use, depending on relevant ASIL. EN50128 limits the requirements to ‘a suitable combination of history of successful use in similar environments and applications within the organization’ (EN50128: 6.7.4.4.a), thus leaves a larger freedom to the assessor to interpret the requirements.

This might of course lead to a divergent qualification level among different qualifications.

Consequently, both these standards allows for two options in our running example:

1. Validation – Extensive tests are performed to detect any design deviations of the tool like code coverage tests, static code analyzer, simulators and model checkers.
2. Detecting all failures in the output – design diagnostics. By applying design diagnostics based on Failure mode and Effect Analysis (FMEA) or Fault Tree analysis (FTA), possible failure modes can be identified to analyze the causes and effects, in order to take protective action. This option is possible even without specification of the tool (COTS). For ISO26262, applying design diagnostics implies reduction to TCL1 by implementing a TD1 error detection system.

The effort needed for the tool qualification approach is directly related to the number of requirements in the standards. For EN50128, the requirements for qualifying tools in class T3 (as in our running example), is applicable to EN50128: 6.7.4.1-5. In case conformance evidences are unavailable, EN50128: 6.7.4.6-6.7.4.11 (see table 5) is applicable. This means that there are at most 6 requirements according to EN50128 for the tool qualification approach in our running example.

Tool Class	# Requirements
T1	1
T2	5
T3	5 or 6

Table 5 – Tool Qualification requirements for EN50128.

ISO26262 defines five general requirements for tool qualification. In addition, depending on qualification methods selected according to table 3 and 4, there are at most four requirements (see table 6).

Qualification Methods	# Requirements
Increased confidence from use in accordance with 11.4.7	4
Evaluation of the tool development process in accordance with 11.4.8	3
Validation of the software tool in accordance with 11.4.9	2

Table 6 – Tool Qualification Requirements for ISO26262-8.

6 Approach III: Protection harness

Both standards allow for "other appropriate methods" to be used for avoiding or handling failures introduced by the tool. This includes applying internal measures regarding evaluation of safety related functions, to prevent or detect malfunctions in the software tool. This allows for an approach of using a protection system like a safety shell that monitors and shelters dangerous output from the dynamic instrumentation toolset. One way to achieve this is to introduce a safety shell [18] as a protection system that detects a malfunction of the tool and also takes action to handle the failure so that the tool retains its safety integrity. The safety shell should for every intermediate step in the tool-chain sequence be involved to evaluate the result, before the output is passed into the next tool in the chain. For example, to search for invalid combinations of machine operations (e.g. invalid writes) in the binary code representing the new instrumentation code, which may otherwise harm the safety integrity of the system. The evaluation result for the safety shell in this case would be to halt the tool execution as a protection harness. The safety shell should be developed according to EN50128 (SIL4) or ISO26262 (ASIL-D), and should also apply design diagnostics like FMEA/FTA, which is possible even for tools without specifications (e.g., COTS). The motivation for this approach is that it allows for existing tools to be used as is, without the need for a rigorous work to rebuild the tool or safety case from scratch according to approach 1 (section 3), or to qualify the tool according to approach 2 (section 4). A limiting requirement is that the tool chain has a limited set of possible outputs that can be evaluated by the safety shell, without introducing too much complexity to the safety shell, which otherwise may complicate the argumentation for safety cases and thus increase the risk for propagation to safety hazards. However, if this requirement is fulfilled, the approach of utilizing a safety shell significantly reduces the qualification effort compared to approach 1 and 2.

7 Discussion

In this paper, we elaborated on approaches to qualify a tool in safety standards like EN50128 and ISO26262, within the context of a dynamic instrumentation tool (**RQ1**). In both standards, there are alternative approaches available that can be utilized to minimize the qualification effort regarding number of requirements to the development process, as described in respective standard for tool qualification (**RQ2**). A possible alternative according to ISO26262: 5.2 includes the opportunity for the developer to reduce the tool confidence level to 1 (TCL1) by introducing an error detection system of level 1 (TD1), and thus allowing a minimal number of requirements to be anticipated – a back door. EN50128 allows for a similar back door alternative, by classifying the tool as T3 with a minimal set of general requirements (not T3 specific). Both these alternate qualification approaches, back doors, reduces the effort significantly compared to approach 1 (development from scratch). For example, only four requirements needs to be fulfilled in ISO26262 in such case, compared to about 370 requirements for approach 1. In case the protection harness approach (3) is applied for ISO26262, the user needs basically (as alternative to full standard compliance) to perform an FMEA, and develop an exception handler to detect erroneous output, and the requirements are fulfilled for qualification! A similar approach can be selected for EN50128, where a T3

classification implies a reduction of requirements to a minimum. We consider these back doors as a weakness in the standard (**RQ3**), since the lightweight approach of not following processes and methods as stated in the standard, significantly increases the risk that safety related hazards may occur in the system.

8 Related work

Conrad et al. [14] summarizes experiences from qualifying tools according to the standard for tools like Mathworks Embedded Coder and Polyspace Client/Server for C/C++ [6]. The authors state that there is no established tool qualification best practice available in the standard. There is no straightforward mapping between activities/tools and their corresponding verification activities, which leave the practice open to interpretation. The authors suggests that there should be a definition of suitable verification and validation measures to be used in conjunction with a qualified tool, to be able to provide a necessary guidance to successfully utilize the tool in projects that need to comply with ISO26262. Hillebrand et al. [15] propose a systematic methodology to establish confidence in the usage of software tools for ISO26262. The method is based on multi-layered analysis that systematically identifies the risk of tool-introduced errors and error detection failures, and also allows for derivation of the tool confidence level (TCL). By using this methodology, existing verification measures used in the development process can be identified and reused. Asplund et al. [16] presents nine safety goals based on safety-related characteristic of a tool chain to be qualified, including EN50128 and ISO26262. The authors suggest an approach for qualification by dealing with software tools as reusable entities deployed in the context of different tool chains. By this method, authors claim that the problem with stipulating either to narrow or to wide qualification effort for tool qualification is solved.

9 Conclusions and future work

Domain specific safety standards like EN50128 for railway applications, ISO/EN13849 for machines with moving parts, DO-178B/C for avionics, or ISO26262 for cars typically require the use of a complex development environment and also require an extensive maintenance process. These standards are mainly intended for systems that are built from scratch. However, most development and test tools are not developed from scratch according to the rigorous processes of these standards.

In this paper, we have elaborated on tool qualification approaches available among these standards, which is an approach to avoid the rigorous process of a complete certification. The research contribution in this paper is the identification of alternate approaches for reducing the effort needed for qualifying one of the most flexible but also complex and dangerous techniques available; dynamic instrumentation of safety related systems during run-time. However, we consider these alternative back doors as a weakness in the standards because most requirements for processes and methods as described in the standards are avoided, thus significantly increases the risk that safety related hazards might occur in the system.

In future work, we will consider alternative qualification methods, including formal proofs in terms of reduced effort with retained safety integrity of all the tool chain components.

Acknowledgements. This research is supported by the Knowledge Foundation (KKS) through ITS-EASY, an Industrial Research School in Embedded Software and Systems, affiliated with Mälardalen University, Sweden.

References

[1] IEC61508:2010 (2010), *Functional safety of electrical/ electronic/programmable electronic safety-related systems,* International Electrotechnical Comission.

[2] EN50128:2011 (2011, *Communication, signalling and processing systems,* Software for railway control and protection systems.

[3] ISO26262:2011 (2011), *Road vehicles – Functional safety*.

[4] D. Evans and D. Larochelle (2002), *Improving Security Using Extensible Lightweight Static Analysis*, IEEE Software, 19(1): p. 42.

[5] EEMBC Adopts DoubleCheck (TM) for Its Industry-Standard Processor; Benchmarks; Green Hills Software's Static Analysis Tool Increases Code Quality (2007).

[6] Techsource-asia Ltd. http://techsource-asia.com/ products-a-solutions/products/92-iec-certification-kit-for-iso-26262-and-iec-61508/description/3.html

[7] Mathworks Inc, http://www.mathworks.co.uk/products/iec-61508/

[8] M. Ekman and H. Thane (2007), *Dynamic Patching of Embedded Software*, IEEE Real-Time and Embedded Technology and Applications Conference, Seattle, WA, USA.

[9] M. Ekman and H. Thane (2012), *Software Instrumentation of Safety Critical Embedded Systems – A Problem Statement*, IEEE System, Software, SoC and Silicon Debug Conference, Vienna, Austria.

[10] M. Ekman and H. Thane (2008), *Real-Time Dynamic Relinking*, Proceedings of the 22:th IEEE International Parallel and Distributed Processing Symposium, Miami, USA.

[11] Mathworks Inc, http://www.vectorcast.com/pdf/ VectorCAST_IEC_Certification_Kit.pdf

[12] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa (1987), *Electron spectroscopy studies on magneto-optical media and plastic substrate interface*, IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741[Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[13] M. Young (1989), The Technical Writer's Handbook. Mill Valley, CA: University Science.

[14] M. Conrad, P. Munier, F. Rauch (2010), *Qualifying Software Tools According to ISO26262*, In proc. Dagstuhl-Workshop Modellbasierte eingebetteter Systeme (MBEES10).

[15] J. Hillebrand, P. Reichenpfader, I. Mandic, H. Siegl, C. Peer (2011), *Establishing Confidence in the Usage of Software Tools in the Context of ISO26262*, SAFECOMP 2011: p 257-269.

[16] F. Asplund, J. El-khoury, M. Törngren, *Qualifying Software Tools, a Systems Approach*, KTH Royal Institute of Technology, Stockholm, Sweden.

[17] EN50129:2003 (2003), *Communication, signaling and processing systems*, Safety related electronic systems for signaling.

[18] J. Katwijk, H. Toetenel, A. Sahraoui, E. Anderson, J. Zalewski (2000), *Specification and Verification of a Safety Shell with Statecharts and Extended Timed Graphs*, SAFECOMP, volume 1943, page 37-52. Springer, 2000.

Experience in Spacecraft On-board Software Development

Juan A. de la Puente, Alejandro Alonso, Juan Zamorano, Jorge Garrido, Emilio Salazar, Miguel A. de Miguel

Universidad Politécnica de Madrid, ETSIT, Avda. Complutense 30, E-28040 Madrid, Spain

Abstract

This paper describes some important aspects of high-integrity software development based on the authors' work. Current group research is oriented towards mixed-criticality partitioned systems, development tools, real-time kernels, and language features. The UPMSat-2 satellite software is being used as technology demonstrator and a case study for the assessment of the research results. The flight software that will run on the satellite is based on proven technology, such as GNAT/ORK+ and LEON3. There is an experimental version that is being built using a partitioned approach, aiming at assessing a toolset targeting partitioned multi-core embedded systems. The singularities of both approaches are discussed, as well as some of the tools that are being used for developing the software.

Keywords: Real-time systems, model-driven engineering, Ada.

1 Introduction

The UPM STRAST group has a long time experience in developing high-integrity real-time systems. The group research in this domain is currently oriented towards mixed-criticality partitioned systems, development tools, real-time kernels, and language features. In order to validate technical achievements in this field, the UPMSat-2 satellite software is being used as a case study. In this paper, ongoing work and experiences from this development are described.

UPMSat-2 is a project aimed at building a micro-satellite that can be used as a platform for experimenting with various technologies and acquiring long-term experience in different aspects of space systems. The project is being carried out by a multi-disciplinary team at UPM, with the collaboration of several research groups and industrial companies. The satellite is expected to be launched in the final quarter of 2015. STRAST is responsible for developing all the software required for the mission, including on-board software for platform and payload management. The flight software is built as a monolithic system, running on top of an ORK+ kernel on a LEON3 [1] computer board. The software is being developed according to the provisions in the ECCS-E-ST-40 [2] and ECCS-Q-ST-80 [3] standards, in order to ensure that the final software product can be validated for the mission.

Mixed-criticality systems are raising a growing interest in the area of embedded systems, due to their potential for improving software productivity and quality. In the context of the MultiPARTES and HI-PARTES projects, methods and tools for mixed-criticality partitioned multi-core embedded systems are being developed. One of the responsibilities of the group is the development of a toolset for supporting this approach. In this context, UPMSat-2 is being used as a case study. In particular, a partitioned implementation running on a XtratuM hypervisor [4] is being developed for demonstration and validation of the project outcomes.

The methodological and architectural approaches used in this work is described in the rest of the paper. Section 2 contains an overview of the satellite system and the architecture of the on-board computer. The main software subsystems and the architectural approaches are also discussed in this section. Section 3 describes the development tools used. Some details of the validation facility are presented in section 4. Finally, a summary of the lessons learned so far and plans for the next future is presented in section 5.

2 The UPMSat2 On-Board Software System

2.1 Overview of the satellite system

UPMSat-2 is a micro-satellite with a geometric envelope of $0.5 \times 0.5 \times 0.6\,\mathrm{m}$ and an approximate mass of $50\,\mathrm{kg}$ (figure 1). It will describe a low Earth noon sun-synchronous polar orbit [5] with a period about $97\,\mathrm{min}$. There are two visibility periods from the ground station every 24 hours, with an approximate duration of $10\,\mathrm{min}$ each.

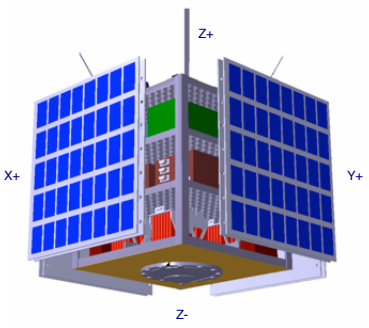


Figure 1: General view of the satellite platform.

Electrical power is provided by solar panels and batteries. Voltage control is analog, keeping voltages for the satellite subsystems within appropriate ranges.

The attitude of the satellite is computer-controlled, using basic sensors and actuators. The attitude is determined by means of magnetometers, which provide a measurement of the Earth magnetic field vector in the satellite reference frame. Deviations are corrected by means of magnetorquers, which create a magnetic field that makes the satellite rotate accordingly.

Communications with the ground station are carried out by means of a dual radio link in the VHF 400 MHz band, with a raw transfer rate of 9600 bit/s. A simplified version of the X.25 data link layer protocol is used for error control and packet transmission.

The payload of the satellite consists of a set of experiments focused on testing different kinds of equipment in a space environment. The experiments have been proposed by industry and some research groups.

There is a single on-board computer (OBC) that executes all the data handling, attitude control, and telecommunications functions. It is based on a LEON3 processor implemented on a radiation-hardened FPGA, with 4 MB RAM, 1 MB EEPROM, and digital and analog interfaces. The on-board software system runs on this hardware platform.

2.2 Software functionality

The main functions of the on-board software can be grouped as follows:

- Platform monitoring and control *(housekeeping)*. Platform data, such as voltages and temperatures at different points, are periodically sampled and checked in order to assess the status of the satellite.
- On-board data handling (OBDH), including decoding and executing telecommands (TC) received from the ground station, and composing and sending telemetry (TM) messages with housekeeping data, event and error logs, or experiment results.
- Attitude determination and control (ADCS). Magnetometer values are read periodically, and used by the control algorithm to compute the intensity output to the magnetorquers in each sampling period.

  Alternative ADCS devices, such as solar sensors or reaction wheel actuators, as well as variations in the control algorithm, will be tested as experiments.
- Experiment management. Most of the experiments require control actions to be executed on them, and sensor data to be collected and sent to ground to be analysed.

Figure 2 shows the software context and the top-level functional blocks.

A key concept in on-board software systems is that of operating modes. The system may be in different modes, and may perform different functions, or execute them in different ways, according to the operating conditions of the system. Figure 3

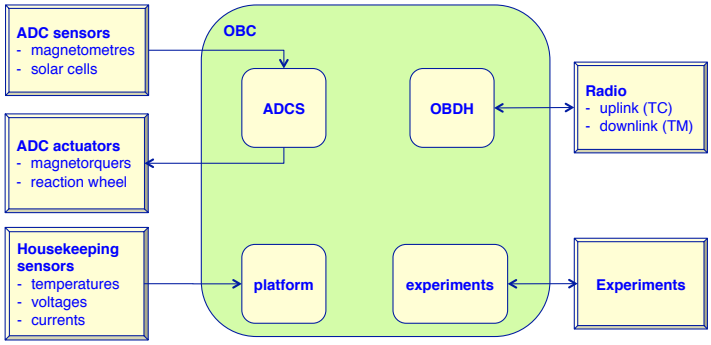


Figure 2: Context and top-level functions.

shows the main operating modes of the on-board software and the events that trigger mode transitions.

The specific functions that are executed in each mode are:

- Initialization mode: load executable code, start execution, and configure I/O devices.
- Nominal mode: housekeeping, OBDH and ADCS as above defined.
- Safe mode: same as nominal mode, with longer periods and reduced functionality in order to save energy power.
- Latency mode: the computer is switched off until batteries are charged (signalled by a hardware timer).
- Experiment mode: one of the experiments is executed, with changes to nominal behaviour if required by the experiment.

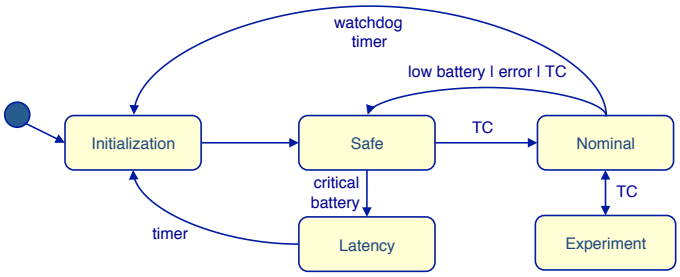


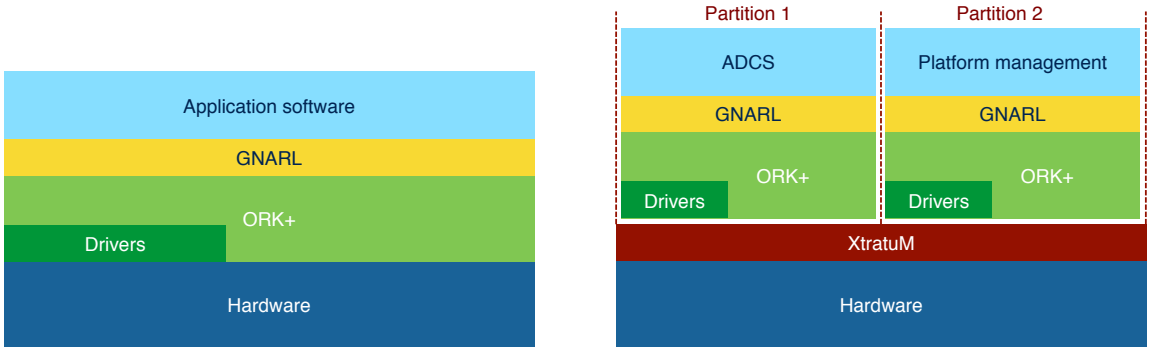
Figure 3: Satellite operating modes.

2.3 Architectural approaches

In order to ensure a timely implementation of the flight software, a monolithic implementation has been designed, using a well known architecture based on GNAT/ORK (figure 4a).

On the other hand, there is growing interest on developing satellite software on partitioned architectures, as exemplified by recent work directed by ESA/ESTEC to develop a partitioned version of the EagleEye reference mission software [6].

The MULTIPARTES project [7] is aimed at developing tools and solutions based on mixed criticality virtualization for multicore platforms. The virtualization kernel is based on XtratuM, a cost-effective open source hypervisor specifically developed for real-time embedded systems [8]. The UPMSat-2 software is being used in MultiPARTES as a case study for



(a) Monolithic architecture. **(b) Partitioned architecture.**

Figure 4: Software architecture.

validating the mixed-criticality technology developed in the project. To this purpose, a partitioned version of the software system is being developed. The partitions run on an adapted version of GNAT/ORK for XtratuM [4]. An example showing the ADCS control subsystem running in one partition and the platform manager providing access to devices in another is shown in figure 4b.

3 Development tools

The increasing complexity of high integrity embedded systems and the need to comply with demanding safety-related standards require suitable toolsets for supporting developers. Model Driven Engineering (MDE) is an appropriate software development approach, that enables the abstraction level of languages and tools used in the development process to be raised. It also helps designers to isolate the information and processing logic from implementation and platform aspects. A basic objective of MDE is to put the model concept on the critical path of software development. This notion changes the previous situation, turning the role of models from contemplative to productive.

The STRAST group has been working with this technology for a long time. The ASSERT project[1] explored the use of MDE technology in space software systems, from which different sets of tools emerged. One of them evolved under the auspices of ESA, resulting in the TASTE toolset [9]. TASTE supports a wide set of modelling languages, such as Simulink [10] and SDL [11], and uses AADL [12] as a glue for architecture modelling. The TASTE tools generate Ravenscar Ada code that can be compiled with GNAT/ORK, and are being used as the primary toolset for the monolithic implementation of the UPMSat-2 software.

Another follow-up of ASSERT was the CHESS project,[2] which was focused on property preservation and composability. In the context of this project, an MDE framework for high-integrity embedded systems was originally developed [13]. In this framework, the functional part of the system is modelled using UML [14]. Models can be enriched with non-functional annotations, in order to integrate different aspects of the software in a single model. This approach has a number of advantages, as it makes models maintenance easier, enables efficient communication within the development team, and supports the validation and analysis of models.

The framework relies on the UML profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE) [15] to describe real-time requirements, properties, resource usage information, and other non-functional properties. A response-time analysis model is automatically generated, which can be used to validate real-time requirements. Finally, source code skeletons in Ravenscar Ada are generated for the main system components.

Mixed-criticality systems are emerging as a suitable approach for dealing with system complexity and reducing development costs, by integrating applications with different criticality levels on the same hardware platform. A separation kernel provides isolation mechanisms in order to guarantee that applications do not interfere with each other. In this way, it is possible to certificate or qualify applications with different criticality levels in an independent way.

Partitioned systems are a remarkable way of providing isolation to applications. In these systems, the separation kernel can be built as a hypervisor that implements a number of partitions as virtual machines isolated from each other in the time and space domains. In this way, applications with different criticality levels can run in different partitions without experimenting any interference from other applications. This approach makes the system development more difficult, as its time behaviour may get much more complex, and requires the hypervisor to be carefully configured.

In the context of the MultiPARTES project, the original CHESS framework is being improved in order to deal with mixed-criticality systems [16]. The first activity accomplished is the identification of toolset requirements, which are driven by the inputs from industrial applications in domains such as aerospace, automotive, video surveillance or wind power generation. The most relevant requirements are:

- Development of mixed-criticality systems: The concept of criticality should be central in the system.

[1] Automated proof-based System and Software Engineering for Real-Time systems. FP6 IST 004033.

[2] Composition with Guarantees for High-integrity Embedded Software Components Assembly, ARTEMIS-2008-1-100022.

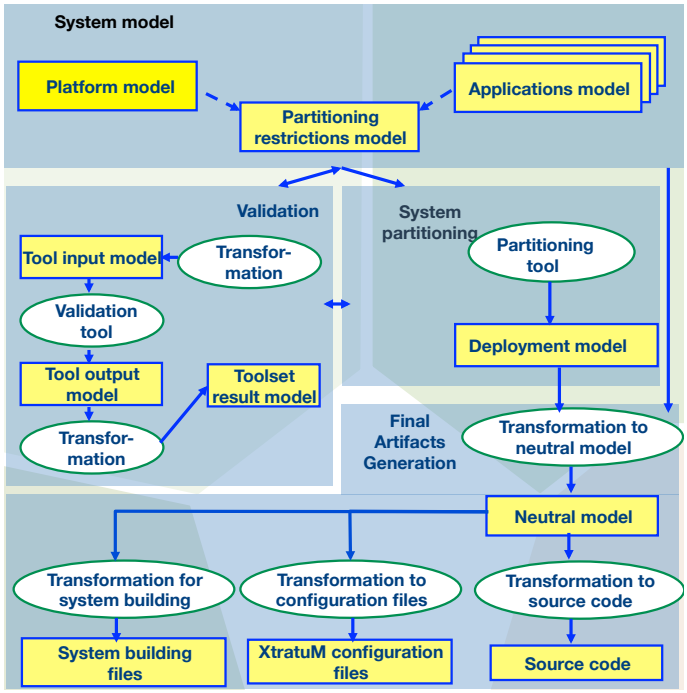


Figure 5: Architecture of the real-time safety systems development framework.

- Support for non-functional requirements: The framework has to provide mechanisms for specifying and validating these requirements. Additionally, the integration of requirements from different components must be supported. Currently safety, real-time, and security requirements are being considered.
- Support for multi-core architectures: Current processor technologies are more and more based on multicores. Design aspects such as modelling, partition allocation, or response time analysis, should be supported.
- System modelling: The toolset must provide means for modelling the whole system, including the applications, platform, and any other elements required for its description.
- Support for system deployment: this implies the generation of a bootable software image with the hypervisor and the partitions code, including the operating systems and applications allocated to them.

The current framework design is shown in figure 5. A system model is composed of three models:

- Platform model: it describes the execution platform, including hardware, hypervisor, operating system. The platform model relies on UML-MARTE, with some extensions.
- Applications model: it includes the functional model and the required non-functional properties.
- Partitioning restrictions model: it describes the restrictions to be fulfilled by a valid partitioning of the system.

Platform and applications models are independent of a particular system. In this way, it is possible to reuse them in different developments. The restrictions model includes information that applies to a particular system, and any specific criteria for partitioning. Restrictions may include statements that must be fulfilled by a valid partitioning, such as “an application must be allocated to a given partition”, “an application must (not) be in the same partition as another one”, “an application requires a particular hardware device”, or “a partition or application must run on a given core or processor”.

The system partitioning component is in charge of generating a valid system partitioning, i.e. a number of partitions, an allocation of applications to partitions, and an assignment of computational resources to partitions. This information is described in the deployment model. A deployment model must meet the defined restrictions, as well as the specification of non-functional requirements. For example, if an application has a certain criticality level, it must not be allocated to the same partition as a non-critical application.

Some non-functional requirements may be difficult to validate. The validation component can provide validation tools for some of them. For example, a validation tool can support the validation of real-time requirements by carrying out a response time analysis of the system.

As above, inputs to validation processes can be generated automatically. Failure to validate one or more requirements can provide feedback to add restrictions to the partitioning model, so that an alternative deployment model can be produced.

The final step of development consists of the generation of the following final artefacts:

- Hypervisor configuration files, for implementing a behaviour compliant with the deployment model.
- System building files, for automatically generating the final executable system.

- Source code skeletons, for the main entities.

The framework is currently under development. There are working versions of system model, and the artifacts generation tools. With respect to the partitioning component, it is possible to define partitions manually. The automatic partitioning algorithm is under development. Finally, there is an ongoing work on the support for response time analysis of partitioned multi-core systems. The framework is currently tailored for a LEON3 hardware architecture, the XtratuM hypervisor, and the ORK+ operating system. Code skeletons generation is targeted to Ravenscar Ada.

4 Software validation approach

The flight version of the on board computer, based on a LEON3 processor, is still under development. For this reason, an engineering model is currently being used for preliminary software validation. The engineering model is based on a GR-XC3S1500 Spartan3 development board with a LEON2 processor at 40 MHz clock frequency and 64 MB of SDRAM. Cache memory is not used in this implementation. The main difference between the LEON2 processor used in the engineering model and the envisaged production LEON3 are that the latter has a 7-stage pipeline instead of the 5-stage pipeline of LEON2. The differences between these versions of the processors are not significant as they are not central for system behaviour.

The engineering version of the OBC is being used to test and analyse some parts of the software that already mature enough for preliminary validation. For example, there is a working version of the ADCS subsystem, implementing an elaborate attitude control algorithm that has been designed by aerospace engineers, based on a mathematical model of the spacecraft dynamics and the torque perturbations. Due to the complexity of the model, a functional model has been created with Simulink in order to design, test and validate the structure of the control algorithm and to tune its parameters to the most appropriate values.

As it is not possible to test the satellite software in its real environment, a software validation facility (SVF) including hardware-in-the-loop (HIL) simulation has been built. The basic idea is to test the embedded system against a simulation model instead of the real environment. The test environment includes a simulation model that interacts with the control module running on the real computer, as shown in figure 6. Some additional components have been included as well, in order to model the sensors and actuators that carry out the interaction between the computer and the modelled environment. The tests performed so far show a correct behaviour of the attitude control software. The SVF approach has also been used to analyse the worst-case execution time and the maximum response time of the attitude control procedure [17].

In order to validate the partitioned architecture described in 2.3, a prototype has been built with the two partitions shown in figure 4b. The partitioned system runs on the on-board computer. The platform management partition interacts with the SVF computer. The ADCS partition executes the control

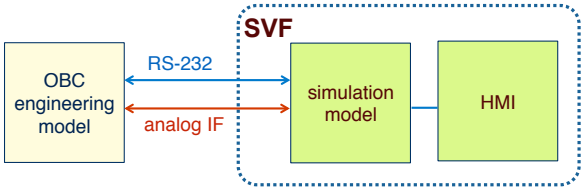


Figure 6: Architecture of the software validation facility.

algorithm, and interacts with the other partition in order to exchange data with the SVF simulation of the spacecraft dynamics.

The results of this exercise have been quite promising. System partitioning has been straightforward using the development framework. The original application was split into two components that were allocated to different partitions. Communication was performed using the XtratuM mechanisms for inter-partition interaction. Partitioning and resource assignment have been done manually. The generation of the artefacts has simplified the deployment of the final system. The next steps include to continue the UPMSat-2 development, and to apply this technology to a more complex system. We are also planning to make an assessment of the response time analysis facilities, in order to guarantee the required timing behaviour.

5 Conclusions and future plans

Model-driven engineering has lived up to its promise to raise abstraction level and make software development easier. Being able to reason with models in the development of the UPMSat-2 software is a real gain over older development methods, and lets the design team concentrate on the system behaviour rather than implementation details.

The use of the TASTE toolset for the monolithic software version is straightforward. We are modelling most of the system behaviour with SDL, except for the ADCS component which is being modelled with Simulink. Automatic code generation from the models has enabled experimenting with the ADCS algorithm and carry out analysis and testing procedures on this subsystem at an early stage. The implementation of this software version on the monolithic GNAT/ORK/LEON3 platform is also straightforward, as this is a well-known and proven platform for space systems.

The partitioned version of the software leaves room for experimenting newer software development methods and tools. Using a partitioned architecture allows the design engineers to separate subsystems from each other and assigning them different criticality levels. For example, the system manager, the ADCS, and the communications (TTC) subsystems can be assigned level B (as per ECSS-Q-ST-80), whereas the experiments in the payload can be considered level C, as they are not critical for the success of the mission. Even subsystems with the same criticality level can profit from partitioning, as they can be validated independently, thus simplifying qualification. However, the virtualization kernel has to be qualified at the maximum criticality level of the partitions, which may in turn make the qualification process more complex. This

issue is being addressed in the MultiPARTES project, where a roadmap to the qualification of the satellite software case study based on UPMSat-2 is being developed.

Effective use of a partitioned approach such as discussed above requires support from a toolset that integrates partitioning with modelling and code generation. The toolset that has been described in the paper has already shown a high potential for this development paradigm, and is also being completed in the framework of MultiPARTES.

Other topics dealt with in the paper, such as the use of a software validation facility with a hardware-in-the-loop configuration and the way to overcome some minor inconsistencies in the design of actuator control tasks, are directly extracted from the authors' experience and have also contributed to clarifying the development process

Plans for the near future include completing the design and implementation of the monolithic software system, and carrying out the complete validation and qualification activities as required by the ESA standards. This requires testing on the flight computer with the actual I/O devices.

With respect to the partitioned software systems, the immediate tasks are completing the toolset, finalise the software design (note that the functional model is the same as in the monolithic version), and complete the roadmap to ESA qualification.

Acknowledgments.

The work described in this paper has been partially funded by the Spanish Government, project HI-PARTES (TIN2011-28567-C03-01), and by the European Commission FP7 programme, project MultiPARTES (IST 287702).

The UPMSat-2 project is led by IDR/UPM.[3] We would like to acknowledge the collaboration of the IDR team, TECNOBIT, as well as the MultiPARTES consortium members.

References

[1] *LEON3 - High-performance SPARC V8 32-bit Processor. GRLIB IP Core User's Manual* (2012).

[2] European Cooperation for Space Standardization (2009), *ECSS-E-ST-40C Space engineering — Software*, Mar. Available from ESA.

[3] European Cooperation for Space Standardization (2009), *ECSS-Q-ST-80C Space Product Assurance — Software Product Assurance*, Mar. 2009. Available from ESA.

[4] A. Esquinas, J. Zamorano, J. A. de la Puente, M. Masmano, I. Ripoll, and A. Crespo (2011), *ORK+/XtratuM: An open partitioning platform for Ada*, in Reliable Software Technologies — Ada-Europe 2011 (A. Romanovsky and T. Vardanega, eds.), no. 6652 in LNCS, pp. 160–173, Springer-Verlag, 2011.

[5] P. Fortescue, G. Swinerd, and J. Stark (2011), *Spacecraft Systems Engineering*. Wiley, 4 ed.

[6] V. Bos, P. Mendham, P. Kauppinen, N. Holsti, A. Crespo, M. Masmano, J. de la Puente, and J. Zamorano (2013), *Time and space partitioning the EagleEye Reference Mission*, in Data Systems in Aerospace — DASIA 2013, (Porto, Portugal).

[7] S. Trujillo, A. Crespo, and A. Alonso (2013), *MultiPARTES: Multicore virtualization for mixed-criticality systems*, in Euromicro Conference on Digital System Design, DSD 2013, pp. 260–265.

[8] M. Masmano, I. Ripoll, A. Crespo, and S. Peiró (2010), *XtratuM for LEON3: An opensource hypervisor for high-integrity systems*, in Embedded Real Time Software and Systems — ERTS2 2010, (Toulouse (France)).

[9] M. Perrotin, E. Conquet, P. Dissaux, T. Tsiodras, and J. Hugues (2010), *The TASTE toolset: Turning human designed heterogeneous systems into computer built homogeneous software*, in Embedded Real Time Software and Systems — ERTS2 2010, (Toulouse (France)).

[10] Mathworks, *Simulink*, 2013.

[11] ITU (2011), *Specification and Design Language – Overview of SDL-2010*. Recommendation ITU-T Z.100.

[12] P. Feiler (2012), *Architecture Analysis & Design Language — SAE AADL AS5506B*. SAE.

[13] E. Salazar, A. Alonso, M. A. de Miguel, and J. A. de la Puente (2013),*A model-based framework for developing real-time safety Ada systems," in Reliable Software Technologies - Ada-Europe 2013* (H. Keller, E. Plödereder, P. Dencker, and H. Klenk, eds.), vol. 7896 of Lecture Notes in Computer Science, pp. 127–142, Springer Berlin Heidelberg.

[14] OMG (2011), *Unified Modeling Language (UML)*. Version 2.4.1.

[15] OMG (2011), *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. Version 1.1.

[16] A. Alonso, E. Salazar, and J. A. de la Puente (2014), *A toolset for the development of mixed-criticality partitioned systems*, in 2nd Workshop on High-performance and Real-time Embedded Systems.

[17] J. Garrido, D. Brosnan, J. A. de la Puente, A. Alonso, and J. Zamorano (2012), *Analysis of WCET in an experimental satellite software development*, in 12th International Workshop on Worst-Case Execution Time Analysis (T. Vardanega, ed.), vol. 23 of OpenAccess Series in Informatics (OASIcs), pp. 81–90.

[18] J. Garrido, J. Zamorano, and J. A. de la Puente (2013), *Static analysis of WCET in a satellite software subsystem*," in 13th International Workshop on Worst-Case Execution Time Analysis (C. Maiza, ed.), vol. 30 of OpenAccess Series in Informatics (OASIcs), pp. 87–96.

[3] Instituto Ignacio da Riva, `www.idr.upm.es`.

SPARK 2014 Rationale: Formal Containers

Claire Dross

AdaCore, France

Abstract

This paper continues the publication of the "SPARK 2014 Rationale", which was started in the previous issue of the Ada User Journal. In this instalment, we present three contributions on the use of formal containers in SPARK.

1 Formal Containers

SPARK 2014 excludes data structures based on pointers. Instead, one can use the library of formal containers. They are variant of the Ada 2012 bounded containers, specifically designed and annotated to facilitate the proof of programs using them.

To work around the previous restriction, the content of a data structure can be hidden using private types. Thanks to specification functions, a model can be defined for the content of these data structures that can then be used to specify the API functions. For example, here is a SPARK 2014 specification of a linked list package using pointers. It uses a specification function Get_Model that returns the content of a linked list as an array:

```
package Lists is

   pragma SPARK_Mode (On);

   type Int_Array is array (Positive range <>) of
                                                Integer;
   type My_List is private;

   function Get_Model (L : My_List) return Int_Array;

   function Head (L : My_List) return Integer with
       Post => Get_Model (L) (1) = Head'Result;

   function Cons (I : Integer; L : My_List)
       return My_List
     with Pre  => Get_Model (L)'Last < Natural'Last,
       Post => Get_Model (Cons'Result) = I &
                  Get_Model (L);

private
   pragma SPARK_Mode (Off);

   type My_List_Record is record
     Head : Integer;
     Tail : My_List;
   end record;

   type My_List is access My_List_Record;

end Lists;
```

Instead of implementing such a linked list package herself, a user may want to use a generic container package from the library. For example, the package Ada.Containers.Doubly_Linked_Lists offers two sub-programs, First_Element and Prepend, that can be used in place of Head and Cons. Unfortunately, these packages are not adapted to formal verification. In particular, cursors, that are iterators over containers, contain an internal reference to their container. As a consequence, every sub-program that modifies a container also silently modifies every cursor that references it.

As part of the SPARK 2014 development, new containers library have been introduced. Those formal containers, available under Ada.Containers.Formal_<something>, closely resemble Ada 2012 bounded containers except that cursors are not tied to a particular container and can be valid in several of them. This modification allows in particular to refer to the element designated by a given cursor Cu in a container Cont before and after a modification of Cont using the same cursor. For example, we can specify a procedure that takes a list of integers L and a cursor Cu and increments the element designated by Cu in L:

```
package My_Lists is new
Ada.Containers.Formal_Doubly_Linked_Lists
 (Element_Type, My_Eq);

use My_Lists;

procedure Increment_Element (L : in out List;
                             Cu : Cursor) with
  Pre  => Has_Element (L, Cu) and then
          Element (L, Cu) < Element_Type'Last,
  Post => Has_Element (L, Cu) and
          Element (L, Cu) = Element (L'Old, Cu) + 1;
```

Note that the specification of the procedure Increment_Element is not complete. Indeed, it does not state that the rest of the list L is not modified, that is, that it contains the same elements in the same order and that iteration can be done on L before and after the modification using the same cursors. These frame condition statements are common enough for the formal containers packages to include specific functions that facilitate their expression.

The function First_To_Previous takes a container Cont and a cursor Cu in argument and returns a container that is Cont where Cu and every cursor following it in Cont have been removed:

```
function First_To_Previous (Container : List;
                            Current : Cursor) return List;
```

The function Current_To_Last takes a container Cont and a cursor Cu in argument and returns a container that is Cont where every cursor preceding Cu in Cont have been removed:

```
function Current_To_Last (Container : List;
                          Current : Cursor) return List;
```

Finally, the function Strict_Equal takes two containers and returns true if and only if they contain the same elements in the same order and iteration can be done on both of them using the same cursors.

```
function Strict_Equal (Left, Right : List)
                                    return Boolean;
```

Thanks to these function, we can complete the specification of Increment_Element:

```
procedure Increment_Element (L : in out List;
                             Cu : Cursor) with
  Pre  => Has_Element (L, Cu) and then
          Element (L, Cu) < Element_Type'Last,
  Post => Has_Element (L, Cu) and then
          Element (L, Cu) = Element (L'Old, Cu) + 1
        and then
          Strict_Equal (First_To_Previous (L, Cu),
                        First_To_Previous (L'Old, Cu))
        and then
          Strict_Equal (Current_To_Last (L, Next(L, Cu)),
                Current_To_Last (L'Old, Next(L'Old, Cu)));
```

The implementation of Increment_Element is as simple as calling procedure Replace_Element from the formal containers' API:

```
procedure Increment_Element (L : in out List;
                             Cu : Cursor) is
begin
  Replace_Element (L, Cu, Element (L, Cu) + 1);
end Increment_Element;
```

And guess what? GNATprove manages to prove automatically that the implementation above indeed implements the contract that we specified for Increment_Element. And that no run-time errors can be raised in this code. Not bad for a non-trivial specification!

2 Expressing Properties over Formal Containers

We saw in the previous post how formal containers can be used in SPARK code. In this post, I describe how to express properties over the content of these containers, using quantified expressions. In their simplest form, quantified expressions in Ada 2012 can be used to express a property over a scalar range. For example, that all integers between 1 and 6 have a square less than 40:

```
(for all J in 1 .. 6 => J * J < 40)
```

or that every even integer greater than 2 can be expressed as the sum of two primes (also known as Goldbach's conjecture):

```
(for all J in Integer =>
  (if J > 2 then
    (for some P in 1 .. J / 2 => Is_Prime (P) and then
       Is_Prime (J - P))))
```

The second form of quantified expressions allows to express a property over a standard container. For example, that all elements of a list of integers are prime, which can be expressed by iterating over cursors as follows:

```
(for all Cu in My_List => Is_Prime (Element (Cu)))
```

The general mechanism in Ada 2012 that provides this functionality relies on the use of tagged types (for the container type) and various aspects involving access types so cannot be applied to the SPARK formal containers.

Instead, we have defined in GNAT an aspect Iterable that provides the same functionality in a simpler way, leading also to much simpler object code. For example, here is how it can be used on a type Container_Type:

```
type Container_Type is ... -- the structure on which we
                           -- want to quantify
  with Iterable => (First => My_First,
                    Has_Element => My_Has_Element,
                    Next => My_Next);
```

where My_First is a function taking a single argument of type Container_Type and returning a cursor:

```
function My_First (Cont : Container_Type)
                   return My_Cursor_Type;
```

and My_Has_Element is a function taking a container and a cursor and returning whether this cursor has an associated element in the container:

```
function My_Has_Element (Cont : Container_Type;
                         Pos : My_Cursor_Type) return Boolean;
```

and My_Next is a function taking a container and a cursor and returning the next cursor in this container:

```
function My_Next (Cont : Container_Type; Pos :
                  My_Cursor_Type) return My_Cursor_Type;
```

Now, if the type of object Cont is iterable in the sense given above, it is possible to express a property over all elements in Cont as follows:

```
(for all Cu in Cont => Property (Cu))
```

The compiler will generate code that iterates over Cont using the functions My_First, My_Has_Element and My_Next given in the Iterable aspect, so that the above is equivalent to:

```
declare
  Cu : My_Cursor_Type := My_First (Cont);
  Result : Boolean := True;
```

```
begin
  while Result and then My_Has_Element (Cont, Cu)
loop
    Result := Result and Property (Cu);
    Cu := My_Next (Cont, Cu);
  end loop;
end;
```

where Result is the value of the quantified expression.

We have used the Iterable aspect to provide quantification for formal containers, using the functions First, Has_Element and Next of the formal containers' API. For example, the definition of formal doubly linked lists looks like:

```
type List (Capacity : Count_Type) is private with
  Iterable => (First  => First,
               Next => Next,
               Has_Element => Has_Element);
```

This allows a user to specify easily the contract of a function that searches for the first occurrence of 0 in a doubly linked list, using here contract cases:

```
function Search (L : List) return Cursor with
  Contract_Cases =>
   -- first case: 0 not in the list
   ((for all Cu in L => Element (L, Cu) /= 0) =>
      Search'Result = No_Element,

   -- second case: 0 is in the list
    (for some Cu in L => Element (L, Cu) = 0) =>
      Element (L, Search'Result) = 0
        and then
      (for all Cu in First_To_Previous (L, Search'Result)
                => Element (L, Cu) /= 0));
```

The first case specifies that, when the input list does not contain the value 0, then the result is the special cursor No_Element. The second case specifies that, when the input list contains the value 0, then the result is the first cursor in the list that has this value.

3 Verifying Properties over Formal Containers

We saw in the previous post how we could express complex properties over formal containers using quantified expressions. In this post, I present how these properties can be verified by the proof tool called GNATprove.

A naive support for formal containers would consist in adding contracts to all subprograms in the API of formal containers, and let GNATprove analyze calls to these subprogram as it does for regular code. The problem with that approach is that, either we add lightweight contracts which don't allow proving all properties of interest, or we add heavyweight contracts that make it harder to prove properties automatically. So instead, we chose to "axiomatize" the library of formal containers, that is, we directly wrote axioms explaining to the prover how they work. This particularity is specified using a specific annotation pragma in the implementation of formal containers:

```
pragma Annotate (GNATprove,
                 External_Axiomatization);
```

GNATprove makes some hypothesis on the function parameters used for the instantiation of a formal container. Let's see how this works on My_Sets, a set of integers defined as follows:

```
type Element_Type is new Integer range 1 .. 100;

function Hash (Id : Element_Type) return Hash_Type;

function Equivalent_Elements (I1, I2 : Element_Type)
                              return Boolean;

function My_Equal (I1, I2 : Element_Type)
                              return Boolean is
(I1 = I2);

package My_Sets is new
Ada.Containers.Formal_Hashed_Sets
   (Element_Type  => Element_Type,
    Hash  => Hash,
    Equivalent_Elements => Equivalent_Elements,
    "=" => My_Equal);
```

Note that we do not support passing operators as actuals in a generic instantiation, so we cannot leave the default argument for the parameter "=" nor explicitly give the equality symbol "=" as an argument. We need to introduce a function wrapper My_Equal.

For GNATprove to give correct results, the arguments of the generic instantiation must respect some crucial properties:

1. The functions Hash, Equivalent_Elements, and My_Equal must not read or write any global.
2. Then, as specified in the Ada reference manual, the function Equivalent_Elements should be an equivalence relation.

GNATprove uses these properties in its proofs, so, for example, it can always prove automatically that the following properties hold for Equivalence_Elements:

```
-- Reflexivity
pragma Assert (for all E in Element_Type =>
                 Equivalent_Elements (E, E));

-- Symmetry
pragma Assert
  (for all E1 in Element_Type =>
     (for all E2 in Element_Type =>
        (if Equivalent_Elements (E1, E2) then
           Equivalent_Elements (E2, E1))));

-- Transitivity
pragma Assert
  (for all E1 in Element_Type =>
     (for all E2 in Element_Type =>
        (for all E3 in Element_Type =>
```

```
            (if Equivalent_Elements (E1, E2) and
                Equivalent_Elements (E2, E3) then
               Equivalent_Elements (E1, E2)))));
```

Let us now see how we can prove programs using My_Sets. As an example, let us consider a function Double_All that takes a set of elements S and returns the set of all the doubles of elements of S:

```
-- A subtype of Set with Capacity 100.
-- The Modulus discriminant is used for the hash
-- function.
subtype My_Set is Set (Capacity => 100,
                       Modulus => Default_Modulus (100));

function Double_All (S : My_Set) return My_Set with
  Pre  => (for all E of S => E <= 50),
  Post => (for all E of S => Contains
                          (Double_All'Result, 2 * E));
```

The precondition of function Double_All states that it is called on a set whose elements are all less than or equal to 50, using a quantified expression. Its postcondition states that the result of the function contains all the doubles of the elements of the function argument.

Here is a possible implementation for Double_All:

```
function Double_All (S : My_Set) return My_Set is
  R : My_Set;
  Current : Cursor := First (S);
begin
  Clear (R);
  while Has_Element (S, Current) loop
    pragma Loop_Invariant
      (Length (R) <= Length (First_To_Previous
                             (S, Current))
        and then
       (for all Cu in First_To_Previous (S, Current) =>
          Contains (R, 2 * Element (S, Cu))));
    Include (R, 2 * Element (S, Current));
    Next (S, Current);
  end loop;
  return R;
end Double_All;
```

Note that we use a call to the procedure Clear at the beginning of the program for GNATprove to know that the set R is empty. The proof of Double_All requires a loop invariant. It states that we have already included in R the elements that appear in S before the Current cursor using the function First_To_Previous. We also need to bound the length of R to be able to prove that we do not exceed the capacity of R. Remark that we cannot prove that the length of R is equal to the number of already included elements. Indeed, since we do not know the definition of Equivalent_Elements, it could be the case that two elements of S have the same double modulo this relation.

GNATprove proves automatically that the code of Double_All cannot raise a run-time exception, and that is implements the contract of the function.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/adaanswers/gems.

Gem #153: Multicore Maze Solving, Part 1

Pat Rogers, AdaCore

Abstract. This Gem series introduces the "amazing" project included with the GNAT Pro compiler examples. The project is so named because it involves maze solving (as in "Joe and Julie go a-mazing"). But these aren't typical mazes that have only one solution. These mazes can have many solutions, tens of thousands, for example. The point is to find all of them as quickly as possible. Therefore, we solve the mazes concurrently, applying multiple CPUs in a divide-and-conquer design. In this first Gem we introduce the program and explain the approach.

Let's get started…

This Gem series introduces the "amazing" project included with the GNAT Pro compiler examples. The project is so named because it involves maze solving (as in "Joe and Julie go a-mazing"). But these aren't typical mazes that have only one solution. These mazes can have many solutions, tens of thousands, for example. The point is to find all of them as quickly as possible. Therefore, we solve the mazes concurrently, applying multiple CPUs in a divide-and-conquer design. In this first Gem we introduce the program and explain the approach.

We actually have two programs for maze solving: one sequential and one concurrent. Based on the notion of a mouse solving a maze, the sequential program is named mouse and the concurrent version is named – you guessed it – mice. Both are invoked from the command line with required and optional switches. The available switches vary somewhat between the two, but in both cases you can either generate and solve a new maze or re-solve a maze previously generated.

When generating a new maze you have the option to make it "perfect", that is, to have only one exit. Otherwise the maze will have an unknown number of solutions. For our purposes we use mazes that are not perfect, and in fact the number of solutions depends solely on the size of the mazes and the random way in which they are generated.

Switches "-h" and "-w" allow you to specify the height and width of a new maze, but other than that their layout is randomly determined. In addition, the concurrent program allows you to specify the total number of tasks available for solving the maze using the "-t" switch. This switch is useful for experimentation, for example in determining the effect of having a great many tasks, or in determining the optimal number of tasks relative to the number of processors available. There are four tasks available by default. The concurrent program will run on as many or as few processors as are available.

Finally, you can control whether a maze and its solutions are displayed. At first glance this might seem a strange option, but displaying them makes the program heavily I/O-bound and serialized, hiding the benefits of parallelism and making it difficult to determine the effects of design changes. Disabling the display is achieved via the "-q" switch.

After either program solves a new maze, you are asked whether you want to keep it. If so, you specify the file name and the program then writes it out as a stream. To re-solve an existing maze you specify both the "-f" switch and the file's name.

As the programs execute they display the maze, the unique solutions through the maze, and the running total of the number of solutions discovered (unless the "-q" switch is applied). Currently there are two kinds of "console" supported for depicting this information. The selection is determined when building the executables, under the control of a scenario variable having possible values "Win32" and "ANSI".(Terminals supporting ANSI escape sequences are common on Linux systems, so there is a wide range of supported machines.

Now that you know what the programs can do, let's see how they do it.

The sequential mouse program illustrates the fundamental approach. As it traverses the maze, it detects junctions in the path where more than one way forward is possible. There may be three ways forward, in fact, but not four because that would involve going back over the location just visited. Hence, at any junction the mouse saves all but one of the other alternative locations, along with the potential solution as it is currently known, and then pursues that one remaining lead. Whenever the mouse can go no further – either because it has encountered a dead end or because it has found the maze exit – it restores one of the previously saved alternative location/solution pairs and proceeds from there. The program is finished when the mouse can go no further and no previous alternatives are stored.

The mice program uses the same basic approach, except it does it concurrently. A "searcher" task type implements the sequential mouse behavior, but instead of storing the alternatives at junctions, it assigns a new searcher task to each of the alternatives. These new searchers continue concurrently (or in parallel) with the searcher that assigned them, themselves assigning new searchers at any junctions they encounter. Only when no additional searcher task is available does any given searcher store alternative leads for later pursuit. If it does restore a lead, it uses the same approach at any further junctions encountered.

A new searcher task may be unavailable when requested, because we use a pool of searcher instances, with a capacity controlled by the command-line parameter. When no further progress is possible, a searcher task puts itself back into this pool for later assignment, so additional searchers may be available when restored leads are pursued. The main program

waits for all the searchers to be quiescent, waiting in the pool for assignments, before finishing.

The body for the Searcher task type (declared in package Search_Team) implementing this behavior follows:

```
task body Searcher is
   Path : Traversal.Trail;
   The_Maze : constant Maze.Reference :=
                                        Maze.Instance;
   Current_Position : Maze.Position;
   Myself : Volunteer;
   Unsearched : Search_Leads.Repository;
begin
   loop
      select
         accept Start (My_ID : Volunteer;
                        Start : Maze.Position;
                        Track : Traversal.Trail)
         do
            Myself := My_ID;
            Current_Position := Start;
            Path := Track;
         end Start;
      or
         terminate;
      end select;

      Searching : loop
         Pursue_Lead (Current_Position, Path,
                      Unsearched);

         if The_Maze.At_Exit (Current_Position) then
            Traversal.Threaded_Display.Show (Path,
                                        On => The_Maze);
         end if;

         exit Searching when Unsearched.Empty;

         -- Go back to a position encountered earlier that
         -- could not be delegated at the time.
         Unsearched.Restore (Current_Position, Path);
      end loop Searching;

      Pool.Return_Member (Myself);
   end loop;
end Searcher;
```

The Searcher task first suspends, awaiting either initiation, to start pursuing a lead, or termination. The rendezvous thus provides the initial location and the currently known solution path. The parameter My_Id is a reference to that same task and is used by the task to return itself back into the pool, when searching is finished. The accept body simply copies these parameters to the local variables. The other local variables include a reference to the maze itself (we use a singleton for that) and the repository of unsearched leads, used to store position/solution pairs for future pursuit.

As the task searches for the exit, procedure Pursue_Lead delegates new searcher tasks to alternatives when junctions are encountered. The procedure returns when no further progress can be made on a given lead. In effect we "flood" the maze with searcher tasks, so this is a divide-and-conquer design typical of classical concurrent programming.

In the next Gem in this series, we will describe a fundamental implementation change made very recently (September 2013) to the original concurrent program. This change solved a critical performance bottleneck that was not present when the original program was first deployed in the 1980s, illustrating one of the fundamental differences between traditional multiprocessing and modern multicore programming.

As mentioned, the "amazing" project is supplied with the GNAT Pro native compiler. Look for it in the share/examples/gnat/amazing directory located under your compiler's root installation. Note that the design change will appear in future releases of the compiler.

Gem#154 Multicore Maze Solving, Part 2

Pat Rogers, AdaCore

Abstract. This series of Gems describes the concurrent maze solver project ("amazing") included with the GNAT Pro examples. The first Gem in the series introduced the project itself and explained the concurrent programming design approach. This second Gem explores the principal change that was required for optimal performance on multicore architectures. This change solved a critical performance bottleneck that was not present when the original program was first deployed in the 1980s, illustrating one of the fundamental differences between traditional multiprocessing and modern multicore programming.

Let's get started…

This series of Gems describes the concurrent maze solver project ("amazing") included with the GNAT Pro examples. The first Gem in the series introduced the project itself and explained the concurrent programming design approach. This second Gem explores the principal change that was required for optimal performance on multicore architectures. This change solved a critical performance bottleneck that was not present when the original program was first deployed in the 1980s, illustrating one of the fundamental differences between traditional multiprocessing and modern multicore programming.

The original target machine was a Sequent Balance 8000, a symmetric multiprocessor with eight CPUs and shared memory. The operating system transparently dispatched Ada tasks to processors, so one could write a highly portable concurrent Ada program for it. In the 1980s this was a very attractive machine, as you might imagine. The resulting program successfully demonstrated Ada's capability to harness such architectures, as well as the general benefits of parallel execution. In particular, the execution time for the sequential version of the maze solver grew at an alarming rate as the number of maze solutions grew larger, whereas the parallel version showed only modest increases. (Remember, the point is to find all the possible solutions to a given maze, not just one.)

The program was indeed highly portable and ran on a number of very different vendors' machines, some parallel and some not. Over time, we have incorporated the language revisions' advances, primarily protected types, and added features such as command-line switches for flexibility, but

the architecture and implementation have largely remained unchanged. Until recently, that is.

As described in the first Gem in this series, the program "floods" the maze with searcher tasks in a classic divide-and-conquer design, each searcher looking for the exit from a given starting point. The very first searcher starts at the maze entrance, of course, but as any searcher task encounters intersections in the maze, it assigns another identical task to each alternative location, keeping one for itself. Thus, a searcher task that finds the exit has discovered only part of a complete solution path through the maze. If the very first searcher happened to find the exit, it would have a complete solution, but all the other searchers have only a part of any given solution path because they did not start at the entrance.

As the searchers traverse the maze they keep track of the maze locations they visit so that those locations can be displayed if the exit is eventually found. But as we have seen, those locations comprise only a partial path through the maze. Therefore, when a successful searcher displays the entire solution it must also know the locations of the solution prior to its own starting point, as well as the locations it traversed itself to reach the exit. To address that requirement, when a searcher is initiated at a given starting location it is also given the current solution as it is known up to that location. The very first searcher is simply given an empty solution, known as a "trail" in the program. Successful searchers display both the part they discovered and the part they were given when started.

Note that these partial solutions are potentially shared, depending on the maze. (Solutions are unique if any constituent maze locations are different, but that does not preclude partial sharing.) Those maze locations closer to the entrance are likely to be heavily shared among a large number of unique solutions. Conceptually, the complete solutions form a tree of location sequences, with prior shared segments appearing earlier in the tree and unique subsegments appearing beneath them. The maze entrance appears once, in the root at the top of the tree, whereas the maze exit appears at the end of every solution.

Imagine, then, how one might want to represent this tree. Given that segments of the solutions – the trails – are likely shared logically, perhaps we can also share them physically. However, as a shared data structure, race conditions are an obvious concern. We therefore want a representation that will minimize the locking required for mutual exclusion. We also want a representation that can contain any number of location pairs per segment because the mazes are randomly generated initially. That is, we don't know how many locations any given solution will contain, much less how many solutions there will be.

An unbounded, dynamically allocated list of maze locations meets these goals nicely. It can directly represent the logical sharing and can handle trails of any length as long as sufficient memory is available. Even better, no mutual exclusion locking is required because we only need to append list segments to prior, existing segments. There is no need to alter the prior segments themselves, so there is no need to lock the tree at all!

The representation seems ideal, and for the original symmetric multiprocessor target it was a reasonable approach, but when the program was run on modern multicore machines the performance was very poor. Indeed, individual processor utilization was so poor that the sequential version of the maze solver was quite competitive with the concurrent version.

Poor processor utilization is the key to the problem. Even though we are harnessing multiple processors and can have as many threads available per processor as we may want, individual processors are performing poorly. The problem is caused by poor cache utilization, itself a result of poor locality of reference. Specifically, the dynamically allocated elements within the trails are not in memory locations sufficiently close to one another to be in the same cache line, thereby causing many cache misses and poor overall processor performance.

The issue is that searcher tasks must also examine the locations within their prior solution trails as they search for the exit. (In other words, not only when displaying solutions.) They do so to prevent false circular solutions through the maze, made possible by the presence of intersections. Therefore, the searcher tasks must determine whether they have been to a given location in the maze before including that location in their solution. Not all location pairs in a trail need be visited, however, to perform this check. The presence of an intersection in a prior path segment suffices to indicate circular motion, so each trail includes a list of intersections, and it is this secondary list that the searchers examine. Unfortunately any benefits of that implementation optimization are overwhelmed by the results of the cache misses.

A different trail representation is needed for programs intended for multicore targets, one with much better locality of reference. Arrays have that precise characteristic, so we have chosen a bounded, array-backed list to represent trails. That choice will not surprise those familiar with this problem, even though the resulting copying and lack of physical sharing would argue against it.

In the next Gem in this series we will provide the details of this implementation change and the reusable components involved.

As mentioned, the "amazing" project is supplied with the GNAT Pro native compiler. Look for it in the share/examples/gnat/amazing/ directory located under your compiler’s root installation. Note that the described design change will appear in future releases of the compiler.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o KU Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Hubert B. Keller
Karlsruher Institut für Technologie (KIT)
Institut für Angewandte Informatik (IAI)
Campus Nord, Gebäude 445, Raum 243
Postfach 3640
76021 Karlsruhe
Germany
Email: Hubert.Keller@kit.edu
URL: ada-deutschland.de

Ada-France

attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. Sergio Sáez
DISCA-ETSINF-Edificio 1G
Universitat Politècnica de València
Camino de Vera s/n
E46022 Valencia
Spain
Phone: +34-963-877-007, Ext. 75741
Email: ssaez@disca.upv.es
URL: www.adaspain.org

Ada in Sweden

attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada-Switzerland

c/o Ahlan Marriott
Altweg 5
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: president@ada-switzerland.ch
URL: www.ada-switzerland.ch