ADA USER JOURNAL

Volume 32
Number 3
September 2011

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

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

The Journal publishes the following types of material:

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

Further details on our approach to these are given below.

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

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. Since not all of our readers have access to resources such as the World Wide Web and Usenet, or have enough time to search through the information that can be found in those resources, we 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 to the Editor, preferably in electronic format. The Editor will only accept typed manuscripts by prior arrangement.
Prospective authors are encouraged to contact the Editor by email to determine the best format for submission. Contact details can be found near the front of each edition.
Example papers conforming to formatting requirements as well as some word processor templates are available from the editor. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

As we had put forward in the first issue of this year, the September issue of the Ada User Journal (Volume 32, Number 3) commences the publication of the Rationale for Ada 2012, an important reference document which is, as usual (and we are thankful for that), being prepared by John Barnes. The first instalment in this issue provides the introduction and an overview of the changes to the language; these will be subsequently detailed in forthcoming chapters. We are certain that the Journal readers will enjoy this contribution (as well as being eagerly waiting for the continuation) as much as we do.

Continuing with the contents of the issue, I would also like to draw your attention to the two papers derived from the Industrial Track of the Ada-Europe 2011 conference. In the first paper, Phil Thornley, from SPARKSure, UK, demonstrates how to prove the correctness of linked data structures with SPARK, providing also some conclusions about the use of proofs in the development process. In the second paper, Jacob Andersen, from Jacob Sparre Andersen Research & Innovation, Denmark, provides an interesting description of a language server which was developed in Ada for an online game, an application domain where we are not used to see Ada being used.

Also in this issue, we present an Ada User Guide, this time on the use of Ada for programming the LEGO Mindstorms platform. In this guide, a contribution from a group of authors from Universidad Politécnica de Madrid, Spain, the reader can find a description on how Ada, using a Ravenscar runtime together with some tools (such as a screwdriver), can be used to develop applications for this robotic kit.

Finally, and as usual, a note to the important information provided in the News, Calendar and Forthcoming Events sections. In particular, the latter provides the advance program of the SIGAda 2011 conference, which will take place next November in Denver, Colorado, USA, and an invitation, from the conference's General Chair, for the 17th International Conference on Reliable Software Technologies – Ada-Europe 2012, to take place June 2012 in Stockholm, Sweden.

Luís Miguel Pinho
Porto
September 2011
Email: lmp@isep.ipp.pt

Quarterly News Digest

Marco Panunzio

University of Padua. Email: panunzio@math.unipd.it

Contents

Ada-related Organizations

30th Anniversary Issue of the AUJ available online

From: Ada-Europe website
Date: Mon, 20 Jun 2011
Subject: On-line Availability of Ada User Journal 30th Anniversary Special Issue
URL: http://www.ada-europe.org/Press_releases/press-release-AUJ.pdf

FOR IMMEDIATE RELEASE

On-line Availability of Ada User Journal 30th Anniversary Special Issue

EDINBURGH, UK (June 20, 2011) – On the occasion of Ada-Europe 2011, the 16th annual Conference on Reliable Software Technologies, Ada-Europe announces the on-line availability of the Ada User Journal 30th Anniversary special issue.

Ada User Journal, the quarterly publication of Ada-Europe, keeps its readership abreast of developments in the standardization, use and promotion of the Ada programming language and technology, as well as issues related with reliable software technologies and engineering in Europe and the rest of the world.

The origins of the Ada User Journal date back to the birth of Ada UK News, which started publication in March 1980. The current name of the Journal first appeared in Volume 15 in the year 1994, when it was still published by Ada UK. Ada-Europe published the Ada-Europe News since June 1989, until it was merged with the Ada User Journal in March 1998. From that time onward, Ada-Europe and Ada UK jointly published the Journal until Ada-Europe took over as the sole publisher from Volume 23 in 2002.

The celebration of the 30th anniversary of the Journal started in the March 2009 with a special article entitled "Thirty years of the Ada User Journal", recalling its three decades of history. Celebratory posters were also exhibited at various Ada-Europe and SIGAda conferences. The celebration closed with a special issue of the Journal, released in March 2010, the official 30th Anniversary Issue, reprinting a selection of the best articles published in the Journal over the past 30 years, selected by a prominent group of Guest Editors.

The Ada User Journal maintains an on-line accessible archive, a work-in-progress resource offered to the Ada community, for browsing, consulting and downloading selected contents of back issues of the Ada User Journal. This Online Archive provides the full contents of all issues dated from March 2001 (Vol. 22, N. 1) to the last-but-four issues. For the latest year, the Online Archives solely provides the table of contents.

It is in this context that the March 2010 issue has been released in the on-line archive, providing the Ada community with free access to "a sample of the papers that may be considered to have had the most impact and relevance at the time of publication, in the several incarnations of the Ada User Journal."

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of Ada into academia, research and industry in Europe.

Current member organizations of Ada-Europe are: Ada-Belgium, Ada in Denmark, Ada-Deutschland, Ada-France, Ada-Spain, Ada in Sweden and Ada-Switzerland. Ada-Europe also includes and welcomes individual members from other European countries with no national organization.

A PDF version of this press release is available at www.ada-europe.org.

Press contact

Dirk Craeynest, Ada-Europe Vice-President, Dirk.Craeynest@cs.kuleuven.be

New website for Ada-Switzerland

From: Thomas Løcke <tl@ada-dk.org>
Date: Tue, 19 Jul 2011
Subject: Ada Programming in Switzerland
URL: http://ada-dk.org/?page=news&news_id=336

As you might know, I both read and post to the identi.ca Ada group, and this morning I read a notice from Gautier de Montmollin about the new Ada-Switzerland website, and I immediately payed them a visit. I'm glad to report that their website is clean, modern and easy to navigate. Also after having been there, it appears to me that Ada is actually doing very well in Switzerland, which of course would explain the famed Swiss efficiency. If all their important infrastructure is powered by Ada, then the Swiss obviously aren't wasting time fixing bugs. :o)

In Denmark it appears that no software is built using Ada. I can't remember a single large public software system ever completed on time and within budget.

And when these large, expensive, systems are finally "completed", for the first couple of years they usually don't work. It's a complete and utter mess.

Perhaps it's time we ask our politicians and decision-makers to divert their eyes towards Switzerland where there appears to be a more widespread usage of Ada in complicated, critical and important software systems.

Well, enough of me ranting. Let me end this post by congratulating Ada in Switzerland on their beautiful new website.

I look forward to reading more about the various Swiss Ada projects.

[http://www.ada-switzerland.ch/projects.aspx —mp]

Ada-related Events

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

The Ada-Connection — Report by an exhibitor

From: Objektum Solutions' blog "The Technical Diaries"
Date: Wed, 22 Jun 2011
Subject: Day 1 Exhibiting at The Ada Connection

URL: http://the-technical-diaries.blogspot.com/2011/06/day-1-at-ada-connection.html

A speedy boarding (and disembarking) Ajay Patel and I stepped down off our easyJet flight in Edinburgh on Monday night ready for The Ada Connection the next day.

The conference was already in full swing and the marquee (or tent for our American readers) was ready for the eager exhibitors to fill on the Tuesday and Wednesday.

Tuesday morning was an early rise and we headed to the John McIntyre Conference Centre on the city's university campus. Thanks to the taxi driver's directions, we had a brisk morning walk around the campus and made sure that the delivery entrances were in full working order before finding the correct entrance and a warm welcome from Joan Atkinson and Tom Anderson, from the University of Newcastle, the conference organisers.

Our stand was erected and the table adorned with data sheets, flyers and the like. Whilst we waited for the delegates to wander in for their morning coffee and pastries, we had a chance to chat to fellow exhibitors - LDRA, Resource Engineering Projects, IPL, Ellidiss Software, AdaCore, Rapita Systems, Verocel and Wind River to name a few.

The delegates filtered in and avoided us until they had their first hit of caffeine. Once they were all well-oiled, we were good to go and within 30 minutes had two companies interested in the Objektum Bridge Suite migration technology and asking for webinars. Legacy migration and obsolescence is a serious challenge facing most organizations and so there were plenty more cards swapped and details taken down throughout the rest of the day.

Lunch was in the student cafeteria but it could have well been a decent restaurant. As one observant exhibitor pointed out, the only give-away was the copious amount of baked beans available.

And then came the rain. Lots and lots of it. Being positioned in the marquee, a short dash from the main conference building and through the waterfall above the door, a couple of us were asking whether the delegates would venture outside or whether they would remain in the dryer foyer area leaving us out in the cold and all alone. A few, including myself, were asking if there were any more heaters were available… The organisers, Tom, Joan and Steve Riddle explored all the options with Edinburgh First, the venue's conference team. A few options had to be discounted because of health and safety regulations (there were stairs we might throw ourselves down) but they persisted and kept us all informed. Meanwhile, at the four o'clock coffee break, the Ada community braved the elements for a couple of seconds and made it into the marquee, much to our appreciation and satisfaction. As the coffee break ended, it was decided that Ada developers were in fact made of strong stuff and we would stay put. I'm sure it would be a different story if we we're exhibiting at the Miss World competition. Those who were cold found a spot inside and everyone was happy. I'm sure all the exhibitors would join me in thanking Joan and Tom for keeping everyone that way.

The thought of the whisky tasting session in the evening kept everyone warm as we waited for 6.15pm to approach when we were allowed to take the lids off the bottles and let the aromas drift amongst the crowd. Each of the exhibitors had a bottle of the finest Scotch whisky on their table and stacks of tasting cups. I was most relieved to have Highland Park as our resident whisky. Not because it is a perfectly balanced 18 year old single malt with a toffee sweetness and a mouth-wateringly smokey finish. I was happy because after a whole day and evening exhibiting, I didn't fancy trying to pronounce Aberlour a'bunadh or Ardbeg Uigeadailthe. The whisky inevitably got everyone chatting and moving around the exhibition tasting what was on offer. I snuck off to have a quick look at the foyer area to see how that was going and when I came back a crowd had formed around our table. I thought Ajay must have pulled out the Objektum Bridge presentation and people had swarmed round to see this so called wizardy. As I approached the crowd, I realised that it wasn't the Bridge that was dazzling the audience. It was Tucker Taft, one of the chief designers of the Ada language. It was 8.30pm and we knew we weren't going to compete with Mr. Taft for attention so Ajay and I gathered our belongings and called it a day.

And a successful day at that.

From: Objektum Solutions' blog "The Technical Diaries"
Date: Thu, 23 Jun 2011
Subject: Day 2 Exhibiting at The Ada Connection
URL: http://the-technical-diaries.blogspot.com/2011/06/day-2-exhibiting-at-ada-connection.html

Day 2 for us at Ada Connection and there was a few more new faces, and after the ice-breaker whisky reception the night before, a more relaxed atmosphere.

The talks focused on code generation in the morning and the people we spoke to in the coffee break were keen for the presentations to start.

As people started to filter back into the conference rooms, we had a rare sighting. The sun. Exactly what those in the marquee wanted. In the tropical exhibition area, we made use of the quiet periods by tapping away on our laptops until the sandwiches were laid out on the central table in the marquee and we were all immediately distracted. We ate our share before the masses arrived and were able to talk to people about training and software migration. We were on a roll and gave several presentations on the Objektum Bridge Suite. It's not all about sales though; for us it's also about using these opportunities to find out views and opinions of various technologies and markets. I had a great conversation with a University lecturer from the US about student attitudes to programming mission & saftey critical software and Ada. (This will form an entirely separate blog post!)

Lunch finished and we were on the final straight. The man of many hats, Ahlan Marriot, treasurer of Ada Europe and next year's conference organiser, came to our stand to talk about next year's conference in Stockholm, Sweden. "Would we be interested in being at the conference next year?" I think so I replied. It's been a well organized and well attended conference and we've met some interesting people whom we hope to work with in the future.

Ada is a specialism of Objektum Solutions and so we will continue to support mission and safety critical software development.

The second and last coffee break came and went with more discussions happening around the room. As the last delegate left, all the exhibitors dragged their boxes out from the behind the stands, pulled the plugs out and started to pack up their camps.

Our official duties as exhibitors had ended and after short power naps in front of the Wimbledon coverage, we grabbed our jackets and bags and made our way to the spectacular Signet Library in the centre of town. We turned in to Parliament Square and right on cue, we followed the piper's bellowing tune to the entrance of this grand building.

Enclosed within the columns and walls of law books of the Lower Library, the Ada community delicately sipped on a glass of bubbly (thanks to AdaCore) and craned their necks to take in the splendour of the surroundings. A man with a big wooden hammer (I'm sure he has an official title) did his thing – namely hitting the hammer on another wooden object - to gather people's attention and he grandly called us into the Upper Library where dinner would be served. So, we made our way up the majestic staircase to the breath-taking setting of our banquet. Professor Les Hatton gave us an entertaining pre-dinner speech; one track of which was "Why programmers are monkeys?". I am not a programmer myself but I work with many of them so I wish I had made notes to prove this theory when I return to the office on Friday.

I'll quickly mention the starter, pea and mint soup, but the plate that stole the show was the haggis. Man carrying hammer (aka MC Hammer) did his bit and welcomed the haggis. We clapped the haggis in as it was accompanied by the piper and made its way to the front of the hall (on a tray, not on legs).

Tom Anderson loudly broke into Scottish verse and we thought he'd gone mad.

It unfolded that this was in fact Scottish tradition and Tony Elliston of Ellidiss Software pointed out that there were some "wee drams" of whisky at the front which would go to those who played a part in the Haggis performance that evening; the Piper, The Artist Formally Known as Tom, MC Hammer and the Haggis. The haggis had rave reviews that evening and one critic said of it, "It's the best I've ever tasted".

Main course of sea bass and desserts were laid down by the silver service staff and the speeches started. There was a tiny bit of chocolate torte left on my plate but the speeches were all entertaining enough to distract me from it for a short while. The colourful John Barnes stood on his seat and addressed the dinner guests with tales of Ada Europe from 20 years ago which celebrated a solicitor's help to receive some money which was rightfully theirs. That solicitor, now a sheriff, was tracked down and invited to the dinner and gracefully received a round of applause and some flowers for her efforts all those years ago.

The clock struck 11.00pm and it was time for us to call it a night.

I conclude these posts from bonny Edinburgh, with a big thank you to all those involved with Ada Connection, both organisers and attendees, for making it an enjoyable and successful few days. It's been great and as I am starting to feel a little weary, I wish I had asked the DHL chap who just picked up our stand to deliver me back to leafy Surrey too.

The Ada-Connection — Report by a participant

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: Mon, 25 Jul 2011
Subject: Ada Europe 2011 / The Ada Connection 2011
URL: http://ada-dk.org/?page=news&news_id=343

On 21-23 June I participated in the annual Ada Europe conference.

This year the conference was held in Edinburgh.

The conference was a good opportunity to present one of my recent projects - the Crimeville language server - and to meet other people interested in reliable software technologies.

Some of the new knowledge I brought home from Edinburgh:

- Software professionals would probably be wise to make note of the Hippocratic oath, and consider how it applies to their work. (thanks to Pippa Moore)
- Supposedly there exists such a thing as "too many cores". (thanks to Alan Burns)
- The term "technical debt" and a bit of knowledge about the SQALE quality assessment model. (thanks to Jean-Pierre Rosen)
- The core of agile methods is to continuously hit the customer over the head with an incomplete product until he submits. (thanks to Les Hatton)
- It is not at all easy to do static verification of linked data structures. (thanks to Phil Thornley)
- A nice way to implement mixed criticality real-time systems in pure Ada. (thanks to Alan Burns)
- There exists an ARM chip with built-in 3D accelerometer. (thanks to one of the exhibitors)

Traditional events:

- A new suggestion for an extension to AdaControl: Make it possible to tell AdaControl which packages contain potentially blocking operations.
- Maciej was quick to find a weak point in the implementation I presented in my talk. It is always nice to have an attentive audience.

New reading materials:

- "High-Integrity Object-Oriented Programming in Ada" from AdaCore.

[…]

Ada-Europe 2012 — Preliminary Call for Papers

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Wed, 6 Jul 2011 20:50:33 +0000
Subject: CfP 17th Conf. Reliable Software Technologies, Ada-Europe 2012
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

PRELIMINARY CALL FOR PAPERS

17th International Conference on

Reliable Software Technologies - Ada-Europe 2012

11-15 June 2012, Stockholm, Sweden

http://www.ada-europe.org/conference2012

Organized by Ada-Europe,

in cooperation with ACM SIGAda (approval pending)

*** CfP in HTML/PDF on web site ***

Ada-Europe organizes annual international conferences since the early 80's.

This is the 17th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain ('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04), York, UK ('05), Porto, Portugal ('06), Geneva, Switzerland ('07), Venice, Italy ('08), Brest, France ('09), Valencia, Spain ('10), and Edinburgh, UK ('11).

General Information

The 17th International Conference on Reliable Software Technologies - Ada-Europe 2012 will take place in Stockholm, Sweden. Following its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical

Schedule

28 November 2011: Submission of regular papers, tutorial and workshop proposals

12 January 2012: Submission of industrial presentation proposals

3 February 2012: Notification of acceptance to all authors

2 March 2012: Camera-ready version of regular papers required

11 May 2012: Industrial presentations, tutorial and workshop material required

[…]

Ada and Education

Course of "Ada for experienced programmer"

From: Ed Colbert <colbert@abssw.com>
Date: Fri, 5 Aug 2011 05:30:19 -0700
Subject: [Announcing] Public Ada Courses 12-16 September 2011 in Carlsbad CA
Newsgroups: comp.lang.ada

[…]

Absolute Software will be holding a public Ada course during the week of 22 August 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 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 an e-mail.

Ada-related Resources

The Crimeville Language Server

From: Thomas Løcke <tl@ada-dk.org>
Date: Wed, 15 Jun 2011
Subject: The Crimeville Language Server
URL: http://ada-dk.org/?page=news&news_id=320

Our very own Jacob Sparre will be giving a presentation at The Ada Connection 2011 Conference on the spelling/language server he has developed for the Danish (soon international) childrens' game Crimeville.

For those of us that aren't fortunate enough to be in Edinburgh for the conference, Jacob has made both the presentation (PDF file) and the source code (ZIP file) available on his website.

[http://www.jacob-sparre.dk/spelling/crimeville-talk.pdf

http://www.jacob-sparre.dk/spelling/crimeville.zip —mp]

Here's a blurb from the presentation:

> When Art of Crime contacted me, their problem was simply described as helping the players write correctly, and limit how much they insult each other. – Already at this stage the plan was to do this at the word level. In short, every word written by a player should be categorized in one of four categories; correct, foul, misspelled or unknown. I proposed a solution with network servers checking words using Ispell compatible Open Source spell checkers.

It is an interesting read for sure, and the accompanying source code is just an added benefit for those of us who'd like to learn how to best tame Ada to do our bidding. This obviously includes me, so thanks a bunch Jacob! :o)

AdaTutor is back online

From: Thomas Løcke <tl@ada-dk.org>
Date: Thu, 30 Jun 2011
Subject: AdaTutor is back online
URL: http://ada-dk.org/?page=news&news_id=326

I just got word from Karl Nyberg that the AdaTutor program is back online:

> I have taken over rehosting of www.adatutor.com for John.
>
> It appears to be functional again.

John is of course John Herro, the original author of the AdaTutor program.

Back in April he mentioned that he would be shutting down adatutor.com.

Luckily Karl decided to put it back online.

I haven't tried AdaTutor yet, as it seems to be very Windows oriented, but I do plan on giving it a whirl and see what happens when it is exposed to Slackware Linux and GNAT GPL 2011. :o)

Ada-related Tools

Ada 95 Booch Components 20110612 and 20110622

From: Simon Wright <simon@pushface.org>
Date: Sun, 12 Jun 2011 20:13:21 +0100
Subject: ANN: Booch Components 20110612
Newsgroups: comp.lang.ada

A new release of the Ada 95 Booch Components is now available.

See https://sourceforge.net/projects/booch95/files/booch95/20110612/ for the software, http://booch95.sourceforge.net/release.html for the release notes.

From: Simon Wright <simon@pushface.org>
Date: Wed, 22 Jun 2011 18:27:22 +0100
Subject: ANN: Ada 95 Booch Components 20110622
Newsgroups: comp.lang.ada

Release 20110612 was missing the top-level bc.gpr and support files for Indefinite Maps.

Compilation warnings eliminated.

Matreshka 0.1.1

From: Vadim Godunko <vgodunko@gmail.com>
Date: Sat, 2 Jul 2011 23:51:08 -0700
Subject: Announce: Matreshka 0.1.1
Newsgroups: comp.lang.ada

We are pleased to announce availability of Matreshka 0.1.1. Here is list of most significant improvements of functionality:

- support for calendars and calendrical calculations;
- support for SQL Database Access, including Oracle, PostgreSQL and SQLite;
- support for persistent application settings; INI files and Windows Registry are supported;
- support for ISO-8859-1 and Windows-1251 encoding;
- several extensions of Universal_String and Universal_String_Vector.

Matreshka is set of Ada libraries to simplify development of Ada applications. For more information please visit

http://adaforge.qtada.com/cgi-bin/tracker.fcgi/matreshka/wiki

[see also "Matreshka 0.0.6" in AUJ 32-1 (Mar 2011), p.9 —mp]

JSON Support in GNATcoll

From: Thomas Løcke <tl@ada-dk.org>
Date: Fri, 22 Jul 2011
Subject: JSON Support in GNATcoll
URL: http://ada-dk.org/?page=news&news_id=339

Today I grabbed the latest GNATcoll developer SVN commit [http://libre.adacore.com/libre/tools/gnat-component-collection/ —mp] and with it came a very nice surprise - it appears that AdaCore is adding JSON support to this already wonderful library. Here are the new files in question:

- gnatcoll-json.ads
- gnatcoll-json.adb
- gnatcoll-json-utility.ads
- gnatcoll-json-utility.adb

Looks very nice! To my knowledge, the only other Ada JSON library available, is the jdaughter library by Tero Koskinen.

[http://hg.stronglytyped.org/jdaughter —mp]

I don't know how the two implementations compare to each other, but it sure is nice to have more choices.

Ada support in *BSD distributions

From: John Marino <dragonlace.cla@marino.st>
Date: Fri, 15 Jul 2011
Subject: GNAT-AUX Updated with GCC 4.6.1
URL: http://www.dragonlace.net/posts/GNAT-AUX_Updated_with_GCC_4.6.1/

GCC 4.6.1 was released on June 27th. While not much was done directly to GNAT, 157 bugs were addressed. GNAT AUX was updated accordingly and it's already available to FreeBSD users. Patches have already been submitted to pkgsrc so that DragonFly and NetBSD fans will also receive the latest version of GNAT-AUX soon.[…]

[See also "Ada support in *BSD and Android distributions" in AUJ 32-2 (Jun 2011), p.72 —mp]

Ada for Android

From: John Marino <dragonlace.cla@marino.st>
Date: Tue, 19 Jul 2011
Subject: FreeBSD64 Android cross-compiler builds Tetris with Tasking

URL: http://www.dragonlace.net/posts/FreeBSD64-Android_cross-compiler_builds_Tetris_with_Tasking/

We still don't own a real Android device, so continued work on the Android compiler has been difficult. This may change in the near future as we have our eyes on an Asus Transformer (ARM v7), but until then we are still limited to using the ARM v5 Android SDK emulator.

A new cross-compiler was built, this time using FreeBSD64 as the host machine.

The goal is to convert the cross compiler into a set of ports so that FreeBSD users can obtain the cross compiler like they get other software.

This will be the next project, and once the Android tablet is obtained, the full testsuite will be run on the cross-compiler.

In the meantime, I found a text version of Tetris written in Ada on the AdaPower site. It turns out this game takes advantage of tasking, and that's how we found out the the Android compiler had broken tasking.

Luckily, it only took a couple of hours to figure out how to fix it.

As proof, a screenshot of FreeBSD64-built Ada Tetris running inside an Android SKD emulator hosted on Ubuntu 10.04 LTS Linux is presented, and it works! The fact that tasking runs is a very good indicator that the compiler should do very well when the testsuite is run.

Hopefully we can get these ports built were interested parties can untar them in FreeBSD's /usr/ports directory and get working Android cross-compilers so they can play with it as well. If all goes well, we'll get the ports officially added to the FreeBSD ports tree.

[…]

From: John Marino <dragonlace.cla@marino.st>
Date: Sat, 23 Jul 2011
Subject: GNATDroid cross-compiler ports created for FreeBSD
URL: http://www.dragonlace.net/posts/GNATDroid_cross-compiler_ports_created_for_FreeBSD/

FreeBSD users now have an easy method to obtain their own Ada capable Android cross-compiler. We have created four FreeBSD ports that can build two different GNAT-AUX (gcc 4.6.1) compilers that target Android ARMv5 and Android ARMv7.

The ARMv5 version creates binaries that can be run on the Android SKD emulator. The ARMv7 version is intended to produce binaries that can be run on ARM Cortex-A8+ CPUs, but until we get that Android tablet, we can't confirm it's functional. The emulator can NOT run ARMv7 binaries, so don't attempt it.

The ports will not be submitted to FreeBSD until we can run the full testsuite on them, but in the meantime we're making them available to anyone that wants to use them now (no promises of course!)

The four packages are:

1. lang/gnatdroid-sysroot
2. lang/gnatdroid-binutils
3. lang/gnatdroid-armv5
4. lang/gnatdroid-armv7

The first two are dependencies of the last two which get automatically built, so the process to install a cross-compiler is simple (logged in as root):

1. > cd ~
2. > fetch http://downloads.dragonlace.net/gnatdroid_cross-compiler.tar.bz2
3. > tar -xyf gnatdroid_cross-compiler.tar.bz2 -C /usr
4. > cd /usr/ports/lang/gnatdroid-armv5
5. > make install

If you prefer PGP-signed downloads, the signature file is available, and can be verified against John Marino's public key.

Feedback or comments about these cross-compilers are always welcome.

[See also "Ada support in *BSD and Android distributions" in AUJ 32-2 (Jun 2011), p.72 —mp]

Zip-Ada v.41 (beta)

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 25 Jul 2011
Subject: Support for UTF-8 in Zip-Ada
URL: http://ada-dk.org/?page=news&news_id=340

A few days ago Gautier de Montmollin mentioned that he'd added support for UTF-8 archive entries in version 41 of his Zip-Ada library, and as can be seen from revision 99, he wasn't kidding around.

[http://sourceforge.net/projects/unzip-ada/ —mp]

The changes in version 41 are:

- Support for Unicode (UTF-8) entry names within archives; see: Zip, Zip.Create, Zip_Streams
- Zip_Streams: Made names more consistent, previous names preserved with pragma Obscolescent

There doesn't appear to be an actual version 41 release announcement (yet), so while we're waiting for that, why not visit Gautier's identi.ca stream and subscribe?

[see also "Zip-Ada v.40" in AUJ 32-2 (Jun 2011), p.73 —mp]

Ada-related Products

AdaCore / Altran Praxis — SPARK Pro 10

From: AdaCore Press Center
Date: Tue, 07 Jun 2011
Subject: AdaCore and Altran Praxis Release SPARK Pro 10
URL: http://www.adacore.com/2011/06/07/sparkpro10/

Increased flexibility and functionality for high-assurance systems

NEW YORK and PARIS, June 7, 2011 – AdaCore and Altran Praxis today announced the release of the SPARK Pro 10 software development and verification environment, providing a major step forward for the developers of high-assurance systems. SPARK Pro now offers increased flexibility to developers of systems with mixed integrity levels or with the need to integrate SPARK with other languages or legacy code.

SPARK Pro is a product jointly developed by Altran Praxis, international specialist in embedded and critical systems engineering, and AdaCore, the leading provider of commercial software solutions for the Ada language. SPARK Pro provides the foremost language, toolset and design discipline for engineering high-assurance software. It combines Altran Praxis' acclaimed SPARK language and verification tools, with the GNAT Programming Studio (GPS) and GNATbench Integrated Development Environments from AdaCore. There are SPARK versions based on Ada 83, Ada 95, and Ada 2005, so all standard Ada compilers and tools work out-of-the-box with SPARK.

SPARK Pro is a language and toolset specifically designed for developing applications where correct operation is vital for safety or security. The SPARK Pro toolset offers static verification that is unrivalled in terms of its soundness, low false-alarm rate, depth and efficiency. The toolset generates evidence for correctness that can be used to meet the requirements of safety and security certification schemes such as DO-178B and the Common Criteria.

SPARK Pro 10's new features include:

Automatic selection of flow analysis mode

The SPARK Pro Examiner now supports automatic selection of information flow or data flow analysis on a per-subprogram basis. This new feature increases flexibility for users by making it easier to analyse SPARK programs which have derives annotations (information flow contracts) only on certain subprograms, for example at the lower levels in the call tree or in those areas of the program with the highest integrity level requirements.

This increased flexibility can equally be applied to facilitate the integration of SPARK code with non-SPARK or legacy code (e.g., full Ada or C). It also allows programs to be developed initially without derives annotations, and to have derives annotations added at a later stage as necessary.

KCG Language Profile

As part of a collaborative development with Esterel Technologies, a new language profile has been added to the Examiner for processing automatically-generated SPARK code produced by Esterel's KCG code generator for SCADE. The SPARK/Ada version of KCG for SCADE will be available in Q4 of 2011 and further releases of both the SCADE and SPARK Pro toolsets in 2011 and 2012 will provide users with a route to static verification of automatically-generated SPARK code. The integration of these technologies

will afford users the benefits of model-driven development with the assurance of a secure programming language and associated verification tool suite.

Derived Numeric Types

Definition of numeric types has been made easier by the introduction of language and tool support for explicitly derived numeric types. This removes the need for a user-supplied base type assertion and removes the risk of the user indicating a base type that is inconsistent with the target.

SPARKBridge preview for Windows

SPARKBridge – a bridge between the SPARK tools and Satisfiable Modulo Theories (SMT) solvers – was initially introduced as a GNU/Linux-only preview in SPARK Pro 9.1. SPARK Pro 10 extends this preview to Windows users, allowing them to experiment with alternate provers for discharging Verification Conditions. A fully-supported version of SPARKBridge will be available in future releases of the Black Belt edition of SPARK Pro.

Library Additions

The SPARK library has been augmented with several new packages including Interfaces, Ada.Characters.Handling, and Ada.Text_IO.

Proof Tools

A number of improvements have been made to the SPARK Pro proof tools.

The Simplifier now has enhanced reasoning capabilities for modular types, allowing more proofs to be automatically discharged. In addition, the proof summary output (from the POGS tool) has been improved to make the management of the proof process easier for large projects.

Availability

SPARK Pro 10 is available now. For more information please visit

http://www.adacore.com/home/products/sparkpro/ or contact info@adacore.com.

Webinar

A webinar providing an introduction to the new features in SPARK Pro 10 will be presented on the 5th July. For more information and to register please visit www.adacore.com/home/products/sparkpro/language_toolsuite/webinars/

About Altran Praxis

Altran Praxis is a specialist systems and software house, focused on the engineering of systems with demanding safety, security or innovation requirements.

Altran Praxis leads the world in specific areas of advanced systems engineering and innovation such as: ultra low defect software engineering, Human Machine Interface (HMI), safety engineering for complex or novel systems, systems engineering and methods/tools (such as SPARK). Altran Praxis offers clients a range of services including turnkey systems development, consultancy, training and R&D. Key market sectors are aerospace and defence, rail, nuclear, air traffic management, automotive, medical and security.

The company operates globally with active projects in the US, Asia and Europe.

The headquarters of Altran Praxis are in Bath (UK) with offices in Sophia Antipolis, London, Paris, Loughborough and Bangalore.

Altran Praxis is an expertise centre within the Altran Group (altran.com) – a global leader in innovation engineering, employing over 17,000 staff across the world. www.altran-praxis.com

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial software solutions for Ada, a state-of-the-art programming language designed for large, long-lived applications where safety, security, and reliability are critical. AdaCore's flagship product is the GNAT Pro development environment, which comes with expert on-line support and is available on more platforms than any other Ada technology. AdaCore has an extensive world-wide customer base; see http://www.adacore.com/home/company/customers/ for further information.

Ada and GNAT Pro see a growing usage in high-integrity and safety-certified applications, including commercial aircraft avionics, military systems, air traffic management/control, railway systems and medical devices, and in security-sensitive domains such as financial services.

AdaCore has North American headquarters in New York and European headquarters in Paris.

www.adacore.com

AdaCore — GNAT GPL 2011

From: Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be>
Date: Wed, 15 Jun 2011
Subject: GNAT GPL 2011 available
Mailing list: ada-belgium-info.cs.kuleuven.be

Dear Ada-Belgium friend,

We hereby forward you an announcement from our long time corporate member and sponsor AdaCore about the new GNAT GPL 2011 release.

[…]

----- Forwarded message -----

Dear GNAT GPL user,

We are pleased to announce the release of GNAT GPL 2011, the integrated Ada, C, and C++ toolset for Academic users and FLOSS developers.

This new edition provides many new features and enhancements in all areas of the technology. The most notable ones are:

- improved support for Ada 2012
- enhanced versions of tools
  - o GPS 5.0 enhanced IDE (improved support for C/C++, more powerful source editing, better usability, ?.),
  - o GtkAda (new widgets, interface to the Cairo graphics library)
- more flexible and more efficient project manager tool
- support for unloading Ada plug-ins
- improved support for Ada constructs on the .NET platform
- more detailed exception messages (-gnateE switch)
- complete support for Lego MINDSTORMS hardware, including audio and I2C sensors

GNAT GPL 2011 comes with version 5.0.1 of the GNAT Programming Studio IDE and GNATbench 2.5.1, the GNAT plug-in for Eclipse.

GNAT GPL 2011 can be downloaded from the "Download" section on

https://libre.adacore.com.

AdaCore / Altran Praxis — SPARK GPL 2011 and SPARKSkein 2011

From: Rod Chapman <roderick.chapman@googlemail.com>
Date: Wed, 20 Jul 2011 05:39:23 -0700

Subject: SPARK GPL 2011 and SPARKSkein 2011
Newsgroups: comp.lang.ada

SPARK GPL 2011 is now up on libre.adacore.com.

We've also updated the SPARKSkein release to meet v1.3 of the Skein specification, and reproduced all analyses and proofs with the GPL 2011 toolset.

This is at www.skein-hash.info.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 21 Jul 2011 04:42:04 -0400
Subject: Re: SPARK GPL 2011 and SPARKSkein 2011
Newsgroups: comp.lang.ada

[…]

Very interesting!

However, I think www.skein-hash.info could use a bit more propaganda about what SPARKskein is, and why people should care about it.

For example, here is the abstract from the paper about SPARKskein:

(http://www.skein-hash.info/sites/default/files/SPARKSkein.pdf)

"This paper describes SPARKSkein – a new reference implementation of the Skein algorithm, written and verified using the SPARK language and toolset. This paper is aimed at readers familiar with the Skein algorithm and its existing reference implementation, but who might not be familiar with SPARK. The new implementation is readable, completely portable to a wide-variety of machines of differing word-sizes and endian-ness, and "formal" in that it is subject to a proof of type safety. This proof also identified a subtle bug in the implementation which persists in the C version of the code. The new code offers similar performance to the existing reference implementation. As a further result of this work, we have identified several opportunities to improve both the SPARK tools and GCC."

Just getting this on the web page somewhere (perhaps on http://www.skein-hash.info/downloads) would be very good. Especially the parts about being completely portable (I'm assuming the C version is not?), finding a bug in the C version, and "similar performance".

This is an excellent opportunity to advertise the benefits of Ada and SPARK.

Hmm. That info is available at

http://www.altran-praxis.com/news/SPARKskein_16_Aug_10.aspx, which is probably a more appropriate place for such things. Pardon my enthusiasm :)

[…]

Ada and CORBA

GNACK — GNU Ada CORBA Kit 1.3

From: Oliver M. Kellogg <okellogg@users.sourceforge.net>
Date: Sat, 16 Jul 2011 18:36:50 -0700
Subject: Ada bindings for the GNOME ORBit CORBA ORB
Newsgroups: comp.lang.ada

Version 1.3 of the GNU Ada CORBA Kit (GNACK) has been released.

The main feature for this version is improved stability for server implementations.

For further info, see the NEWS file included in gnack-1.3.tar.gz.

http://sourceforge.net/projects/orbitada

[See also "GNACK — GNU Ada CORBA Kit" in AUJ 28-1 (Mar 2007), p.17. —mp]

Ada and GNU/Linux

Support for Ada in Fedora Linux

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 27 Jun 2011
Subject: Ada in Fedora
URL: http://ada-dk.org/?page=news&news_id=323

For quite some time now, Ada has been a primary citizen in Debian Linux with lots of packages available by simple typing the familiar apt-get command.

Now it seems as if Fedora users will get the same treatment.

[http://fedoraproject.org/wiki/Packaging:Ada —mp]

I don't know if someone has stepped up to the plate yet, but simply having a policy for how to build Ada packages for Fedora is a good start.

Maybe I should consider doing the same for Slackware? I could write a few slackbuild scripts and submit them to slackbuilds.org.

Hmm, it's worth considering at least.

Debian Ada VM

From: R. Tyler Croy <tyler@linux.com>
Date: 03 Aug 2011 04:27:06 GMT
Subject: Debian Ada VM
Newsgroups: comp.lang.ada

I've been tinkering with Vagrant (http://vagrantup.com) and Puppet lately and I've created a simple set of manifests for automatically provisioning a Debian Ada VM:

https://github.com/rtyler/debian-ada-vm

It's pretty bare-bones right now since I'm not doing anything tricky with that. If you're interested in extending it and adding repositories from GitHub (ockham, adbci, etc) or something like that, it should be pretty easy.

Happy hacking.

References to Publications

High-integrity object-oriented programming with Ada

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 08 Aug 2011 13:52:28 -0700
Subject: High-integrity object-oriented programming with Ada
URL: http://ada-dk.org/?page=news&news_id=346

Written by Benjamin Brosgol from AdaCore, this article series dig deep into the bowels of Ada and its OOP features, specifically pertaining to high-integrity systems. It is a very interesting read, with lots of exact information. Clearly Benjamin is an experienced and skilled engineer.

It's a three part article, of which two parts are currently available.

Part 1 of this three-part article reviews the basics of object-oriented programming and summarizes the challenges it presents for high-integrity programming. Part 2 will provide a primer on the Ada programming language, and Part 3 will detail the tools Ada offers to help developers meet the OOP challenges.

If you are in any way interested in Ada and OOP, then these articles can serve as a good place to start.

- High-integrity object-oriented programming with Ada - Part 1

  [http://www.eetimes.com/design/military-aerospace-design/4218039/High-Integrity-Object-Oriented-Programming-with-Ada —mp]

- High-integrity object-oriented programming with Ada - Part 2

  [http://www.eetimes.com/design/military-aerospace-design/4218257/High-integrity-object-oriented-programming-with-Ada---Part-2 —mp]

I will of course link to part 3 of the article series, as soon as it's made available.

Ada Inside

Ada used in railway control and information systems by Siemens

From: AdaCore Press Center

Date: Thu, 30 June 2011
Subject: Siemens Switzerland Selects AdaCore Toolset for Railway Project
URL: http://www.adacore.com/2011/06/30/siemens-railway/

GNAT Pro to be used for safety-critical software development of railway control system

NEW YORK, PARIS, ZURICH, June 30, 2011 – 16th International Conference on Reliable Software Technologies AdaCore, a leading supplier of Ada development tools and support services, today announced that the Mobility Division of Siemens Switzerland Ltd., has selected GNAT Pro, along with the CodePeer static analysis tool, to develop the next generation of its railway control and information system. The contract with AdaCore provides Siemens software developers with state-of-the-art Ada tools and direct access to the world's largest team of Ada experts, many of whom have years of experience in safety-critical application development.

The Siemens railway control system is a modern networked application that covers every aspect of the railway control domain. It uses a distributed architecture to allow a computer to automatically take over control of a cell from another computer in the same cell due to a hardware failure or planned maintenance. This architecture guarantees high-availability of the system in accordance with European railway software standards. The current version of the system controls the train traffic throughout major parts of Switzerland and also parts of Austria, Hungary and Malaysia.

"Safety has the highest priority in the railway business. Therefore, we invest a lot of time and energy in code-review and testing activities.

Recently, Siemens is experiencing a renaissance in demand for its railway control software, which is placing a heavy load on our software development resources. In order to meet the demand, without compromising safety or quality, we recognized the need for tools that would allow us to work more efficiently. Our two most important requirements were an Ada compiler that could be configured to analyze code against a rigorous set of specific criteria, and an automated code review and validation tool to identify potential runtime errors. Now, with detailed feedback from the GNAT compiler and CodePeer we are able to discover problems at the source instead of in the test lab and the code-review process is now essentially automated," said Daniel Bigelow, Siemens software developer.

[…]

Indirect Information on Ada Usage

[Extracts from and translations of job-ads and other postings illustrating Ada usage around the world. —mp]

Job offer [France]: Ada Software Engineer

[…] You will report to the technical lead of the embedded software division.

You will be involved in a project developed in partnership with our main clients of the railway industry. During your assignment, you will develop software for the on-board and ground segment of the CBTC [Communications-Based Train Control —mp] safety system.

You will mainly develop new functionalities for the CBTC (analysis, design specification, coding and testing) and correct bugs.

Additionally, you will support the project team in the integration and validation of the system.

Profile:

You hold a degree in industrial informatics and have previous experience in the development of embedded software for the railway, defence or avionics domains.

You have a good knowledge of the Ada programming language and of the EN50128 standard.

Knowledge of the CBTC system is an asset.

[…]

[Translated from French —mp]

Job offer [Italy]: Software Engineer

We are looking for a software engineer.

You hold a master's degree in electronic, aerospace or computer engineering and have at least 2/3 years of experience in the development and testing of real-time embedded software.

The tasks for this job position include:

- Detailed design with UML from software requirements.
- Development of source code in Ada 95.
- Definition of the testing procedures for functional and software verification
- Testing and analysis of results.

You shall also fulfill the following requirements:

- Knowledge of Ada 95 and of an associated compilation and debugging toolchain (preferably AdaMULTI)
- Knowledge of design tools based on UML (preferably Real-Time Studio)
- Knowledge of Matlab and Simulink
- Knowledge of Visual Basic
- Knowledge of a configuration tool (preferably PVCS)
- Excellent proficiency in written and spoken English

Knowledge of the software life-cycle and related standards (e.g. RTCA/DO178B) is an asset.

[Translated from Italian —mp]

Job offer [Spain]: Senior Software Engineer

- Bachelor's or master's degree

[…]

- Experience in the management of small projects or work packages of big projects.
- Experience in software development life cycle
- Experience in Ada, C, C++. Knowledge of Polyspace and C# is an advantage.
- Knowledge of the complete life cycle of a software project

Experience 3-5 years

We are looking for […] 2 experts in real-time systems with experience in avionics and/or railway signalling who will work in the management area and want to assume technical leadership.

[Translated from Spanish —mp]

Job offer [Spain]: Embedded Software Engineer

[…]

Education: A degree in Computer, Electronic or Telecommunications engineering.

Project domain: Automation, Consumer Electronics.

Experience: 2 years

We are collaborating in R&D projects for the industrialization of electronic products for automation and consumer and professional electronics.

In the automotive domain, we are working with first-tier integrators and suppliers.

We are looking for engineers with experience in the development of real-time software (specification, architecture, coding, unit testing, integration testing, validation), with good knowledge of one or more programming languages (C, C++, ADA [sic —mp]), microcontrollers, real-time operating systems (VxWorks, Nucleus, pSOS, MCarol etc.), tools for configuration management […], and communication busses […].

Requirements : Experience of at least 2 years as embedded software engineer. Technical knowledge: RTOS (pSOS, VxWorks, MCarol…), microcontrollers (Hitachi, ST, Intel, Motorola…), programming languages (C, C++, TCL, VB, ADA…), communication busses (CAN, Most, I2C), tool for configuration

management (Clearcase, Continuus, PVCS)…

[Translated from Spanish —mp]

Job offer [United Kingdom]: Lead Software Engineer

Lead Software Engineer, Aerospace - C, C++, Ada

You will be responsible for defining technical concepts and top level implementation solutions, providing technical oversight for services contracted for outside providers and design, implement, program and test software ensuring applicable processes are followed and quality specifications are met.

Ideally, the successful candidate will be an experienced Software Engineer with previous experience of working in a customer-facing position.

Experience of programming in the following languages is highly desirable:

C, C++, Ada and Assembler.

[…]

Ada in Context

Best practices for the "use" clause

From: Arnauld Michelizza <a.michelizza@gmail.com>
Date: Tue, 7 Jun 2011 08:32:25 -0700
Subject: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

Hi guys,

Sorry with my annoying questions, but it's not easy to program in Ada after 20 years programming in asm / C ;-)

Maybe a not so anedoctical question : when using the 'use' clause ?

My first thought is that using 'use' is not good because it masks the package tree.

[…]

But always avoiding 'use' give some rather obfuscated code, especially when using some specific arithmetic operator. For example, playing with Address arithmetic without 'use' :

```
with System;
with System.Storage_Elements;
procedure Main is
  A : System.Address;
  N : Integer;
begin
  A := System.Storage_Elements.
                To_Address(16#10000#);
  N := 4;
  System.Storage_Elements.
        "+"(System.Storage_Elements.
             Storage_Offset(N),A);
end Main;
```

The same code with 'use' is more readable […]

From: Simon Wright <simon@pushface.org>
Date: Tue, 07 Jun 2011 16:59:05 +0100
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> My first thought is that using 'use' is not good because it masks the package tree.

100% agree. Well, actually more like 90%! I have no problem writing

```
with Ada.Text_IO; use Ada.Text_IO;
```

(but, I'd most often be using Text_IO for debug trace output, not in operational code).

And some packages (Ada.Strings.Unbounded) are designed to be "use"d.

AdaCore tend to use package renaming, e.g.

```
package SSE renames
   System.Storage_Elements;
```

[…]

> But always avoiding 'use' give some rather obfuscated code, especially when using some specific arithmetic operator. […]

See "use type", designed for exactly this purpose.

http://www.adaic.org/resources/add_content/standards/05rm/html/RM-8-4.html#I3407

Although you can "use type" in the context clauses (the "with"s) I much prefer to put them as close as possible to the actual use: e.g., in the declarative region of a subprogram.

Of course, if all the subprograms in a package need to "use type", move the "use type" to package scope.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 7 Jun 2011 18:22:46 +0200
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> Maybe a not so anedoctical question : when using the 'use' clause ?

Always, and packages has to be designed to be "use-friendly."

[…]

In general, the idea of fully qualified names is incompatible with generic programming, i.e. when some operations are defined on a class of types with different bodies for different members of the class.

P.S. All Ada users are subdivided into use-haters and with-haters. The former are in majority.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 7 Jun 2011 19:25:29 +0200
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> use-haters and use-lovers? Hating "with" would match pretty well with hating Ada, I'd have thought!

In

```
with P; use P;
```

the "with P" is meaningless here.

"With" does not carry any useful information for the reader. You have to manage sets of "with" as you develop your packages. I bet even most stubborn use-haters do not take the idea of "with" seriously. Because otherwise they would have to mention *all* implicitly or explicitly referenced packages in "with" clauses. This mammoth task would go to waste. Because the information that brings is null, and interesting only to the compiler-linker. And if you are in favor of dotted names longer than the source line, why would you need an additional "with" if the source is already full of package names?

BTW, there is another dichotomy: child packages-haters vs. with-haters.

Package dependencies introduced by children-parent relation are more evident and less arbitrary than ones by "with." The only problem is that multiple parents are not allowed.

From: Simon Wright <simon@pushface.org>
Date: Tue, 07 Jun 2011 18:29:48 +0100
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> Because otherwise they would have to mention *all* implicitly or explicitly referenced packages in "with" clauses. This mammoth task would go to waste.

Perhaps that's why the language designers didn't go that way.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 7 Jun 2011 21:38:32 +0200
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

[…]

It was different in the 80s, projects were much smaller. There were almost no reusable components. You could really gasp all relationships between your packages. The structure of packages was very rigid, designed up front.

In these days this is just unrealistic, so the coding style must adapt and the language should provide some support. I would prefer stricter rules on declarations hiding each other (requiring explicit resolution of all conflicts) [*]. I definitely want a

stronger than "use", transitive clause to incorporate the declaration scope of a package into the scope of another package. I would like to have multiple parents too.

* When people play this card against MI, they forget that packages allow exactly same. And if fully qualified names were a solution for packages, why it could not be for MI as well?

From: Shark8 <onewingedshark@gmail.com>
Date: Tue, 7 Jun 2011 16:38:53 -0700
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> […] My first thought is that using 'use' is not good because it masks the package tree.

While this is true it can be handy for reducing prefix-clutter, as could a rename. So, while you may be hesitant to use them on the package, I find using them in declare-blocks and subprograms to be quite readable and usable.

[…]

From: Pascal Obry <pascal@obry.net>
Date: Tue, 07 Jun 2011 19:33:29 +0200
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> But always avoiding 'use' give some rather obfuscated code, especially when using some specific arithmetic operator.[…]

I don't like use so when *I* design a package I prefer to take a convention where use is not necessary. But when you want to use an already built API you must often use it the way it has been designed to be used. Take for example Unbounded_String, nobody will avoid using use in this case… Look at this code:

```
with Ada.Strings.Unbounded;
procedure Whatever is
   S : Ada.Strings.Unbounded.
                     Unbounded_String;
begin
   Ada.Strings.Unbounded.Append (
                     S, "toto");
end Whatever;
```

Far better with:

```
with Ada.Strings.Unbounded;
procedure Whatever is
   use Ada.Strings.Unbounded;
   S : Unbounded_String;
begin
   Append (S, "toto");
end Whatever;
```

Because Ada.Strings.Unbounded (and all Ada runtime packages) has been designed to be used with a use clause.

Now I do prefer:

```
package Circle is
   type Object is…
```

And use it as:

```
with Circle;
O : Circle.Object;
```

To

```
package Circle is
   type Circle_Type is…
```

And use it as:

```
with Circle; use Circle;
O : Circle_Type;
```

But all styles are around…

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 7 Jun 2011 21:42:10 -0500
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> […] Take for example Unbounded_String, nobody will avoid using use in this case... Look at this code: […]

Looks good to me, it's what I would write.

> Far better with: […]

Maybe, if that is the entire package. But that's not usually the case, and typically the use of Unbounded_Strings is a very small fraction of the code.

Moreover, if you also have fixed strings floating around (which I usually do), and some uses of Ada.Strings.Fixed as well (which also are pretty likely), use clauses are simply not going to work. The typical expression cannot be understood by the compiler, and trying to figure out why will be impossible.

I sometimes rename the horribly named "To_Unbounded_String" to "+" (this operation needs to be short), and occasionally will use a "use clause" in a subprogram scope. But that's about it.

> Because Ada.Strings.Unbounded (and all Ada runtime packages) has been designed to be used with a use clause.

Right, and it is unfortunate.

With Claw, we tried to split the difference; we used short subprogram names and longer type names. The "Object" trick was too weird for me at the time (and I still don't like it much).

From: Jeffrey Carter <jrcarter@acm.org>
Date: Tue, 07 Jun 2011 11:51:43 -0700
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

[…]

This is something of a religious topic; people have strong opinions one way or the other. "Best" is difficult to define objectively.

However, it's interesting to note that use clauses are not allowed in SPARK. If correctness of your SW is important enough to use SPARK, you won't have use clauses.

From: Peter C. Chapin <PChapin@vtc.vsc.edu>
Date: Tue, 07 Jun 2011 19:04:46 -0500
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

[…]

It should be noted that SPARK does allow a (limited) form of 'use type' precisely so that operators can be made directly visible.

From: Stefan Lucks <stefan.lucks@uni-weimar.de>
Date: Wed, 8 Jun 2011 08:06:58 +0200
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

[…]

There are alternatives to the use in the syntax clause:

-> use type;

-> a renames clause

-> a local use

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 12 Jun 2011 03:55:12 -0400
Subject: Re: using `use' : what is the best practice ?
Newsgroups: comp.lang.ada

> […] So, while you may be hesitant to use them on the package, I find using them in declare-blocks and subprograms to be quite readable and usable.

This is my policy as well; 'use' clauses are forbidden in package specs, and must be localized as much as possible in package bodies.

On the suppression of run-time checks

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Fri, 17 Jun 2011 11:42:45 +0200
Subject: Runtime check : what about you ?
Newsgroups: comp.lang.ada

Hello, just out of curiosity as much as because this may be worth to discuss […]: how many of you typically compile releases with run-time check and how many of you typically compile releases without runtime check ?

[…]

Feel free to add any specific context information with your reply (I guess most of you will feel the need)

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Fri, 17 Jun 2011 12:56:06 +0300

Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] I release with run-time checks (-gnato -fstack-check, although I must sometimes omit the latter due to compile-time problems). This is a non-interactive, non-embedded, non-real-time application for PCs.

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Fri, 17 Jun 2011 03:06:21 -0700
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] With checks on since 1995 - the real difference (for me) was the move from 68000's to PowerPC - the processor was fast enough to allow them […] Unless the safety of the system required no run-time exceptions, of course! […]

From: Simon Wright <simon@pushface.org>
Date: Fri, 17 Jun 2011 11:43:59 +0100
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] The mission-critical system I used to work on was released with run-time checks enabled. We would have used stack checking, but there was a problem with the old GNAT release we were working with (I forget now exactly what).

It was considered better to halt one of the redundant systems, and fail over to the other, rather than proceed with the system in an unexpected and probably unstable state.

From: J-P. Rosen <rosen@adalog.fr>
Date: Fri, 17 Jun 2011 13:35:23 +0200
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] AdaControl is always released with all checks on - and a number of internal additional checks.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 17 Jun 2011 09:06:47 -0700
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] On all systems and platforms - Windows/aix

We use gnato and fstack-check

On some systems we went from -O0 to -O2. Disaster on both platforms.

We also use -g This is with GNAT Pro.

The trouble with -O2 is probably fixed, this was several years ago.

Warehouse management/control system.

The penalty is overweighted by ease of find reasons for crashes (if we get them)

From: Jeffrey Carter <jrcarter@acm.org>
Date: Fri, 17 Jun 2011 10:53:56 -0700
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] The description of the language in the ARM includes run-time checks. If checks are turned off, then you're using some other language, not Ada.

We always have checks on for our soft-real-time application.

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 17 Jun 2011 11:59:37 -0700
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] Then I guess the RM sections on the Suppress pragma must be a big fat misprint. Obviously, they're rogue pages that sneaked into the RM from the standard for some other language.

Really, I don't see the point of statements like that. Ada is a tool, to be used for practical purposes. It's not a religion. And it was certainly part of the intent of Ada's designers that developers would develop their programs with checking turned on but then turn it off after the program has been tested and is ready to be put into production. It's interesting to me that no one here has admitted doing this; I don't know what this means, except that perhaps they're only developing programs for which the computation time is small to the amount of time spent waiting for the user to figure out where to move the cursor, or something. Or that no one is developing programs that require a long intensive algorithm on a large 2-D array or something like that, for which turning off checking could easily make a huge difference.

From: J-P. Rosen <rosen@adalog.fr>
Date: Sat, 18 Jun 2011 09:16:50 +0200
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

> And it was certainly part of the intent of Ada's designers that developers would develop their programs with checking turned on but then turn it off after the program has been tested and is ready to be put into production. […]

Definitely not. Don't forget that a pragma, including pragma suppress, can be put in very limited scopes.

The intent is that IF you have identified the innermost loop that eats up 90% of the computing time, and IF you have determined by careful measurement that a significant part of it is taken by checks, and IF your program cannot meet otherwise the performances from its requirements, THEN you can include this loop into a block statement to which a pragma suppress applies.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 17 Jun 2011 19:26:29 -0500
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

> […] It's interesting to me that no one here has admitted doing this;

I've done it, but not much recently. Even computationally intensive programs like my Solitare solver only needed it in a couple of very limited locations. And in the most recent such cases, I restructured the code (and admittedly, made the compiler smarter) so that the checks aren't generated in the first place -- which is of course the best of both worlds -- fast code which the compiler has proved to have no check failures.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 17 Jun 2011 19:15:39 -0500
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…]

Janus/Ada is released with checking off. That was because the compiler with checking on was too large for typical machines back in the day, and there are various reasons that it is best to keep this the same going forward. OTOH, all of the beta releases of Janus/Ada are with checking all.

All of RRS's other programs and my other programs are released/used with checking on. Modern Ada compilers do a very good job of removing extra checks, and it is very rare that I have seen a case where it is worth the effort to suppress them. For things like the AdaIC search engine, it's many times better to have the protection of the checks in case there is some bug (out-of-range, null pointer deref, etc.) in the code -- with checking on, such bugs have no effect than causing a denial-of-service to the caller; with checking off, who knows what could happen?

I personally believe in the seatbelt analogy: "turning off checks in released software is like using seatbelts in the driveway and then taking them off when you reach the highway". For me, this also applies to assertions and contracts as well -- I only turn these things off if they are tremendously expensive (in which case I usually remove them permanently). I know there are others (like Bob Duff) who think this analogy is silly.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Fri, 17 Jun 2011 20:29:29 -0400
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

No, I don't think it's silly. I think it applies in some cases, but not others. I think turning checks on or off is a difficult engineering decision that should depend on various factors.

From: Björn Lundin <b.f.lundin@gmail.com>
Date: Fri, 17 Jun 2011 13:30:05 -0700
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] In my case, we are spending most of the time on either database I/O or communication I/O.

From: Tom Moran <tmoran@acm.org>
Date: Sat, 18 Jun 2011 03:08:56 +0000
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] I've never in real life seen a difference that could be called "huge".

Although I grant that for some things, like a 2-D FFT on many images, I call an optimized asm library routine (which has no checks).

My current code runs a small PEG TV station, where nobody dies if the time-and-temperature slide misses an update cycle, or a DVD is unreadable, etc. All Ada checks are on, and the handler logs, sends an email, or sends a text message if there's a Minor .. Significant unforeseen problem.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Sat, 18 Jun 2011 11:04:09 +0300
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] We effectively did that in my preceding Ada project (the platform on-board SW for the GOCE satellite) where we tested on a workstation using native compilation with checks on, but released cross-compiled target code with checks off. The target compiler does not support standard exception handling so we did not even have to think about whether and how we could have handled check failures on the target. (We did of course run the tests on the target, too, not just on the workstation.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 18 Jun 2011 08:56:41 +0200
Subject: Re: Runtime check : what about you ?
Newsgroups: comp.lang.ada

[…] We leave most checks on (stack, integer overflow) in the release versions. […] The platforms are VxWorks and Windows, used for distributed automation and control systems (many sensors, actuators, protocols etc). When performance question arise, which happens due to very tight requirements on the latencies we have, we try to optimize the software, so that the compiler would remove unnecessary checks.

On task components and finalization

From: Pablo Rego <pvrego@gmail.com>
Date: Mon, 11 Jul 2011 20:24:04 -0700
Subject: Task origin track from a class
Newsgroups: comp.lang.ada

I have a class Def_Class which defines a record which is a task. Say:

```
task type My_Task_Kind;

type Def_Class is tagged limited
  record
    Some_Element : Integer; -- or other
                            --type anyway
    My_Task : My_Task_Kind;
  end record;
```

and I want to access my class inside the task body, something like

```
type body My_Task_Kind (
              Origin: Def_Class) is
begin
  if Origin.Some_Element = 1 then
                             (…)
  end if;
end My_Task_Kind;
```

So how can I do it? (I tried to use an entry type, but got problems with limited/non limited types, so asking for help!!)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 12 Jul 2011 09:41:24 +0200
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…]

> I have a class Def_Class which defines a record which is a task.

You shouldn't do that, because most likely you will later have serious (unsolvable) problems with the object's finalization.

Unless the task has somewhere a select with an open terminate alternative, the object's finalization will hang.

Note that deriving it from Ada.Finalization.Limited_Controlled won't help.

Consider this:

```
type Parent;
task type Worker (
      Data : not null access
             Parent'Class) is
  entry Do_Stuff;
  entry Stop;
end Worker;
type Parent is
  new Ada.Finalization.
             Limited_Controlled with
record
  My_Task: Worker(Parent'Access);
end record;
overriding procedure Finalize
      (Object : in out Parent);
```

and the implementation:

```
procedure Finalize (
      Object : in out Parent) is
begin
  Object.My_Task.Stop; -- Kill the
                       -- task
end Finalize;

task body Worker is
begin
  loop
    select
      accept Do_Stuff;
       -- Some useful stuff to do
    or accept Stop;
      exit;
    end select;
  end loop;
end Worker;
```

This does *not* work! The problem is that the task must complete *before* Finalize of the containing object is called. It would work if the select of the task would have:

```
or terminate;  -- Complete if asked
```

(and then the Stop entry call removed from Finalize, of course).

The problem with this is that too frequently there is no way to add the terminate alternative (for various reasons, which are irrelevant here).

So the unfortunate rule of the thumb is: never use task components, but access to task instead. From Finalize you would call Stop entry or an equivalent and then free the task object using Unchecked_Deallocation.

From: Simon Wright <simon@pushface.org>
Date: Tue, 12 Jul 2011 11:29:59 +0100
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…]

With GNAT, best not to free the task object until 'Terminated is True (GNATs I have used would silently fail to actually free the TCB! [Task Control Block —mp] resulting in an insidious memory leak).

From: Simon Wright <simon@pushface.org>
Date: Tue, 12 Jul 2011 11:31:45 +0100
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…] Oops, I forgot to add that I'd aborted the task first (as Dmitry said, it's not always possible to arrange a clean shutdown).

From: Shark8 <onewingedshark@gmail.com>
Date: Tue, 12 Jul 2011 17:36:29 -0700

Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…]

Hm, would it be a usable idea for say the memory manager for an OS, such that the requests to the manager from programs (and perhaps even compilers) are delegated to the task; after all you don't want to shut the memory-manager down (terminate the task) at any point in normal operation.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 13 Jul 2011 09:41:11 +0200
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

The issue has nothing to do with memory management. The actual problems are:

1. The procedure of destruction of the objects having task components. Tasks are completed *before* Finalize is called. Ada's approach to construction/ destruction is very much broken. This is just one example of this.

2. The usage of the terminate alternative which cannot be mixed with for example the delay alternative.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Wed, 13 Jul 2011 10:43:15 +0200
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

> 1. The procedure of destruction of the objects having task components. […]

When the life time of objects is determined by Ada's language rules (and not by the type system), isn't it normal to expect that the language defined wrecking ball smashes the thing only after it has finished?

What would be the alternative? Would it be that the programmer then has to actively manage all parts of destruction himself?

When an implementation collects garbage, when would the "destructor" run? Do finalization and RAII [Resource Acquisition Is Initialization —mp] destruction(?) have to be separate things?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 13 Jul 2011 13:59:13 +0200
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…]

Finalize is expected to have the object fully operational if called for the first time. Which is not the case for the task components.

> What would be the alternative? Would it be that the programmer then has to actively manage all parts of destruction himself?

Not at all. The alternative is for the user-defined constructor/destructor's hook to meet all components valid when called.

Finalize is not such a hook.

> When an implementation collects garbage, when would the "destructor" run?

The destructor of the type T to be called before the memory allocated for the object loses its attribution to the type T. Note that this does not imply deallocation. For instance when S is demoted to its base type T, the destructor of S must be called to make a T out of S.

> Do finalization and RAII destruction(?) have to be separate things?

Finalization is another word for destruction. (I am not a language lawyer and don't known what the RM might say about it)

Again, observe that promotion requires a partial construction while demotion does a partial destruction. The primitive operation Finalize is a partial destructor. A complete destruction includes the bodies of the Finalize from all bases and destructors of all components.

Note also that a consistent model must also provide hooks for promotion/demotion to the class (e.g. S -> T'Class, T'Class -> S).

From: Shark8 <onewingedshark@gmail.com>
Date: Wed, 13 Jul 2011 09:31:47 -0700
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…]

> The issue has nothing to do with memory management.

You misunderstand me; I was asking that if it's generally impossible to destroy such a construct then would the use of such a construct be allowable (or desirable) in the cases where the destruction of such construct is itself generally undesirable/invalid.

> The actual problems are:

> 1. The procedure of destruction of the objects having task components. Tasks are completed *before* Finalize is called.

Shouldn't they be completed before Finalization of the object?

And, if they are still running, shouldn't a finalize force termination?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 13 Jul 2011 19:22:50 +0200
Subject: Re: Task origin track from a class
Newsgroups: comp.lang.ada

[…] It is possible, the requirement, as I said, is an open terminate alternative.

[…]

> Shouldn't they be completed before Finalization of the object?

Yes, the question is at which part of finalization. Finalization is a complex action.

BTW, it is not just Finalize. If you considered to pass some parameters to a task component, you cannot do it from Initialize. The following does *not* work:

```
task type Worker is
   entry Start (Text : String);
end Worker;
type T is
   new Ada.Finalization.
            Limited_Controlled with
record
   My_Task : Worker;
end record;
overriding procedure Initialize
                  (Object : in out T);

task body Worker is
begin
   accept Start (Text : String) do
      Ada.Text_IO.Put_Line (Text);
   end Start;
end Worker;
procedure Initialize (
            Object : in out T) is
begin
   Object.My_Task.Start ("Hey");
            -- Beware, it will hang!
end Initialize;
```

This is not a compiler bug, it is a mandated behavior.

> And, if they are still running, shouldn't a finalize force termination?

You cannot force task termination because the tasking model is cooperative.

In particular you should never use the abort statement unless the task was carefully designed to be abortable. It should not allocate any resources which might get lost upon a preemptive task termination. One way to make a task abortable is to have a controlled object of which does the cleanup:

```
task Abortable is
   Resources : Controlled_Object;
begin
   loop
      ...
   end loop;
end Abortable;
```

when Abortable is aborted the Resources' Finalize will be called so that you could do the necessary cleanup before the task dies. Note that Finalize is abort-deferred, it means that you cannot abort a Finalize (when Finalize is called as a part of finalization), it must complete first.

On the rationale for task components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 13 Jul 2011 20:52:02 +0200
Subject: Task components, the rationale
Newsgroups: comp.lang.ada

OK, to avoid false impression that Ada was carelessly designed, it must be said that there was a valid reason why task components are broken this way.

That is to prevent a much worse disaster when so-called Rosen's trick is used.

Consider the pattern discussed earlier (the Rosen's trick):

```
type T;
task type Worker (Self : not null
                         access T'Class);
type T is new Ada.Finalization.
                    Limited_Controlled
  with record
   My_Worker : Worker (T'Access);
end record;
overriding procedure Initialize(
                  Object : in out T);
procedure Foo (Object : in out T)
   is abstract; -- A primitive operation
```

Now, if My_Worker started before completion of Initialize then this body

```
task body Worker is
begin
   Self.Foo; -- Boom!
```

could call Foo of T or any of its derived type *before* Initialize, i.e. before the object's construction is done! That would be a much worse problem.

There is no simple solution for this. To start with tasks must be inheritable from and their bodies must be primitive or class-wide operations, because aggregation (composition) + Rosen's trick is necessarily broken.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 13 Jul 2011 13:58:36 -0700
Subject: Re: Task components, the rationale
Newsgroups: comp.lang.ada

[…]

I don't even think you need to introduce tasks to show the problem - what if the component is of another controlled type? Then you have two nested calls to distinct Initialize operations - the first one for the component (where you have the discriminant access value to play with) and the second one for the whole, which is too late:

```
with Ada.Finalization;
with Ada.Text_IO;

procedure Test is

   type Outer;

   type Inner (Shell : access Outer)
     is new Ada.Finalization.
                  Limited_Controlled
   with null record;

   overriding procedure Initialize (
                  Self : in out Inner);

   type Outer is new Ada.Finalization.
                  Limited_Controlled
   with record
     I : Inner (Outer'Access);
     Some_Value : Integer;
   end record;

   overriding procedure Initialize (
                  Self : in out Outer);

   procedure Initialize (
                  Self : in out Inner) is
   begin
     Ada.Text_IO.Put_Line
       ("initializing inner,
        Self.Shell.Some_Value =" &
        Integer'Image(Self.Shell.all.
                        Some_Value));
   end Initialize;

   procedure Initialize (
                  Self : in out Outer) is
   begin
     Self.Some_Value := 123;
     Ada.Text_IO.Put_Line
       ("initialized outer, Some_Value ="
        &  Integer'Image(Self.
                        Some_Value));
   end Initialize;

   X : Outer;

begin
   null;
end Test;
```

```
$ gnatmake test
…
$ ./test
initializing inner,
Self.Shell.Some_Value = 0
initialized outer, Some_Value = 123
```

We are messing with the state that does not yet exist. Oops.

> There is no simple solution for this.

You have to just, you know, simply, introduce constructors to the language. This is my pet feature for Ada 2020. :-)

> To start with tasks must be inheritable from and their bodies must be primitive or class-wide operations, because aggregation (composition) + Rosen's trick is necessarily broken.

It's not about tasks, it's about access discriminants to outer records - they introduce circular references (outer has inner, inner knows outer) and as such are evil.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de
Date: Thu, 14 Jul 2011 11:23:07 +0200
Subject: Re: Task components, the rationale
Newsgroups: comp.lang.ada

> […] I don't even think you need to introduce tasks to show the problem - what if the component is of another controlled type?

Yes, but the hack was made specifically for tasks.

If anybody wished to have safe construction/destruction in presence of components spoiled by the Rosen's trick, he would need to postpone parts of constructors/destructors to arrange them in certain order. It guaranteed is impossible to do in certain cases. With task components that manifests itself as a deadlock. (I don't know if the problem is detectable through static analysis, but I doubt it is.)

[…]

Well, constructors need to be properly crafted to handle this. Note that the problem is in inconsistencies at the typing level. Returning to the tasks, you have to properly attribute the task body. Is it a primitive operation? Is it class-wide? etc. Depending on that you should be able or not to dispatch from the body and that will determine the earliest stage of construction when the body is allowed to start and the latest point before it started. I think it would not be possible to do without class-wide constructors, e.g. ones constructing classes out of specific types. (This is a subproblem of a more general problem: dispatching upon construction/destruction.)

[…]

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Thu, 14 Jul 2011 10:52:59 +0200
Subject: Re: Task components, the rationale
Newsgroups: comp.lang.ada

[…]

Out of curiosity, would this be enough? How will it work?

Assuming, naively, not knowing C++, that constructors of C++ could lead the way, I get

```
#include <iostream>

namespace
{
```

```
  class Outer;

  class Inner {
  private:
    Outer* shell;
  public:
    Inner(Outer*);
  };

  class Outer {
  private:
    Inner i;
  public:
    int some_value;
    Outer();
  };

  Inner::Inner(Outer* wrap) {
    this->shell = wrap;
    std::cout << "initializing inner,
                 this->shell->some_value = "
              << this->shell->some_value
              << std::endl;
  }

  Outer::Outer() : i(this) {
    this->some_value = 123;
    std::cout << "initialized outer,
                      this->some_value = "
              << this->some_value
              << std::endl;
  }
}

int main()
{
  Outer x;
  return 0;
}
```

```
$ c++ news23.cpp
$ ./a.out
initializing inner, this->shell-
>some_value = 1606422610
initialized outer, this->some_value = 123
```

From: Maciej Sobczak <maciej@msobczak.com>
Date: Thu, 14 Jul 2011 11:15:53 -0700
Subject: Re: Task components, the rationale
Newsgroups: comp.lang.ada

It would not be sufficient, but it would be necessary.

The point is, in order to solve these kind of puzzles you have to recognize initialization as a special operation (i.e. stop pretending that it can be a regular primitive operation of a type) and use that notion to impose special rules.

In the case of access discriminants the circular relationship is possible to discover statically. After all, the whole T'Access "expression" is special, and allowed only in this particular case. Once you statically know you have a problem, you can work from there - but no matter what kind of restrictions or provisions you impose in the constructor, you have to recognize that it is a special operation, not a regular primitive one.

If you ask me from the top of my head how *exactly* this can be solved, I will not attempt to give a full solution (hey, the committee has a full decade for it ;-)), but one of the possible ideas might involve adding a lifetime information to the access discriminant, just as it is done for tracking scopes of types and objects with anonymous access parameters today. That is, raise Program_Error when you discover that within the constructor of T its access discriminant (pointer to outer) is dereferenced while the outer was not yet initialized.

Most cases (like the two examples we have shown) can be fully analyzed statically for this.

> Assuming, naively, not knowing C++, that constructors of C++ could lead the way,

They will not lead the way in solving the problem of dangling pointers, because this is not the problem that C++ was designed to solve in general. But recognizing that the constructor is a special place is an important contribution.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Jul 2011 18:28:16 -0500
Subject: Re: Task components, the rationale
Newsgroups: comp.lang.ada

[…]

Well, actually, work on Ada 2020 would need to be finished by late 2018 in order to have a good chance of being standardized in 2020. Since it is mid-2011 now, I think that is more like 7 years than 10.

[…]

> but one of the possible ideas might involve adding a lifetime information to the access discriminant, just as it is done for tracking scopes of types and objects with anonymous access parameters today.

That was suggested for Ada 2012 [by me and others], and it turns out that it cannot be done (at least with the sorts of lifetime indications that Ada has used to date). If it was mandated, it would necessarily make Ada implementations far more expensive than they currently are -- so I doubt very much that we'll see that. (Sorry, I don't remember which AI we were discussing at the time, so I can't give you a reference.)

The static accessibility model for access discriminants is *very* problematical; it leads to distributed overhead for functions that might return something with a discriminant -- yet that still is considered preferable to any dynamic model. My preference is to not use them at all (not always possible, as shown by some of these examples).

On invariants and the 'Valid attribute

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 21 Jun 2011 14:08:15 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

> package P1 is

    type T1 is tagged private

        with Invariant => Is_Valid (T1);

Unrelated to Ada, but in theory, an invariant is a private implementation dependent thing. An invariant is trivially true in all public views of the object, i.e. between any two calls to the object's operations. From that follows, when mentioned in a public part then:

```
type T1 is tagged private
  with Invariant => True;
```

(Again, I don't know which ideas Ada designers had about invariants, I am not a language lawyer.)

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Tue, 21 Jun 2011 14:17:11 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

In another theory, the invariant may express things such as

```
Num_Green_Lights (T1) >= 3;
```

or

```
'Length < State_of_Things (T1) * 2;
```

where Num_Green_Lights is a publicly visible function whose result is somehow computed. These predicates would be informative, and formal.

Would they be private implementation dependent things? Or could I expect, seeing the public view and its invariant, the possibility of different implementations (of both the view and the invariant)?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 21 Jun 2011 14:31:49 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

That is not an invariant, but a constraint. Constraint creates a subtype, it is a type-

algebraic operation. Invariant does nothing, it is just a predicate known to be true for all instances of the type in public contexts. As such it can be removed any time without changing the program semantics. A constraint cannot be removed, because its violation is 1) possible and legal, 2) has defined effect (exception). Violation of an invariant in public context is impossible in a correct program.

```
subtype Line is String (1..80);
                -- 1..80 is a constraint
Line'Length = 80 -- This is an invariant
```

P.S. Again, no idea how Ada 2012 treats this issue, differently I guess.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Tue, 21 Jun 2011 15:29:22 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

> That is not an invariant, but a constraint.

I don't see bounds in the above.

I can state, though, that there will be---invariably---at least three green lights because that is a property of how each implementation of the type will be constructed. How is that variant? That is, if the (theoretical) assertion is "at least, no matter what, under all circumstances, in each implementation", isn't this an invariant?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 21 Jun 2011 16:42:05 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

Constraint could be any, not only bounds. It is not the formula, but the meaning of:

```
S = { x | x in T and P (x) }
```

here P is a constraint, which produces S.

```
x in S => Q (x)
```

here Q is an invariant of S. It might happen that Q (x) <=> P (x). Then you could define S using Q, that would make Q constraint and P invariant.

> That is, if the (theoretical) assertion is "at least, no matter what, under all circumstances, in each implementation", isn't this an invariant?

1. Different implementations of the same specification may have different invariants. That are the predicates which cannot be derived from the specification.
2. When a predicate can be derived from the specification that does not yet imply its equivalence to the specification. Invariant does not necessarily defines the type.
3. The difference is the intent. The specification defines the [sub]type. Invariant merely is a predicate provable true for the given implementation of the specification. (Properly constructed invariants can be used in construction of implementations, e.g. Dijkstra's approach to programming, loop invariants etc.)
4. You cannot distinguish predicates used in definitions from ones used in proofs by just looking at them. It is the language's task to do this by syntax means.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 21 Jun 2011 20:53:17 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

> While that's OK in theory, in practice the user may wish methods to check for validity rules defined for a type.

Validity is a misconception. In a properly typed language any value is valid, that is the property of being typed. A value is invalid when the type system was circumvented, which should never happen publicly.

> Then, Is_Valid is abstract enough. And after-all, you already have 'Valid attribute in Ada. This is useful for designs relying on defensive programming and which disallow use of exceptions.

'Valid is a hack around missing value initialization enforcement or some kludges to support Unchecked_Conversion. It cannot justify anything because 'Valid itself lacks credibility.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 21 Jun 2011 22:52:22 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

> […] How do you properly initialise and/or validate values coming from an untrusted external source (i.e. "bus")? Always using the full bit pattern and write the conversion routine yourself?

Yes, I always do exactly this, at least in order to make my program portable. E.g. instead of querying the endianness of the machine and trying to guess what kind of bit shuffling might be appropriate in order to map an external representation onto the machine one through Unchecked_Conversion (provided such mapping exists, which in real life could not be the case when working with bus encodings), I just interpret bits as they are described. It is safer, cleaner, easier to understand, requires no preprocessing. If this could be slightly less efficient, I don't care.

BTW, what I do miss for this stuff is cross type checks. E.g.

```
X : Integer := ...;
if X in Unsigned_32'Range then
```

The problem is that both

```
if X in Integer (Unsigned_32'First)..
        Integer (Unsigned_32'Last) then
```

or

```
if Unsigned_32 (X) in
   Unsigned_32'Range then
```

might fail on different machines. I need a test if the value of the type T can be converted to the type S. (For real types it can be a quite non-trivial to test)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 22 Jun 2011 09:55:43 +0200
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…] I always provide a low-level I/O package which defines operations for getting objects from, say, an octet array and putting it back:

```
procedure Get
      ( Data    : Octet_Array;
        Pointer : in out Integer;
       -- Advanced to the next octet
        Value   : out Clock_Source
       -- May raise Data_Error
      );
```

I never use memory mapping of read data. Incoming data are parsed by a sequence of calls to the corresponding Get operations.

On incomplete types and invariants

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Tue, 21 Jun 2011 01:53:31 -0700
Subject: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

A fairly common Ada idiom is to define the full view of a private type using an incomplete declaration. Thus leaving the actual implementation to the package spec. Trying this out with the public view defined with an invariant lead to a compiler error - is this:

a) expected?

b) an unexpected consequence? or

c) a compiler bug?

Example:

```
package P1 is
  type T1 is tagged private
    with Invariant => Is_Valid (T1);
  function Create  return T1;
```

```
   function Is_Valid (This : T1) return
Boolean;
private
   type Imp;
   type T1 is tagged
      record
         I : Imp;
      end record;
end P1;
```

```
gnatmake -ws -c -u -PH:\Ada\
test_invariants\test_invariants.gpr
p1.ads
gcc -c -g -g -gnatE -gnatVn -gnato
-fstack-check -gnat12 -gnatf -I-
-gnatA H:\Ada\test_invariants\src\p1.ads

p1.ads:11:04: type "Imp" is frozen at line
3 before its full declaration
[...]
p1.ads:15:14: invalid use of type before
its full declaration
gnatmake: "H:\Ada\test_invariants\src\
p1.ads" compilation error

[2011-06-21 09:46:55] process exited
with status 4 (elapsed time: 00.26s)
```

[Correct version of the code included as per a later post —mp]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 21 Jun 2011 03:43:24 -0700
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…]

This looks like a consequence of 13.14(8.2/1): "If an expression is implicitly converted to a type or subtype T, then at the place where the expression causes freezing, T is frozen." (where in this case the expression is T1, which stands for the current instance of the type, and the type T is also T1).

This subclause however seems to contradict 13.14(7.2/3): "At the freezing point of the entity associated with an aspect_specification, any expressions or names within the aspect_specification cause freezing."; this subclause would defer the freezing point of T1 until the end of the enclosing package spec or the declaration of a constant of the type, whichever comes first (as is normal for tagged types).

Another possible interpretation is that Is_Valid must be called as part of the elaboration of type T1, in which case the aspect_specification is a function call, which freezes the types of its parameters (per 13.14(10.1/3)). But I doubt this is true.

So, this looks like an area of the language definition that needs clarifying (but then again, freezing rules have always been difficult to understand). The behavior of the compiler definitely looks undesirable to me. So I vote for b) an unexpected (and undesirable) consequence (of existing rules).

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Tue, 21 Jun 2011 03:46:35 -0700
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

> […] Just out of curiosity: which GNAT flavor do you use for Ada 2012 ?

GNAT GPL 2011

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Tue, 21 Jun 2011 07:31:10 -0400
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…] This has nothing to do with invariants. Incomplete types can only be used in very restricted ways. Not as components. You need to use an access type.

This dates back to Ada 83 -- it's always been illegal, and still is.

When compiling clients of P1 that declare objects of type T1, how would the compiler know the size? It could treat it as dynamic, or it could take a peek at the body, but if either of those was the intended compilation model, then there would be no need for private parts in the first place -- we'd put the completion of a private type in the body, where it belongs.

I suggest you erase the invariant, fix the errors, and then put the invariant back in. There's nothing wrong with your invariant.

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Tue, 21 Jun 2011 05:01:48 -0700
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

[…] Ok, with no Invariant:

```
package P1 is
   type T1 is tagged private;
   function Is_Valid (This : T1)
            return Boolean;
private
   type Imp;
   type Imp_Ref is not null access Imp;
   type T1 is tagged
      record
         I : Imp_Ref;
      end record;
end P1;
```

I get:

```
gnatmake -d -PH:\Ada\test_invarients\
test_invarients.gpr p1.ads
gcc -c -g -g -gnatE -fstack-check -gnato
-gnatf -fcallgraph-info=su,da -gnat12 -I-
-gnatA H:\Ada\test_invarients\src\p1.ads

cannot generate code for file p1.ads
(package spec)
gnatmake: "H:\Ada\test_invarients\
src\p1.ads" compilation error

[2011-06-21 12:58:10] process exited
with status 4 (elapsed time: 00.37s)
```

i.e. no error.

The most reduced version I can come up with is now:

```
package P1 is
   type T1 is tagged private
     with Invariant => True;
      -- NB: not even a 'real' function
private
   type Imp;
   type Imp_Ref is not null access Imp;
   type T1 is tagged
      record
         I : Imp_Ref;
      end record;
end P1;
```

which produces the same original error:

```
[...]
p1.ads:5:04: type "Imp" is frozen at line
2 before its full declaration
gnatmake: "H:\Ada\test_invarients\
src\p1.ads" compilation error
[…]
```

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Tue, 21 Jun 2011 05:22:04 -0700
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

> […] Looks like a compiler bug. […]

Thanks! I'll just have to stick with "with Post =>" on all the operations instead for now…until GNAT GPL 2012!! :-)

Shame the GPL version only updates once per year…I suppose there is not any chance of even a twice-yearly release schedule?…

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Tue, 21 Jun 2011 06:00:59 -0700
Subject: Re: Ada2012 Invariants and obaque types
Newsgroups: comp.lang.ada

> […] Or don't use incomplete types completed in the body. […]

Yes, I was thinking of replacing this with discriminant to an interface instead…(think 'strategy pattern' but with only 1 strategy)…wonder if hold Invar/Pre/Post will cope with that!! :-)

On multiple dispatch in Ada

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Sat, 11 Jun 2011 12:13:21 +0200
Subject: Multiple dispatch
Newsgroups: comp.lang.ada

[…] In your opinion, what is the best known design pattern for multiple dispatch in Ada ? (as far as I know, there is no way to get multiple dispatching in Ada the direct way).

As with any design matters, an answer to this question will probably depends on the concrete case, so here is an overview of the concrete matter: it deals with serialization.

An example: you have multiple object of different types rooted at a some root type; you also have multiple containers of different types too, also rooted at another root type. Now, say objects are all to have a serialization methods, a different one for each type. This could be dispatching, OK, except that you have the requirement these objects are also to be serialized a way or another, depending on the container which will hold the serialized data.

Add to this that you can't change this, because this is part of some standard or any other kind of things already fixed.

Say Object_1_Type, to be serialized to Container_1_Type, will use Method_1_1

… Object_1_Type, to be serialized to Container_2_Type, will use Method_1_2

… Object_2_Type, to be serialized to Container_1_Type, will use Method_2_1

… Object_2_Type, to be serialized to Container_2_Type, will use Method_2_2

… and so on

A quick solution could be:

1) Define two methods for object: one would be Serialize_To_Container_1 and Serialize_To_Container_2.

2) Then, a master Serialize method could get two parameters, one object and one container,

3) This method would discriminate on container's type, so would invoke either Serialize_To_Container_1 or Serialize_To_Container_1 depending on the container's type, and this call would be dispatching on object's type.

OK, but why not the opposite? And then, where the master dispatcher should reside ? In the module defining objects? In the module defining containers? In a third module? Who should own the knowledge about serialization? I feel it's natural to say, Objects, of course; but this also requires dispatching on container's types… Still seems natural the serialization should be driven by objects, at least because objects to be serialized may hold private stuff, or else are the only ones to know which of their properties are to be stored and which are to be derived from the ones stored.

Alternatively, may be a seed of a solution: the containers would define some serialization primitives for some basic property types objects are made of, and objects could request the container to provide these method, via dispatching calls.

This would end into…

Steps for the serialization of an object to a container:

1) Determine the object serialization method to use depending on its type.

2) The object is composed of properties of types Property_1_Type, Property_2_Type, and so on.

3) The containers provides methods Serialize_Property_Type_1, Serialize_Property_Type_2, and so on.

4) The object's serialization method invokes these container's methods, via dispatching calls.

Seems OK? Any one see a better design pattern? Do someone see something wrong with this repartitions of knowledge and responsibilities between modules?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 11 Jun 2011 13:52:05 +0200
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

[…] Yes, that looks like the usual pattern for protocols and similar stuff, when objects do not depend on the containers. The opposite case is represented by drivers, when objects are maintained by the driver, you might first dispatch on the driver type and then on the type of its objects.

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Sat, 11 Jun 2011 15:19:00 +0200
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

What kind of dependencies do you have in mind? Object existence depending on the container ? Or object's properties depending on the container?

I was trying to figure many cases (in an attempt to invalidate the design, and fail with that, to prove this is the good design); among others, this one : if the container know enough about the object types because it is fully dedicated to these object types, one may argue the container could also serialize objects on its own side. However, this would imply a switch on object types, which is not clean, due to an implicit though behind this: dispatching calls and switch are somewhat similar things, however with an important difference, which is that with dispatching calls, the corresponding switch is always owned by (under the responsibility of) the type on which the switch is to be done, while with the switch approach, anything could do a switch. The former more enforce structuring.

Another approach, with a more concrete-like example (which is not the one I am actually dealing with… I wont tell more, as I prefer to keep this talk as much abstract as possible) : you have two types, one for a character string and one for an array of numbers, and two containers. Say one container is a file and the other is anything else you wish (could be a serial communication wire plugged to some device, as an example). Let's say there are two way to serialize each type. For the string, there is a C-like serialization, that is, all characters first, with a final Unicode U+0000, and a Pascal-like serialization, with a length first and then the all characters. Let's say there is something similar with the array of numbers: it could either be serialized with a kind of null terminator, and the other way, starting with its length and then its numbers. Say each of the two container expect one or the other serialization.

This case seems more ambiguous at first sight, at it could seem as much easy and clean to dispatch on either the container or the object. Eh, but only one container type knows about what null terminators are and only one knows about what lengths are. Now let's say none of the array of numbers or the character strings, know about what null terminators are. With such a case, wouldn't it be better to dispatch first on the container? Is that the kind of dependency you though about?

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 11 Jun 2011 15:57:34 +0200
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

> […] What kind of dependencies do you have in mind? Object existence depending on the container? Or object's properties depending on the container?

Any dependencies (correlations). Imagine the dispatching table. It is a 2D matrix for double dispatch. In full dispatch the table is irregular, i.e. there is no preferred way to index this matrix either by columns or by rows.

In special cases there can be similarities between rows or between columns, in that case you dispatch across the most varying dimension leaving the least variance to the secondary dispatch.

> With such a case, wouldn't it be better to dispatch first on the container? Is that the kind of dependency you though about?

Yes, that is full dispatch, which is not decomposable. Returning to the example with the 2D matrix of dispatch D, when it was the representation:

D (i, j) = a (i) * b (j)

you can decompose full dispatch into cascaded dispatch (first a(i), then b(j)).

[…]

Typically driver has a task processing I/O requests. Queuing the request naturally goes to the driver, so you dispatch first on the driver object. Once you dequeue the request, you dispatch on the object handled by the driver, e.g. write analogue 24-bit output etc.

From: Emmanuel Briot <briot.emmanuel@gmail.com>
Date: Sat, 11 Jun 2011 11:42:25 -0700
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

I have prepared recently a "gem" on the Visitor design pattern (see the Gang of Four book on design patterns), that will be published this month on the Adacore web site. I think you might find the pattern interesting if you do not know about it yet, so looking at the book might be of interest to you.

[unfortunately the Ada Gem has not been published yet (Aug 1, 2011) —mp]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sun, 12 Jun 2011 00:13:03 -0500
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

[…]

Serialization is one of those problems that really requires being implemented with composition, such that every type knows how to finalize itself and dispatches properly to use the similar routine of any component types. (It's annoying that Ada compilers know how to do this automatically, given that they do it for streams and for equality, but they won't help you do it in other cases.)

However, if you have external requirements that prevent you from doing that in the natural way, then some other solution will be needed. That solution is likely to look like a hack -- and that's OK, because the problem itself requires a hack (it doesn't map to a natural solution).

Which is a long way of saying that there are a lot of problems that can't really be solved elegantly, and it isn't very worthwhile to look for the perfect design for such problems. Come up with some design that solves the problem and don't obsess about it too much.

From: Natasha Kerensikova <lithiumcat@gmail.com>
Date: Wed, 15 Jun 2011 10:30:43 +0000
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

> […] Imagine the dispatching table. It is a 2D matrix for double dispatch. In full dispatch the table is irregular, i.e. there is no preferred way to index this matrix either by columns or by rows.

I remember having faced that kind of issues with C through dynamic linker introspection : there is a function that returns a function pointer corresponding to the given symbol name. I crafted the symbol name using an operation like "dispatch_prefix_" & First_Tag & "_" & Second_Tag

This makes the dispatch table completely isotropic and allows to implement any element of the matrix in any compilation unit (as long as it is linked (statically or dynamically) to the final binary). I found this feature very nice when the "glue" between the row type and the column type does not obviously belong to either type (in the original example, that would be a very special representation of a given general data type in a given general container).

I guess the introspection of object files is too low-level for Ada, and even importing libdl from C wouldn't be of much use because of the name-mangling performed by Ada compiler (or you have to use C symbol names and give up namespaces, case-insensitivity and other cool Ada features).

I know other high-level languages do have introspection mechanisms, but I haven't seen anything like that in Ada. Have I missed them?

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Wed, 15 Jun 2011 16:59:13 +0200
Subject: Re: Multiple dispatch
Newsgroups: comp.lang.ada

[…] You could build something around (external) tag names. […]

Ada and security code standards

From: Nasser M. Abbasi <nma@12000.org>
Date: Sat, 28 May 2011 11:53:25 -0700
Subject: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

I saw that CMU makes now what is called CERT (secure coding standards) for different languages. They have Java, C, C++ in there.

These are supposed to be rules that a programmer should adopt to make the code written by that language more 'safe' and 'secure'

Here is the one for C for example:

https://www.securecoding.cert.org/confluence/display/seccode/CERT+C+Secure+Coding+Standard

I was wondering if Ada would benefit of having something like these secure programming rules customized for Ada.

Or if it is even needed as much for Ada? Some of the rules seem good to know about.

May be some of this material is already in the Ada rationale in different places. […]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sat, 28 May 2011 21:32:37 +0200
Subject: Re: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

[…] This is addressed by ISO/IEC JTC 1/SC 22/WG 23 Programming Language Vulnerabilities [1].

There are language-specifix annexes for Ada, SPARK and several other languages. The annexes for Ada and SPARK are in the Ada User Journal [2], Volume 32, No 3 and 4 respectively [Volume 31 —mp].

[1] http://www.aitcnet.org/isai/

[2] http://www.ada-europe.org/journal.html

From: Simon Wright <simon@pushface.org>
Date: Sat, 28 May 2011 22:37:55 +0100
Subject: Re: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

I wouldn't have rated

https://www.securecoding.cert.org/confluence/display/seccode/POS39-C.+Use+the+correct+byte+ordering+when+transferring+data+between+systems

as a _guideline_ exactly!

Or

https://www.securecoding.cert.org/confluence/display/seccode/FIO09-C.+Be+careful+with+binary+data+when+transferring+data+across+systems

[…]

From: Mark Ngbapai <lightningbolt31@gmail.com>
Date: Sun, 29 May 2011 06:29:15 -0700
Newsgroups: comp.lang.ada

Subject: Re: Does Ada need a 'secure coding standard' as well?
[…]

There are references to Ada in the NASA Software Safety Book, it is worth reading and can be downloaded freely at:

http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sun, 29 May 2011 08:23:52 -0700
Subject: Re: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

[…]

> Or if it is even needed as much for Ada?

Apparently it is, as several such documents were written for Ada.

Apart from those already mentioned, these two might be of interest:

"Ada95 Trustworthiness Study: Guidance on the Use of Ada95 in the Development of High Integrity Systems"

ISO/IES TR 15942: "Guide for the use of the Ada programming language in high integrity systems"

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sun, 29 May 2011 23:03:07 +0200
Subject: Re: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

I don't think the CERT guide is targeted at high-integrity systems.

It's intended for an extremely broad range of things, from server software to productivity applications for end users. This means that certain features are taken for granted, such as the need to restart applications from time to time (because of a non-compacting dynamic memory manager) and the ability of software to scale with available resources.

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 30 May 2011 12:25:20 +0200
Subject: Re: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

[…]

> In the particular area of Object-Oriented Design applied to High-Integrity applications, there is one AdaCore worked on:
>
> www.open-do.org/wp-content/uploads/2011/04/HighIntegrityAda.pdf
>
> I know Jean-Pierre Rosen also took part to a similar workshop, but I have no reference to this.

There will be a panel on this topic at the upcoming Ada-Europe conference. One more reason to attend ;-)

From: J-P. Rosen <rosen@adalog.fr>
Date: Mon, 30 May 2011 12:27:28 +0200
Subject: Re: Does Ada need a 'secure coding standard' as well?
Newsgroups: comp.lang.ada

> Do you know some reference to papers published after the workshop Jean-Pierre Rosen talked about here some months ago ? This was about OOD in applications with hard requirement for safety. I posted one link to such a document, but a pointer to this other material would still be worth.

I wrote one that I sent (a bit late, sorry) to Ada Letters. Should be in the next issue.

Conference Calendar

Dirk Craeynest

K.U.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.

2011

☺ October 04-07 30th **IEEE International Symposium on Reliable Distributed Systems** (SRDS'2011), Madrid, Spain. Topics include: distributed systems design, development and evaluation, particularly with emphasis on reliability, availability, safety, security, trust and real time; high-confidence systems; distributed objects and middleware systems; formal methods and foundations for dependable distributed computing; analytical or experimental evaluations of dependable distributed systems; etc.

October 10-12 13th **International Symposium on Stabilization, Safety, and Security of Distributed Systems** (SSS'2011), Grenoble, France. Topics include: Fault-Tolerance and Dependable Systems, Safety and Verification, Security, etc.

☺ October 10-14 20th **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2011), Galveston Island, Texas, USA. Topics include: Parallel computational models; Compilers and tools for parallel computer systems; Support for correctness in hardware and software (esp. with concurrency); Parallel programming languages, algorithms and applications; Middleware and run time system support for parallel computing; Applications and experimental systems studies; etc.

October 17-20 18th **Working Conference on Reverse Engineering** (WCRE'2011), Lero, Limerick, Ireland. Topics include: Program comprehension; Mining software repositories; Empirical studies in reverse engineering; Redocumenting legacy systems; Reverse engineering tool support; Reengineering to distributed architectures; Software architecture recovery; Program analysis and slicing; Reengineering patterns; Program transformation and refactoring; etc.

☺ October 20-22 12th **International Conference on Parallel and Distributed Computing, Applications, and Techniques** (PDCAT'2011), Gwangju, Korea. Topics include: all areas of parallel and distributed computing; Reliability, and fault-tolerance; Formal methods and programming languages; Software tools and environments; Parallelizing compilers; Component-based and OO Technology; Parallel/distributed algorithms; Task mapping and job scheduling; etc.

☺ October 22-27 **ACM Conference on Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2011), Portland, Oregon, USA. Includes: panels on "Language-based security as extreme modularity", Multicore, manycore, and cloud computing: is a new programming language paradigm required?", ...; Educators and Trainers Symposium; etc.

☺ October 23 **Workshop on Transitioning to Multicore** (TMC'2011). Topics include: tools and systems for parallel programming that are interoperable with legacy code, minimize the annotation burden for developers, and match well with current industry practice; Surveys or empirical studies measuring current practice for multicore programming in industry; Field studies identifying barriers and benefits to using existing tools; Analysis tools focused on correctness, performance, or understandability of existing programs; New programming models which are interoperable with legacy multithreaded systems; etc.

☺ October 23 1st **Workshop on Combined Object-Oriented Modeling and Programming** (COOMP'2011). Topics include: Differences and similarities between modeling and programming; Modeling constructs not supported by programming languages and vice

versa; Support for concurrent / distributed modeling and programming; Tools for modeling and programming; Implementation techniques; New mechanisms to raise the level of abstraction; Experience reports regarding pros/cons in using separate modeling and programming languages, modeling in a programming language, executable modeling languages, ...; etc.

☺ October 24 3rd **Workshop on Evaluation and Usability of Programming Languages and Tools** (PLATEAU'2011). Topics include: methods, metrics and techniques for evaluating the usability of languages and language tools, such as empirical studies of programming languages; methodologies and philosophies behind language and tool evaluation; software design metrics and their relations to the underlying language; user studies of language features and software engineering tools; critical comparisons of programming paradigms; tools to support evaluating programming languages; etc.

☺ October 23 6th **Workshop on Programming Languages and Operating Systems** (PLOS'2011), Cascais, Portugal. Topics include: critical evaluations of new programming language ideas in support of OS construction; type-safe languages for operating systems; language-based approaches to crosscutting system concerns, such as security and run-time performance; language support for system verification; the use of OS abstractions and techniques in language runtimes; etc.

October 25-28 13th **International Conference on Formal Engineering Methods** (ICFEM'2011), Durham, UK. Topics include: Abstraction and refinement; Formal specification and modelling; Software verification; Program analysis; Tool development and integration; Software safety, security and reliability; Experiments involving verified systems; Applications of formal methods; etc.

♦ Nov 06-10 **ACM SIGAda Annual International Conference on Ada and Related Technologies** (SIGAda'2011), Denver, Colorado, USA. Sponsored by ACM SIGAda, in cooperation with SIGAPP, SIGBED, SIGCAS, SIGCSE, SIGPLAN, Ada-Europe, and the Ada Resource Association (cooperation approvals pending). Deadline for early registration: October 8, 2011.

November 09-11 30th **International Conference of the Chilean Computer Science Society** (SCCC'2011), Curicó, Chile. Topics include: Theory of Computer Science, Security, Distributed and Parallel Systems, Software Engineering, Programming Languages, Computer Science and Education, etc.

Novermber 14-18 9th **International Conference on Software Engineering and Formal Methods** (SEFM'2011), Montevideo, Uruguay. Topics include: 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; light-weight and scalable formal methods; tool integration; applications of formal methods, industrial case studies and technology transfer; etc.

☺ December 07-09 17th **IEEE International Conference on Parallel and Distributed Systems** (ICPADS'2011), Tainan, Taiwan. Topics include: Parallel and Distributed Applications and Algorithms; Multi-core and Multithreaded Architectures; Resource Provision, Monitoring, and Scheduling; Security and Privacy; Dependable and Trustworthy Computing and Systems; Real-Time Systems; etc.

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

December 18-21 18th IEEE **International Conference on High Performance Computing** (HiPC'2011), Bengaluru, Bangalore, India. Topics include: Parallel and Distributed Algorithms, Parallel Languages and Programming Environments, Scheduling, Fault-Tolerant Algorithms and Systems, Scientific/Engineering/Commercial Applications, Compiler Technologies for High-Performance Computing, Software Support, etc. Deadline for early registration: November 14, 2011.

2012

☺ January 25-27 39th **ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages** (POPL'2012), Philadelphia, USA. Topics include: all aspects of programming languages and systems, with emphasis on how principles underpin practice.

February 22-25 5th **India Software Engineering Conference** (ISEC'2012), Kanpur, India. Topics include: Testing and Static Analysis, Specification and Verification, Model Driven Software Engineering, Software

Architecture and Design, Tools and Environments, Development Paradigms and Processes, Maintenance and Evolution, Quality Management, Component Based Software Engineering, Object-Oriented Analysis and Design, Distributed Software Development, Case Studies and Industrial Experience, Software Engineering Education, Mining Software Repositories, etc.

☺ Feb 29 – Mar 03 43rd **ACM Technical Symposium on Computer Science Education** (SIGCSE'2012), Raleigh, North Carolina, USA.

Mar 24 – Apr 01 **European Joint Conferences on Theory and Practice of Software** (ETAPS'2012), Tallinn, Estonia. 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; TACAS, Tools and Algorithms for the Construction and Analysis of Systems.

☺ March 25-29 SAC2012 - **Track on Object-Oriented Programming Languages and Systems** (OOPS'2012), Riva del Garda, Trento, Italy. Topics include: Language design and implementation; Type systems, static analysis, formal methods; Integration with other paradigms; Aspects, components, and modularity; Distributed, concurrent or parallel systems; Interoperability, versioning and software adaptation; etc.

March 25-30 11th **International Conference on Aspect-Oriented Software Development** (AOSD'2012), Potsdam, Germany. Topics include: Complex systems; Software design and engineering; Programming languages (language design, compilation and interpretation, verification and static program analysis, ...); Varieties of modularity (model-driven development, generative programming, software product lines, contracts and components, ...); Tools (evolution and reverse engineering, crosscutting views, refactoring, ...); Applications (distributed and concurrent systems, middleware, ...); etc. Deadline for submissions: October 13, 2011 (abstracts round 3), October 17, 2011 (papers round 3).

♦ June 20-24 17th **International Conference on Reliable Software Technologies - Ada-Europe'2012**, Stockholm, Sweden. Sponsored by Ada-Europe, in cooperation with ACM SIGAda (approval pending). Deadline for submissions: November 28, 2011 (papers, tutorials, workshops), January 12, 2012 (industrial presentations).

Advance Program

ACM Annual International Conference on Ada and Related Technologies: Engineering Safe, Secure, and Reliable Software

Magnolia Hotel
Denver, Colorado 80202 (USA)
November 6-10, 2011

Sponsored by the ACM Special Interest Group on the Ada Programming Language (SIGAda)
in cooperation with Ada-Europe, Ada Resource Association, and ACM Special Interest Groups on Embedded Systems, Programming Languages, Computers and Society, and Computer Science Education

Featured Speakers and Session Leaders

Everything I Know I Learned from Ada
Grady Booch, IBM Fellow
(Chief Scientist for Software Engineering, IBM Research)
Presentation via Second Life

Why I Came Back To Ada
Martin Carlisle, Ph.D.
(US Air Force Academy)

Software Safety, and Related Language Considerations
Jim Rogers (MEI Technologies, Inc.)

How to Make Ada go "Viral"
JP Rosen
(AdaLog)

Corporate Sponsors – Platinum

AdaCore
The GNAT Pro Company

Corporate Sponsors – Silver

Ellidiss Software
TNI Europe Limited

LDRA

ACM SIGAda Annual International Conference, November 6-10, 2011
Summary Conference Schedule

PRE-CONFERENCE TUTORIALS Sunday, November 6

Introduction to Ada (SF1 - Full Day)
Michael Feldman (George Washington Univ. retired)
How to measure and optimize reliable embedded software (SA1 - Morning)
Ian Broster(Rapita Systems)
Service-Oriented Architecture (SOA) Concepts and Implementations (SA2 - Morning)
Ricky Sward (MITRE Corp) and Jeff Boleng (USAF Academy)
DO-178C: The Next Avionics Safety Standard (SP1 - Afternoon)
Ben Brosgol (AdaCore)
Improving the Quality of Ada Software with Range Analysis (SP2 - Afternoon)
Jay Abraham (The Mathworks, Inc.)

PRE-CONFERENCE TUTORIALS Monday, November 7

Building Embedded Real-Time Applications (MF1 - Full Day)
John McCormick (Univ. of Northern Iowa) and Frank Singhoff (Univ. of Brest)
Experimenting with ParaSail – Parallel Specification and Implementation Language (MA1 - Morning)
Tucker Taft (SofCheck)
Ada Coding Standards (MP1 - Afternoon)
J-P Rosen (AdaLog)

TECHNICAL PROGRAM Tuesday, November 8

9 – 10:30 am	**Greetings from SIGAda and Conference Officers** ***Keynote Address:*** **Everything I Know I Learned from Ada** *Grady Booch (IBM Fellow, Chief Scientist for Software Engineering, IBM Research)* **Via Second Life Broadcast**
10:30 – 11:00 am	**Morning Break - Exhibits Open**
11 – 12:30 pm	**A Parallel Programming Model for Ada** *Hazem Ali and Luís Miguel Pinho (CISTER Research Centre, Polytechnic Institute of Porto, Portugal)* **Stack Safe Parallel Recursion with Paraffin** *Brad Moore (General Dynamics, Canada)* **AdaCore Sponsor Presentation**
12:30 – 2:00 pm	**Mid-day Break and Exhibits**
2 – 4:00 pm	**Panel: How to Make Ada go "Viral"** *JP Rosen (AdaLog), Tucker Taft (SofCheck), Brad Moore (GD Canada)* **Ellidiss Sponsor Presentation**
4:00 – 4:30 pm	**Afternoon Exhibits**
4:30 – 5:30	**Birds of a feather: ParaSail**

Tuesday Evening Reception (7:00pm - 10:00pm)

TECHNICAL PROGRAM Wednesday, November 9

9:- 10:30 am	**SIGAda Awards** **Keynote Address: Why I Came Back to Ada** *Martin Carlisle (US Air Force Academy)*
10:30 – 11:00 am	**Morning Break and Exhibits**
11 am – 12:30 pm	**Software Vulnerabilities Precluded by SPARK** *Paul E. Black, PhD and Chris E. Dupilka (NIST and US DoD) - F. David Jones and Joyce L. Tokar, PhD (Pyrrhus Software)* **Enhancing SPARK's Contract Checking Facilities Using Symbolic Execution** *John Hatcliff, Jason Belt, and Robby Robby (Kansas State Univ.)* **LDRA Sponsor Presentation**
12:30 – 2:00 pm	**Mid-day Break and Exhibits**
2 – 4:00 pm	**An Ada Design Pattern Recognition Tool for AADL Performance Analysis** *V. Gaudel, F. Singhoff, A. Plantec, and S. Rubini (Univ. of Brest, France) - P. Dissaux and J. Legrand (Ellidiss Software)* **Improving the Quality of Ada Software with Range Analysis** *Jay Abraham, Christian Bard, Jeff Chapple, Patrick Munier, and Cyril Preve (The Mathworks, Inc.)* **Making the Non-executable ACATS Tests Executable** *Dan Eilers, (Irvine Compiler Corp.) and Tero Koskinen*
4:00 – 4:30 pm	**Afternoon Break**
4:30 – 5:30	**Birds of a Feather: GNAT**
5:30 – 7:00 pm	**Dinner Break**
7:00 – 11:00	**Workshops**

TECHNICAL PROGRAM Thursday, November 10

9 – 10:15am	**Conference Best Paper Awards** **Invited Talk: Software Safety, and Related Language Considerations** *Jim Rogers* (MEI Technologies, Inc.)
10:15 – 10:30 am	**Morning Break**
10:30 – 12 noon	**Towards Ada 2012, An Interim Report** *John Barnes (*John Barnes Informatics*)* **Ada-Europe 2012 Announcement** **ACM SIGAda 2012 Announcement** **Closing Remarks**

http://www.sigada.org/conf/sigada2011

Dear reader,

It is my pleasure to bring to your attention the Call for Contributions for the 17th International Conference on Reliable Software Technologies - Ada-Europe 2012, that will take place in the beautiful venue of Stockholm, Sweden, from June 11 to June 15, 2012.

This conference is the forthcoming edition in a series of annual international conferences, regularly held since the early 80's, under the auspices of, and organization by, Ada-Europe, the European non-profit organization that promotes the knowledge and use of the Ada programming language and reliable software technologies.

We expect that next year's conference will confirm the success of the 2011 event, in Edinburgh, UK, on June 20-24, which attracted over 130 delegates coming from Belgium, Brazil, Canada, Denmark, Egypt, Finland, France, Germany, Israel, Italy, Norway, Poland, Portugal, Russia, Slovakia, South Africa, Spain, Sweden, Switzerland, The Netherlands, UK and USA, representing more than 20 Universities and 50 companies.

The 17th International Conference on Reliable Software Technologies – Ada-Europe 2012, addresses a wide range of topics of interest, under the general umbrella of reliable software systems, with a strong but not exclusive interest on Ada-related views on the subject. To mark the completion of the technical work for the Ada 2012 standard revision process, contributions that discuss the potential of the revised language are especially sought. The challenges presented to the development of reliable software by the need for multicore programming models will be another prime topic of interest to the 2012 conference.

We therefore invite you to consider contributing, to the conference by the submission of a paper or proposal for industrial presentation, exhibition, demonstration, panel discussion, workshop, and of course by promoting the event to your contacts and working environment.

Looking forward for your participation.

With my best wishes,

Ahlan Marriott
Conference Chair

Call for Papers

17th International Conference on Reliable Software Technologies Ada-Europe 2012

11-15 June 2012, Stockholm, Sweden

http://www.ada-europe.org/conference2012

Conference Chair

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

Program Co-Chairs

Mats Brorsson
KTH Royal Institute of Technology, Sweden
matsbror@kth.se

Luís Miguel Pinho
CISTER Research Centre/ISEP, Portugal
lmp@isep.ipp.pt

Tutorial Chair

Albert Llemosí
Universitat de les Illes Balears, Spain
albert.llemosi@uib.cat

Industrial Chair

Jørgen Bundgaard
Rovsing A/S, Denmark
jbg@rovsing.dk

Publicity Chair

Dirk Craeynest
Aubay Belgium & K.U.Leuven, Belgium
Dirk.Craeynest@cs.kuleuven.be

Local Chair

Rei Stråhle
Ada-Sweden
rei@ada-sweden.org

In cooperation with
ACM SIGAda
(approval pending)

General Information

The 17th International Conference on Reliable Software Technologies – Ada-Europe 2012 will take place in Stockholm, Sweden. Following its traditional style, the conference will span a full week, including, from Tuesday to Thursday, three days of parallel scientific, technical and industrial programs, along with parallel tutorials and workshops on Monday and Friday.

Schedule

28 November 2011	**Submission of regular papers, tutorial and workshop proposals**
12 January 2012	Submission of industrial presentation proposals
3 February 2012	Notification of acceptance to all authors
2 March 2012	Camera-ready version of regular papers required
11 May 2012	Industrial presentations, tutorial and workshop material required

Topics

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

To mark the completion of the technical work for the **Ada 2012** standard revision process, contributions that discuss the potential of the revised language are sought after. In parallel, facing the challenges presented to the development of reliable concurrent software, **multicore programming models** is added to the conference topics of interest.

Topics of interest to this edition of the conference include but are not limited to:

- **Multicore Programming Models**: Reliable Parallel Software, Parallel Execution of Ada Programs, Compositional Parallelism Models, Performance Modelling, Deterministic Debugging.
- **Real-Time and Embedded Systems**: Real-Time Software, Architecture Modeling, HW/SW Co-Design, Reliability and Performance Analysis.
- **Theory and Practice of High-Integrity Systems**: Distribution, Fault Tolerance, Security, Reliability, Trust and Safety, Languages Vulnerabilities.
- **Software Architectures**: Design Patterns, Frameworks, Architecture-Centered Development, Component and Class Libraries, Component-based Design and Development.
- **Methods and Techniques for Software Development and Maintenance**: Requirements Engineering, Object-Oriented Technologies, Model-driven Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- **Enabling Technologies**: Compilers, Support Tools (Analysis, Code/Document Generation, Profiling), Run-time Systems, Distributed Systems, Ada and other Languages for Reliable Systems.
- **Software Quality:** Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Energy, Games and Serious Games, etc.
- **Experience Reports**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **The Future of Ada**: New language features, implementation and use issues; positioning in the market and in education; where should Ada stand in the software engineering curriculum; lessons learned on Ada Education and Training Activities with bearing on any of the conference topics.

Program Committee

Alan Burns, University of York, UK
Albert Llemosí, Universitat de les Illes Balears, Spain
Alfons Crespo, Universidad Politécnica de Valencia, Spain
Bernd Burgstaller, Yonsei University, Korea
Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium
Ed Schonberg, AdaCore, USA
Elena Troubitsyna, Åbo Akademi University, Finland
Erhard Plödereder, Universität Stuttgart, Germany
Franco Mazzanti, ISTI-CNR Pisa, Italy
Jan Jonsson, Chalmers University of Technology, Sweden
Jérôme Hugues, ISAE Toulouse, France
Johann Blieberger, Technische Universität Wien, Austria
John McCormick, University of Northern Iowa, USA
Jorge Real, Universidad Politécnica de Valencia, Spain
Jørgen Bundgaard, Rovsing A/S, Denmark
José Javier Gutiérrez, Universidad de Cantabria, Spain
José Ruiz, AdaCore, France
Juan A. de la Puente, Universidad Politécnica de Madrid, Spain
Juan Zamorano, Universidad Politécnica de Madrid, Spain
Julio Medina, Universidad de Cantabria, Spain
Jürgen Mottok, Regensburg University of Applied Sciences, Germany
Kristina Lundqvist, Mälardalen University, Sweden
Laurent Pautet, Telecom Paris, France
Luís Miguel Pinho, CISTER Research Centre/ISEP, Portugal
Mats Brorsson , KTH Royal Institute of Technology, Sweden
Michael González Harbour, Universidad de Cantabria, Spain
Peter Hermann, Universität Stuttgart, Germany
Santiago Urueña, GMV, Spain
Sergio Sáez, Universidad Politécnica de Valencia, Spain
Stephen Michell, Maurya Software, Canada
Ted Baker, US National Science Foundation, USA
Theodor Tempelmeier, Univ. of Applied Sciences Rosenheim, Germany
Tullio Vardanega, Università di Padova, Italy

Industrial Committee

Ahlan Marriott, White-Elephant GmbH, Switzerland
Alok Srivastava, TASC Inc, USA
Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium
Erik Wedin, Saab, Sweden
Hubert Keller, Forschungszentrum Karlsruhe GmbH, Germany
Ian Broster, Rapita Systems, UK
Ismael Lafoz, Airbus Military, Spain
Jamie Ayre, AdaCore, France
Jean-Loup Terraillon, European Space Agency, The Netherlands
Jean-Pierre Rosen, Adalog, France
Jørgen Bundgaard, Rovsing A/S, Denmark
Paolo Panaroni, Intecs, Italy
Paul Parkinson, Wind River, UK
Rod Chapman, Altran Praxis Ltd, UK
Rod White, MBDA, UK

Call for Regular Papers

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall be in English, complete and not exceeding 14 LNCS-style pages in length. Authors should submit their work via the EasyChair conference system (*http://www.easychair.org/conferences/?conf=adaeurope2012*). The format for submission is solely PDF. Should you have problems to comply with format and submission requirements, please contact the *Program Chairs*.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the start of the conference. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by March 2, 2012. For format and style guidelines authors should refer to the following URL: *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The conference is ranked class A in the CORE ranking, is among the top quarter of CiteSeerX Venue Impact Factor, and listed in DBLP, SCOPUS and Web of Science Conference Proceedings Citation index, among others.

Awards

Ada-Europe will offer honorary awards for the best regular paper and the best presentation.

Call for Industrial Presentations

The conference also seeks industrial presentations which deliver value and insight, but may not fit the selection process for regular papers. Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation by January 12, 2012. Please follow the submission instructions on the conference website. The *Industrial Committee* will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by May 11, 2012, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the *Industrial Chair* directly.

Call for Tutorials

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

Call for Workshops

Workshops on themes that fall within the conference scope may be proposed. Proposals may be submitted for half- or full-day events, to be scheduled at either end of the conference week. Workshop proposals should be submitted to the *Conference Chair*. The workshop organizer shall also commit to preparing proceedings for timely publication in the Ada User Journal.

Call for Exhibitors

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

Grant for Reduced Student Fees

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

Rationale for Ada 2012: Introduction

John Barnes

John Barnes Informatics, 11 Albert Road, Caversham, Reading RG4 7AN, UK; Tel: +44 118 947 4125; email: jgpb@jbinfo.demon.co.uk

Abstract

This is the first of a number of papers describing the rationale for Ada 2012. In due course it is anticipated that the papers will be combined (after appropriate reformatting and editing) into a single volume for formal publication.

This first paper covers the background to the development of Ada 2012 and gives a brief overview of the main changes from Ada 2005. Later papers will then look at the changes in more detail.

Keywords: rationale, Ada 2012.

1 Revision process

Ada has evolved over a number of years and, especially for those unfamiliar with the background, it is convenient to summarize the processes involved. The first version was Ada 83 and this was developed by a team led by the late Jean Ichbiah and funded by the USDoD. The development of Ada 95 from Ada 83 was an extensive process also funded by the USDoD. Formal requirements were established after comprehensive surveys of user needs and competitive proposals were then submitted resulting in the selection of Intermetrics as the developer under the leadership of Tucker Taft. Then came Ada 2005 and this was developed on a more modest scale. The work was almost entirely done by voluntary effort with support from within the industry itself through bodies such as the Ada Resource Association and Ada-Europe.

After some experience with Ada 2005 it became clear that some further evolution was appropriate. Adding new features as in Ada 2005 always brings some surprises regarding their use and further polishing is almost inevitable. Accordingly, it was decided that a further revision should be made with a goal of completion in 2012.

As in the case of Ada 2005, the development is being performed under the guidance of ISO/IEC JTC1/SC22 WG9 (hereinafter just called WG9). Previously chaired by Jim Moore, it is now under the chairmanship of Joyce Tokar. This committee has included national representatives of many nations including Belgium, Canada, France, Germany, Italy, Japan, Sweden, Switzerland, the UK and the USA. WG9 developed guidelines [1] for a revision to Ada 2005 which were then used by the Ada Rapporteur Group (the ARG) in drafting the revised standard.

The ARG is a team of experts nominated by the national bodies represented on WG9 and the two liaison organizations, ACM SIGAda and Ada-Europe. In the case of Ada 2005, the ARG was originally led by Erhard Plödereder and then by Pascal Leroy. For Ada 2012, it is led by Ed Schonberg. The editor, who at the end of the day actually writes the words of the standard, continues to be the indefatigable Randy Brukardt.

Suggestions for the revised standard have come from a number of sources such as individuals on the ARG, national bodies on WG9, users and implementers via email discussions on Ada-Comment and so on. Also several issues were left over from the development of Ada 2005.

At the time of writing (August 2011), the revision process is approaching completion. The details of all individual changes are now clear and they are being integrated to form a new version of the Annotated Ada Reference Manual. The final approved standard should emerge towards the end of 2012.

2 Scope of revision

The changes from Ada 95 to Ada 2005 were significant (although not so large as the changes from Ada 83 to Ada 95). The main additions were

- in the OO area, multiple inheritance using interfaces and the ability to make calls using prefixed notation,
- more flexible access types with anonymous types, more control over null and constant, and downward closures via access to subprogram types,
- enhanced structure and visibility control by the introduction of limited with and private with clauses and by an extended form of return statement,
- in the real-time area, the Ravenscar profile [2], various new scheduling polices, timers and execution time budget control,
- some minor improvements to exception handling, numerics (especially fixed point) and some further pragmas such as Assert,
- various extensions to the standard library such as the introduction of operations on vectors and matrices, further operations on times and dates, and operations on wide wide characters; and especially:
- a comprehensive library for the manipulation of containers of various kinds.

The changes from Ada 2005 to Ada 2012 were intended to be relatively modest and largely to lead on from the experience of the additions introduced in Ada 2005. But

one thing led to another and in fact the changes are of a similar order to those from Ada 95 to Ada 2005.

From the point of view of the ISO standard, Ada 2005 is the Ada 95 standard modified by two documents. First there was a Corrigendum issued in 2001 [3] and then an Amendment issued in 2005 [4]. In principle the poor user thus has to study these three documents in parallel to understand Ada 2005. However, they were informally incorporated into the Ada 2005 Reference Manual [5].

In the case of Ada 2012, this process of developing a further formal amendment would then lead to the need to consult four documents and so the intention is that the new Edition will formally be a single Revision.

The scope of this Revision is guided by a document issued by WG9 to the ARG in October 2008 [1]. The essence is that the ARG is requested to pay particular attention to

A Improvements that will maintain or improve Ada's advantages, especially in those user domains where safety and criticality are prime concerns. Within this area it cites improving the use and functionality of containers, the ability to write and enforce contracts for Ada entities (for instance, via preconditions) and the capabilities of Ada on multicore and multithreaded architectures.

B Improvements that will remedy shortcomings in Ada. It cites in particular the safety, use, and functionality of access types and dynamic storage management.

So the ARG is asked to improve both OO and real-time with a strong emphasis on real-time and high integrity features. Moreover, "design by contract" features should be added whereas for the previous amendment they were rejected on the grounds that they would not be static.

The ARG is also asked to consider the following factors in selecting features for inclusion:

- Implementability. Can the feature be implemented at reasonable cost?
- Need. Do users actually need it?
- Language stability. Would it appear disturbing to current users?
- Competition and popularity. Does it help to improve the perception of Ada and make it more competitive?
- Interoperability. Does it ease problems of interfacing with other languages and systems?
- Language consistency. Is it syntactically and semantically consistent with the language's current structure and design philosophy?

As before, an important further statement is that "In order to produce a technically superior result, it is permitted to compromise backwards compatibility when the impact on users is judged to be acceptable." In other words don't be paranoid about compatibility.

Finally, there is a warning about secondary standards. Its essence is don't use secondary standards if you can get the material into the RM itself.

The guidelines conclude with the target schedule. This includes WG9 approval of the scope of the amendment in June 2010 which was achieved and submission to ISO/IEC JTC1 in late 2011.

3 Overview of changes

It would be tedious to give a section by section review of the changes as seen by the Reference Manual language lawyer. Instead, the changes will be presented by areas as seen by the user. There can be considered to be six areas:

1 Introduction of dynamic contracts. These can be seen to lead on from the introduction of the Assert pragma in Ada 2005. New syntax (using **with** again) introduces aspect specifications which enable certain properties of entities to be stated where they are declared rather than later using representation clauses. This is put to good use in introducing pre- and postconditions for subprograms and similar assertions for types and subtypes.

2 More flexible expressions. The introduction of preconditions and so on increases the need for more powerful forms of expressions. Accordingly, if expressions, case expressions, quantified expressions and expression functions are all added. A related change is that membership tests are generalized.

3 Structure and visibility control. Functions are now permitted to have **out** and **in out** parameters, and rules are introduced to minimize the risk of inadvertent dependence on order of evaluation of parameters and other entities such as aggregates. More flexibility is permitted with incomplete types and another form of use clause is introduced. There are minor enhancements to extended return statements.

4 Tasking and real-time improvements. Almost all of the changes are in the Real-Time Systems annex. New packages are added for the control of tasks and budgeting on multiprocessor systems, and the monitoring of time spent in interrupts. There are also additional facilities for non-preemptive dispatching, task barriers and suspension objects.

5 Improvements to other general areas. More flexibility is allowed in the position of labels, pragmas, and null statements. A number of corrections are made to the accessibility rules, improvements are made to conversions of access types, and further control over storage pools is added. The composability of equality is now the same for both tagged and untagged record types.

6 Extensions to the standard library. Variants on the existing container packages are introduced to handle bounded containers more efficiently. Additional containers are added for a simple holder, multiway trees and queues. Moreover, a number of general features

have been added to make containers and other such reusable libraries easier to use. Minor additions cover directories, locale capabilities, string encoding and further operations on wide and wide wide characters.

The reader might feel that the changes are quite extensive but each has an important role to play in making Ada more useful. Indeed the solution of one problem often leads to auxiliary requirements. The desire to introduce stronger description of contracts led to the search for good syntax which led to aspect specifications. And these strengthened the need for more flexible forms of expressions and so on. Other changes were driven by outside considerations such as the multiprocessors and others stem from what now seem to be obvious but minor flaws in Ada 2005.

A number of other changes were rejected as really unnecessary. For example, the author was at one time enthused by a desire for fixed point cyclic types. But it proved foolish without base 60 hardware to match our inheritance of arithmetic in a Babylonian style for angles.

Before looking at the six areas in a little more detail it is perhaps worth saying a few words about compatibility with Ada 2005. The guidelines gave the ARG freedom to be sensible in this area. Of course, the worst incompatibilities are those where a valid program in Ada 2005 continues to be valid in Ada 2012 but does something different. It is believed that serious incompatibilities of this nature will never arise.

However, incompatibilities whereby a valid Ada 2005 program fails to compile in Ada 2012 are tolerable provided they are infrequent. A few such incompatibilities are possible. The most obvious cause is the introduction of one more reserved word, namely **some**, which is used in quantified expressions to match **all**. Thus if an existing Ada 2005 program uses **some** as an identifier then it will need modification. Once again, the introduction of a new category of unreserved keywords was considered but was eventually rejected as confusing.

3.1 Contracts

One of the important issues highlighted by WG9 for the Amendment was the introduction of material for enforcing contracts such as preconditions and postconditions. As a simple example consider a stack with procedures Push and Pop. An obvious precondition for Pop is that the stack must not be empty. If we have a function Is_Empty for testing the state of the stack then a call of Is_Empty would provide the basis for an appropriate precondition.

The question now is to find a good way to associate the expression **not** Is_Empty with the specification of the procedure Pop. Note that it is the specification that matters since it is the specification that provides the essence of the contract between the caller of the procedure Pop and the writer of its body. The contract provided by a traditional Ada subprogram specification is rather weak – essentially it just provides enough information for the compiler to generate the correct code for the calls but says nothing about the semantic behaviour of the associated body.

The traditional way to add information of this kind in Ada is via a pragma giving some kind of aspect clause. However, there were problems with this approach. One is that there is no convenient way to distinguish between several overloaded subprograms and another is that such information is given later on because of interactions with freezing and linear elaboration rules.

Accordingly, it was decided that a radical new approach should be devised and this led to the introduction of aspect specifications which are given with the item to which they relate using the reserved word **with**. So to give the precondition for Pop we augment the specification of Pop by writing

```
procedure Pop(S: in out Stack; X: out Item)
  with Pre => not Is_Empty(S);
```

In a similar way we might give a postcondition as well which might be that the stack is not full. So altogether the specification of a generic package for stacks might be

```
generic
  type Item is private;
package Stacks is
  type Stack is private;

  function Is_Empty(S: Stack) return Boolean;
  function Is_Full(S: Stack) return Boolean;

  procedure Push(S: in out Stack; X: in Item)
   with
     Pre => not Is_Full(S),
     Post => not Is_Empty(S);

  procedure Pop(S: in out Stack; X: out Item)
   with
     Pre => not Is_Empty(S),
     Post => not Is_Full(S);

private
   ...
end Stacks;
```

Note how the individual aspects Pre and Post take the form of

```
aspect_mark => expression
```

and that if there are several then they are separated by commas. The final semicolon is of course the semicolon at the end of the subprogram declaration as a whole. Thus the overall syntax is now

```
subprogram_declaration ::=
  [overriding_indicator]
  subprogram_specification
  [aspect_specification] ;
```

and in general

```
aspect_specification ::=
  with aspect_mark [ => expression] { ,
     aspect_mark [ => expression] }
```

Pre- and postconditions are controlled by the same mechanism as assertions using the pragma Assert. It will be

recalled that these can be switched on and off by the pragma Assertion_Policy. Thus if we write

```
pragma Assertion_Policy(Check);
```

then assertions are enabled whereas if the parameter of the pragma is Ignore then all assertions are ignored.

In the case of a precondition, whenever a subprogram with a precondition is called, if the policy is Check then the precondition is evaluated and if it is False then Assertion_Error is raised and the subprogram is not entered. Similarly, on return from a subprogram with a postcondition, if the policy is Check then the postcondition is evaluated and if it is False then Assertion_Error is raised.

So if the policy is Check and Pop is called when the stack is empty then Assertion_Error is raised whereas if the policy is Ignore then the predefined exception Constraint_Error would probably be raised (depending upon how the stack had been implemented).

Note that, unlike the pragma Assert, it is not possible to associate a specific message with the raising of Assertion_Error by a pre- or postcondition. The main reason is that it might be confusing with multiple conditions (which can arise with inheritance) and in any event it is expected that the implementation will give adequate information about which condition has been violated.

Note that it is not permitted to give the aspects Pre or Post for a null procedure; this is because all null procedures are meant to be interchangeable.

There are also aspects Pre'Class and Post'Class for use with tagged types (and they can be given with null procedures). The subtle topic of multiple inheritance of pre- and postconditions will be discussed in detail in a later paper.

Two new attributes are useful in postconditions. X'Old denotes the value of X on entry to the subprogram whereas X denotes the value on exit. And in the case of a function F, the value returned by the function can be denoted by F'Result in a postcondition for F.

As a general rule, the new aspect specifications can be used instead of aspect clauses and pragmas for giving information regarding entities such as types and subprograms.

For example rather than

```
type Bit_Vector is array (0 .. 15) of Boolean;
```

followed later by

```
for Bit_Vector'Component_Size use 1;
```

we can more conveniently write

```
type Bit_Vector is array (0 .. 15) of Boolean
  with Component_Size => 1;
```

However, certain aspects such as record representation and enumeration representations cannot be given in this way because of the special syntax involved.

In cases where aspect specifications can now be used, the existing pragmas are mostly considered obsolescent and condemned to Annex J.

It should be noted that pragmas are still preferred for stating properties of program units such as Pure, Preelaborable and so on. However, we now talk about the pure property as being an aspect of a package. It is a general rule that the new aspect specifications are preferred with types and subprograms but pragmas continue to be preferred for program units. Nevertheless, the enthusiast for the new notation could write

```
package Ada_Twin
  with Pure is
end Ada_Twin;
```

which illustrates that in some cases no value is required for the aspect (by default it is True).

A notable curiosity is that Preelaborable_Initialization, although a property of a type, still has to be specified by a pragma (this is because of problems with different views of the type).

Note incidentally that to avoid confusion with some other uses of the reserved word **with**, in the case of aspect specifications **with** is at the beginning of the line.

There are two other new facilities of a contractual nature concerning types and subtypes. One is known as type invariants and these describe properties of a type that remain true and can be relied upon. The other is known as subtype predicates which extend the idea of constraints. The distinction can be confusing at first sight and the following extract from one of the Ada Issues might be helpful.

"Note that type invariants are not the same thing as constraints, as invariants apply to all values of a type, while constraints are generally used to identify a subset of the values of a type. Invariants are only meaningful on private types, where there is a clear boundary (the enclosing package) that separates where the invariant applies (outside) and where it need not be satisfied (inside). In some ways, an invariant is more like the range of values specified when declaring a new integer type, as opposed to the constraint specified when defining an integer subtype. The specified range of an integer type can be violated (to some degree) in the middle of an arithmetic computation, but must be satisfied by the time the value is stored back into an object of the type."

Type invariants are useful if we want to ensure that some relationship between the components of a private type always holds. Thus suppose we have a stack and wish to ensure that no value is placed on the stack equal to an existing value on the stack. We can modify the earlier example to

```
package Stacks is
  type Stack is private
  with
    Type_Invariant => Is_Unduplicated(Stack);
```

```
function Is_Empty(S: Stack) return Boolean;
function Is_Full(S: Stack) return Boolean;
function Is_Unduplicated(S: Stack) return Boolean;

procedure Push(S: in out Stack; X: in Item)
 with
   Pre => not Is_Full(S),
   Post => not Is_Empty(S);

-- and so on
```

The function Is_Unduplicated then has to be written (in the package body as usual) to check that all values of the stack are different.

Note that we have mentioned Is_Unduplicated in the type invariant before its specification. This violates the usual "linear order of elaboration". However, there is a general rule that all aspect specifications are only elaborated when the entity they refer to is frozen. Recall that one of the reasons for the introduction of aspect specifications was to overcome this problem with the existing mechanisms which caused information to become separated from the entities to which it relates.

The invariant on a private type T is checked when the value can be changed from the point of view of the outside user. That is primarily

- after default initialization of an object of type T,
- after a conversion to type T,
- after a call that returns a result of a type T or has an **out** or **in out** or access parameter of type T.

The checks also apply to subprograms with parameters or results whose components are of the type T.

In the case of the stack, the invariant Is_Unduplicated will be checked when we declare a new object of type Stack and each time we call Push and Pop.

Note that any subprograms internal to the package and not visible to the user can do what they like. It is only when a value of the type Stack emerges into the outside world that the invariant is checked.

The type invariant could be given on the full type in the private part rather than on the visible declaration of the private type (but not on both). Thus the user need not know that an invariant applies to the type.

Type invariants, like pre- and postconditions, are controlled by the pragma Assertion_Policy and only checked if the policy is Check. If an invariant fails to be true then Assertion_Error is raised at the appropriate point.

There is also an aspect Type_Invariant'Class for use with tagged types.

The subtype feature of Ada is very valuable and enables the early detection of errors that linger in many programs in other languages and cause disaster later. However, although valuable, the subtype mechanism is somewhat limited. We can only specify a contiguous range of values in the case of integer and enumeration types.

Accordingly, Ada 2012 introduces subtype predicates as an aspect that can be applied to type and subtype declarations. The requirements proved awkward to satisfy with a single feature so in fact there are two aspects: Static_Predicate and Dynamic_Predicate. They both take a Boolean expression and the key difference is that the static predicate is restricted to certain types of expressions so that it can be used in more contexts.

Suppose we are concerned with seasons and that we have a type Month thus

```
type Month is (Jan, Feb, Mar, Apr, May, ..., Nov, Dec);
```

Now suppose we wish to declare subtypes for the seasons. For most people winter is December, January, February. (From the point of view of solstices and equinoxes, winter is from December 21 until March 21 or thereabouts, but March seems to me generally more like spring rather than winter and December feels more like winter than autumn.) So we would like to declare a subtype embracing Dec, Jan and Feb. We cannot do this with a constraint but we can use a static predicate by writing

```
subtype Winter is Month
  with Static_Predicate => Winter in Dec | Jan | Feb;
```

and then we are assured that objects of subtype Winter can only be Dec, Jan or Feb (provided once more that the Assertion_Policy pragma has set the Policy to Check). Note the use of the subtype name (Winter) in the expression where it stands for the current instance of the subtype.

The aspect is checked whenever an object is default initialized, on assignments, on conversions, on parameter passing and so on. If a check fails then Assertion_Error is raised.

The observant reader will note also that the membership test takes a more flexible form in Ada 2012 as explained in the next section.

If we want the expression to be dynamic then we have to use Dynamic_Predicate thus

```
type T is ... ;
function Is_Good(X: T) return Boolean;
subtype Good_T is T
  with Dynamic_Predicate => Is_Good(Good_T):
```

Note that a subtype with predicates cannot be used in some contexts such as index constraints. This is to avoid having arrays with holes and similar nasty things. However, static predicates are allowed in a for loop meaning to try every value. So we could write

```
for M in Winter loop...
```

Beware that the loop uses values for M in the order, Jan, Feb, Dec and not Dec, Jan, Feb as the user might have wanted.

As another example, suppose we wish to specify that an integer is even. We might expect to be able to write

```
subtype Even is Integer
  with Static_Predicate => Even mod 2 = 0;  -- illegal
```

Sadly, this is illegal because the expression in a static predicate is restricted and cannot use some operations such as **mod**. We have to use a dynamic predicate thus

```
subtype Even is Integer
  with Dynamic_Predicate => Even mod 2 = 0;  --OK
```

and then we cannot write

```
for X in Even loop ...
```

but have to spell it out in detail such as

```
for X in Integer loop
  if X mod 2 = 0 then                -- or if X in Even then
   ... -- body of loop
  end if;
end loop;
```

The assurance given by type invariants and subtype predicates can depend upon the object having a sensible initial value. There is a school of thought that giving default initial values (such as zero) is bad since it can obscure flow errors. However, it is strange that Ada does allow default initial values to be given for components of records but not for scalar types or array types. This is rectified in Ada 2012 by aspects Default_Value and Default_Component_Value. We can write

```
type Signal is (Red, Amber, Green)
  with Default_Value => Red;

type Text is new String
  with Default_Component_Value =>
                          Ada.Characters.Latin_1.Space;

subtype Day is Integer range 1 .. 31
  with Default_Value => 1;
```

Note that, unlike default initial values for record components, these have to be static.

Finally, two new attributes are introduced to aid in the writing of preconditions. Sometimes it is necessary to check that two objects do not occupy the same storage in whole or in part. This can be done with two attributes thus

```
X'Has_Same_Storage(Y)
X'Overlaps_Storage(Y)
```

As an example we might have a procedure Exchange and wish to ensure that the parameters do not overlap in any way. We can write

```
procedure Exchange(X, Y: T)
  with Pre => not X'Overlaps_Storage(Y);
```

Attributes are used rather than predefined functions since this enables the semantics to be written in a manner that permits X and Y to be of any type and moreover does not imply that X or Y are read.

3.2 Expressions

Those whose first language was Algol 60 or Algol 68 or who have had the misfortune to dabble in horrid languages such as C will have been surprised that a language of the richness of Ada does not have conditional expressions. Well, the good news is that Ada 2012 has at last introduced conditional expressions which take two forms, if expressions and case expressions.

The reason that Ada did not originally have conditional expressions is probably that there was a strong desire to avoid any confusion between statements and expressions. We know that many errors in C arise because assignments can be used as expressions. But the real problem with C is that it also treats Booleans as integers, and confuses equality and assignment. It is this combination of fluid styles that causes problems. But just introducing conditional expressions does not of itself introduce difficulties if the syntax is clear and unambiguous.

If expressions in Ada 2012 take the form as shown by the following statements:

```
S := (if N > 0 then +1 else 0);

Put(if N = 0 then "none" elsif N = 1 then "one" else
                                                "lots");
```

Note that there is no need for **end if** and indeed it is not permitted. Remember that **end if** is vital for good structuring of if statements because there can be more than one statement in each branch. This does not arise with if expressions so **end if** is unnecessary and moreover would be heavy as a closing bracket. However, there is a rule that an if expression must always be enclosed in parentheses. Thus we cannot write

```
X := if L > 0 then M else N + 1;          -- illegal
```

because there would be confusion between

```
X := (if L > 0 then M else N) + 1;        -- and

X := (if L > 0 then M else (N + 1));
```

The parentheses around N+1 are not necessary in the last line above but added to clarify the point.

However, if the context already provides parentheses then additional ones are unnecessary. Thus an if expression as a single parameter does not need double parentheses.

It is clear that if expressions will have many uses. However, the impetus for providing them in Ada 2012 was stimulated by the introduction of aspects of the form

```
Pre => expression
```

There will be many occasions when preconditions have a conditional form and without if expressions these would have to be wrapped in a function which would be both heavy and obscure. For example suppose a procedure P has two parameters P1 and P2 and that the precondition is that if P1 is positive then P2 must also be positive but if P1 is not positive then there is no restriction on P2. We could express this by writing a function such as

```
function Checkparas(P1, P2: Integer) return Boolean is
begin
  if P1 > 0 then
   return P2 > 0;
  else          -- P1 is not positive
```

```
    return True; -- so don't care about P2
  end if;
end Checkparas;
```

and then we can write

```
procedure P(P1, P2: Integer)
  with Pre => Checkparas(P1, P2);
```

This is truly gruesome. Apart from the effort of having to declare the wretched function Checkparas, the consequence is that the meaning of the precondition can only be determined by looking at the body of Checkparas and that could be miles away, typically in the body of the package containing the declaration of P. This would be a terrible violation of information hiding in reverse; we would be forced to hide something that should be visible.

However, using if expressions we can simply write

```
Pre => (if P1 > 0 then P2 > 0 else True);
```

and this can be abbreviated to

```
Pre => (if P1 > 0 then P2 > 0);
```

because there is a convenient rule that a trailing **else** True can be omitted when the type is a Boolean type. Many will find it much easier to read without **else** True anyway since it is similar to saying P1 > 0 implies P2 > 0. Adding an operation such as implies was considered but rejected as unnecessary.

The precondition could be extended to say that if P1 equals zero then P2 also has to be zero but if P1 is negative then we continue not to care about P2. This would be written thus

```
Pre => (if P1 > 0 then P2 > 0 elsif P1 = 0 then P2 = 0);
```

There are various sensible rules about the types of the various branches in an if expression as expected. Basically, they must all be of the same type or convertible to the same expected type. Thus consider a procedure Do_It taking a parameter of type Float and the call

```
Do_It (if B then X else 3.14);
```

where X is a variable of type Float. Clearly we wish to permit this but the two branches of the if statement are of different types, X is of type Float whereas 3.14 is of type *universal_real*. But a value of type *universal_real* can be implicitly converted to Float which is the type expected by Do_It and so all is well.

There are also rules about accessibility in the case where the various branches are of access types; the details need not concern us in this overview!

The other new form of conditional expression is the case expression and this follows similar rules to the if expression just discussed. Here is an amusing example based on one in the AI which introduces case expressions.

Suppose we are making a fruit salad and add various fruits to a bowl. We need to check that the fruit is in an appropriate state before being added to the bowl. Suppose we have just three fruits given by

```
type Fruit_Kind is (Apple, Banana, Pineapple);
```

then we might have a procedure Add_To_Salad thus

```
procedure Add_To_Salad(Fruit: in Fruit_Type);
```

where Fruit_Type is perhaps a discriminated type thus

```
type Fruit_Type (Kind: Fruit_Kind) is private;
```

In addition there might be functions such as Is_Peeled that interrogate the state of a fruit.

We could then have a precondition that checks that the fruit is in an edible state thus

```
Pre => (if Fruit.Kind = Apple then Is_Crisp(Fruit)
    elsif Fruit.Kind = Banana then Is_Peeled(Fruit)
    elsif Fruit.Kind = Pineapple then Is_Cored(Fruit));
```

(This example is all very well but it has allowed the apple to go in uncored and the pineapple still has its prickly skin.)

Now suppose we decide to add Orange to type Fruit_Kind. The precondition will still work in the sense that the implicit **else** True will allow the orange to pass the precondition unchecked and will go into the fruit salad possibly unpeeled, unripe or mouldy. The trouble is that we have lost the full coverage check which is such a valuable feature of case statements and aggregates in Ada.

We overcome this by using a case expression and writing

```
Pre => (case Fruit.Kind is
            when Apple => Is_Crisp(Fruit),
            when Banana => Is_Peeled(Fruit),
            when Pineapple => Is_Cored(Fruit),
            when Orange => Is_Peeled(Fruit));
```

and of course without the addition of the choice for Orange it would fail to compile.

Note that there is no **end case** just as there is no **end if** in an if expression. Moreover, like the if expression, the case expression must be in parentheses. Similar rules apply regarding the types of the various branches and so on.

Of course, the usual rules of case statements apply and so we might decide not to bother about checking the crispness of the apple but to check alongside the pineapple (another kind of apple!) that it has been cored by writing

```
Pre => (case Fruit.Kind is
          when Apple | Pineapple => Is_Cored(Fruit),
          when Banana | Orange => Is_Peeled(Fruit));
```

We can use **others** as the last choice as expected but this would lose the value of coverage checking. There is no default **when others** => True corresponding to **else** True for if expressions because that would defeat coverage checking completely.

A further new form of expression is the so-called quantified expression. Quantified expressions allow the checking of a boolean expression for a given range of values and will again be found useful in pre- and postconditions. There are two forms using **for all** and **for some**. Note carefully that **some** is a new reserved word.

Suppose we have an integer array type

```
type Atype is array (Integer range <>) of Integer;
```

then we might have a procedure that sets each element of an array of integers equal to its index. Its specification might include a postcondition thus

```
procedure Set_Array(A: out Atype)
 with Post => (for all M in A'Range => A(M) = M);
```

This is saying that for all values of M in A'Range we want the expression A(M) = M to be true. Note how the two parts are separated by =>.

We could devise a function to check that some component of the array has a given value by

```
function Value_Present(A: Atype; X: Integer) return
                                                Boolean
 with Post => Value_Present'Result =
                (for some M in A'Range => A(M) = X);
```

Note the use of Value_Present'Result to denote the result returned by the function Value_Present.

As with conditional expressions, quantified expressions are always enclosed in parentheses.

The evaluation of quantified expressions is as expected. Each value of M is taken in turn (as in a for statement and indeed we could insert **reverse**) and the expression to the right of => then evaluated. In the case of universal quantification (a posh term meaning **for all**) as soon as one value is found to be False then the whole quantified expression is False and no further values are checked; if all values turn out to be True then the quantified expression is True. A similar process applies to existential quantification (that is **for some**) where the roles of True and False are reversed.

Those with a mathematical background will be familiar with the symbols ∀ and ∃ which correspond to **for all** and **for some** respectively. Readers are invited to discuss whether the A is upside down and the E backwards or whether they are both simply rotated.

As a somewhat more elaborate example suppose we have a function that finds the index of the first value of M such that A(M) equals a given value X. This needs a precondition to assert that such a value exists.

```
function Find(A: Atype; X: Integer) return Integer
 with
  Pre => (for some M in A'Range => A(M) = X),
  Post => A(Find'Result) = X and
    (for all M in A'First .. Find'Result–1 => A(M) /= X);
```

Note again the use of Find'Result to denote the result returned by the function Find.

Quantified expressions can be used in any context requiring an expression and are not just for pre- and postconditions. Thus we might test whether an integer N is prime by

```
RN := Integer(Sqrt(Float(N)));
if (for some K in 2 .. RN => N mod K = 0) then
  ...    -- N not prime
```

or we might reverse the test by

```
if (for all K in 2 .. RN => N mod K / = 0) then
  ...    -- N is prime
```

Beware that this is not a recommended technique if N is at all large!

We noted above that a major reason for introducing if expressions and case expressions was to avoid the need to introduce lots of auxiliary functions for contexts such as preconditions. Nevertheless the need still arises from time to time. A feature of existing functions is that the code is in the body and this is not visible in the region of the precondition – information hiding is usually a good thing but here it is a problem. What we need is a localized and visible shorthand for a little function. After much debate, Ada 2012 introduces expression functions which are essentially functions whose visible body comprises a single expression. Thus suppose we have a record type such as

```
type Point is tagged
 record
  X, Y: Float := 0.0;
 end record;
```

and the precondition we want for several subprograms is that a point is not at the origin. Then we could write

```
function Is_At_Origin(P: Point) return Boolean is
 (P.X = 0.0 and P.Y = 0.0);
```

and then

```
procedure Whatever(P: Point; ... )
 with Pre => not P.Is_At_Origin;
```

and so on.

Such a function is known as an expression function; naturally it does not have a distinct body. The expression could be any expression and could include calls of other functions (and not just expression functions). The parameters could be of any mode (see next section).

Expression functions can also be used as a completion. This arises typically if the type is private. In that case we cannot access the components P.X and P.Y in the visible part. However, we don't want to have to put the code in the package body. So we declare a function specification in the visible part in the normal way thus

```
function Is_At_Origin(P: Point) return Boolean;
```

and then an expression function in the private part thus

```
private
 type Point is ...

 function Is_At_Origin(P: Point) return Boolean is
  (P.X = 0.0 and P.Y = 0.0);
```

and the expression function then completes the declaration of Is_At_Origin and no function body is required in the package body.

Observe that we could also use an expression function for a completion in a package body so that rather than writing the body as

```
function Is_At_Origin(P: Point) return Boolean is
begin
  return P.X = 0.0 and P.Y = 0.0;
end Is_At_Origin;
```

we could write an expression function as a sort of shorthand.

Incidentally, in Ada 2012, we can abbreviate a null procedure body in a similar way by writing

```
procedure Nothing(...) is null;
```

as a shorthand for

```
procedure Nothing(...) is
begin
  null;
end Nothing;
```

and this will complete the procedure specification

```
procedure Nothing(...);
```

Another change in this area is that membership tests are now generalized. In previous versions of Ada, membership tests allowed one to see whether a value was in a range or in a subtype, thus we could write either of

```
if D in 1 .. 30 then
```

```
if D in Days_In_Month then
```

but we could not write something like

```
if D in 1 | 3 | 5 | 6 ..10 then
```

This is now rectified and following **in** we can now have one or more of a value, a range, or a subtype or any combination separated by vertical bars. Moreover, they do not have to be static.

A final minor change is that the form qualified expression is now treated as a name rather than as a primary. Remember that a function call is treated as a name and this allows a function call to be used as a prefix. For example suppose F returns an array (or more likely an access to an array) then we can write

```
F(...)(N)
```

and this returns the value of the component with index N. However, suppose the function is overloaded so that this is ambiguous. The normal technique to overcome ambiguity is to use a qualified expression and write T'(F(...)). But in Ada 2005 this is not a name and so cannot be used as a prefix. This means that we typically have to copy the array (or access) and then do the indexing or (really ugly) introduce a dummy type conversion and write T(T'(F(...)))(N). Either way, this is a nuisance and hence the change in Ada 2012.

3.3 Structure and visibility

What will seem to many to be one of the most dramatic changes in Ada 2012 concerns functions. In previous versions of Ada, functions could only have parameters of mode **in**. Ada 2012 permits functions to have parameters of all modes.

There are various purposes of functions. The purest is simply as a means of looking at some state. Examples are the function Is_Empty applying to an object of type Stack. It doesn't change the state of the stack but just reports on some aspect of it. Other pure functions are mathematical ones such as Sqrt. For a given parameter, Sqrt always returns the same value. These functions never have any side effects. At the opposite extreme we could have a function that has no restrictions at all; any mode of parameters permitted, any side effects permitted, just like a general procedure in fact but also with the ability to return some result that can be immediately used in an expression.

An early version of Ada had such features, there were pure functions on the one hand and so-called value-returning procedures on the other. However, there was a desire for simplification and so we ended up with Ada 83 functions.

In a sense this was the worst of all possible worlds. A function can perform any side effects at all, provided they are not made visible to the user by appearing as parameters of mode **in out**! As a consequence, various tricks have been resorted to such as using access types (either directly or indirectly). A good example is the function Random in the Numerics annex. It has a parameter Generator of mode **in** but this does in fact get updated indirectly whenever Random is called. So parameters can change even if they are of mode **in**. Moreover, the situation has encouraged programmers to use access parameters unnecessarily with increased runtime cost and mental obscurity.

Ada 2012 has bitten the bullet and now allows parameters of functions to be of any mode. But note that operators are still restricted to only **in** parameters for obvious reasons.

However, there are risks with functions with side effects whether they are visible or not. This is because Ada does not specify the order in which parameters are evaluated nor the order in which parts of an expression are evaluated. So if we write

```
X := Random(G) + Random(G);
```

we have no idea which call of Random occurs first – not that it matters in this case. Allowing parameters of all modes provides further opportunities for programmers to inadvertently introduce order dependence into their programs.

So, in order to mitigate the problems of order dependence, Ada 2012 has a number of rules to catch the more obvious cases. These rules are all static and are mostly about aliasing. For example, it is illegal to pass the same actual parameter to two formal **in out** parameters – the rules apply to both functions and procedures. Consider

```
procedure Do_It(Double, Triple: in out Integer) is
begin
  Double := Double * 2;
  Triple := Triple * 3;
end Do_It;
```

Now if we write

```
Var: Integer := 2;
...
Do_It(Var, Var);          -- illegal in Ada 2012
```

then Var might become 4 or 6 in Ada 2005 according to the order in which the parameters are copied back.

These rules also apply to any context in which the order is not specified and which involves function calls with **out** or **in out** parameters. Thus an aggregate such as

```
(Var, F(Var))
```

where F has an **in out** parameter is illegal since the order of evaluation of the expressions in an aggregate is undefined and so the value of the first component of the aggregate will depend upon whether it is evaluated before or after F is called.

Full details of the rules need not concern the normal programmer – the compiler will tell you off!

Another change concerning parameters is that it is possible in Ada 2012 to explicitly state that a parameter is to be aliased. Thus we can write

```
procedure P(X: aliased in out T; ...);
```

An aliased parameter is always passed by reference and the accessibility rules are modified accordingly. This facility is used in a revision to the containers which avoids the need for expensive and unnecessary copying of complete elements when they are updated. The details will be given in a later paper.

A major advance in Ada 2005 was the introduction of limited with clauses giving more flexibility to incomplete types. However, experience has revealed a few minor shortcomings.

One problem is that an incomplete type in Ada 2005 cannot be completed by a private type. This prevents the following mutually recursive structure of two types having each other as an access discriminant

```
type T1;
type T2 (X: access T1) is private;
type T1 (X: access T2) is private;  -- OK in Ada 2012
```

The rules in Ada 2012 are changed so that an incomplete type can be completed by any type, including a private type (but not another incomplete type obviously).

Another change concerns the use of incomplete types as parameters. Generally, we do not know whether a parameter of a private type is passed by copy or by reference. The one exception is that if it is tagged then we know it will be passed by reference. As a consequence there is a rule in Ada 2005 that an incomplete type cannot be used as a parameter unless it is tagged incomplete. This has forced the unnecessary use of access parameters.

In Ada 2012, this problem is remedied by permitting incomplete types to be used as parameters (and as function results) provided that they are fully defined at the point of call and where the body is declared.

A final change to incomplete types is that a new category of formal generic parameter is added that allows a generic unit to be instantiated with an incomplete type. Thus rather than having to write a signature package as

```
generic
  type Element is private;
  type Set is private;
  with function Empty return Set;
  with function Unit(E: Element) return Set;
  with function Union(S, T: Set) return Set;
  ...
package Set_Signature is end;
```

which must be instantiated with complete types, we can now write

```
generic
  type Element;
  type Set;
  with function Empty return Set;
  ...
package Set_Signature is end;
```

where the formal parameters Element and Set are categorized as incomplete. Instantiation can now be performed using any type, including incomplete or private types as actual parameters. This permits the cascading of generic packages which was elusive in Ada 2005 and will be explained in detail in a later paper. Note that we can also write **type** Set **is tagged**; which requires the actual parameter to be tagged but still permits it to be incomplete.

There is a change regarding discriminants. In Ada 2005, a discriminant can only have a default value if the type is not tagged. Remember that giving a default value makes a type mutable. But not permitting a default value has proved to be an irritating restriction in the case of limited tagged types. Being limited they cannot be changed anyway and so a default value is not a problem and is permitted in Ada 2012. This feature is used in the declaration of the protected types for synchronized queues in Section 3.6.

Another small but useful improvement is in the area of use clauses. In Ada 83, use clauses only apply to packages and everything in the package specification is made visible. Programming guidelines often prohibit use clauses on the grounds that programs are hard to understand since the origin of entities is obscured. This was a nuisance with operators because it prevented the use of infixed notation and forced the writing of things such as

```
P."+"(X, Y)
```

Accordingly, Ada 95 introduced the use type clause which just makes the operators for a specific type in a package directly visible. Thus we write

```
use type P.T;
```

However, although this makes the primitive operators of T visible it does not make everything relating to T visible. Thus it does not make enumeration literals visible or other primitive operations of the type such as subprograms. This is a big nuisance.

To overcome this, Ada 2012 introduces a further variation on the use type clause. If we write

```
use all type P.T;
```

then *all* primitive operations of T are made visible (and not just primitive operators) and this includes enumeration literals in the case of an enumeration type and class wide operations of tagged types.

Finally, there are a couple of small changes to extended return statements which are really corrections to amend oversights in Ada 2005.

The first is that a return object can be declared as **constant**. For example

```
function F(...) return LT is
...
  return Result: constant LT := ... do
   ....
  end return;
end F;
```

We allow everything else to be declared as **constant** so we should here as well especially if LT is a limited type. This was really an oversight in the syntax.

The other change concerns class wide types. If the returned type is class wide then the object declared in the extended return statement need not be the same in Ada 2012 provided it can be converted to the class wide type.

Thus

```
function F(...) return T'Class is
...
  return X: TT do
  ...
  end return;
end F;
```

is legal in Ada 2012 provided that TT is descended from T and thus covered by T'Class. In Ada 2005 it is required that the result type be identical to the return type and this is a nuisance with a class wide type because it then has to be initialized with something and so on. Note the analogy with constraints. The return type might be unconstrained such as String whereas the result (sub)type can be constrained such as String(1 .. 5).

3.4 Tasking and real-time facilities

There are a number of improvements regarding scheduling and dispatching in the Real-Time Systems annex.

A small addition concerns non-preemptive dispatching. In Ada 2005, a task wishing to indicate that it is willing to be preempted has to execute

```
delay 0.0;
```

(or **delay until** Ada.Real_Time.Time_First in Ravenscar). This is ugly and so a procedure Yield is added to the package Ada.Dispatching.

A further addition is the ability to indicate that a task is willing to be preempted by a task of higher priority (but not the same priority). This is done by calling Yield_To_Higher which is declared in a new child package with specification

```
package Ada.Dispatching.Non_Preemptive is
  pragma Preelaborate(Non_Preemptive);
  procedure Yield_To_Higher;
  procedure Yield_To_Same_Or_Higher renames Yield;
end Ada.Dispatching.Non_Preemptive;
```

Another low-level scheduling capability concerns suspension objects; these were introduced in Ada 95. Recall that we can declare an object of type Suspension_Object and call procedures to set it True or False. By calling Suspend_Until_True a task can suspend itself until the state of the object is true.

Ada 2005 introduced Earliest Deadline First (EDF) scheduling. The key feature here is that tasks are scheduled according to deadlines and not by priorities. A new facility introduced in Ada 2012 is the ability to suspend until a suspension object is true and then set its deadline sometime in the future. This is done by calling the aptly named procedure Suspend_Until_True_And_Set_Deadline in a new child package Ada.Synchronous_Task_Control.EDF.

A new scheduling feature is the introduction of synchronous barriers in a new child package Ada.Synchronous_Barriers. The main features are a type Synchronous_Barrier with a discriminant giving the number of tasks to be waited for.

```
type Synchronous_Barrier(Release_Threshold:
                         Barrier_Limit) is limited private;
```

There is also a procedure

```
procedure Wait_For_Release(
                    The_Barrier: in out Synchronous_Barrier;
                    Notified: out Boolean);
```

When a task calls Wait_For_Release it gets suspended until the number waiting equals the discriminant. All the tasks are then released and just one of them is told about it by the parameter Notified being True. The general idea is that this one task then does something on behalf of all the others. The count of tasks waiting is then reset to zero so that the synchronous barrier can be used again.

A number of other changes in this area are about the use of multiprocessors and again concern the Real-Time Systems annex.

A new package System.Multiprocessors is introduced as follows

```
package System.Multiprocessors is
  pragma Preelaborate(Multiprocessors);
  type CPU_Range is range 0..implementation-defined;
```

```
   Not_A_Specific_CPU: constant CPU_Range := 0:
   subtype CPU is CPU_Range
                          range 1 .. CPU_Range'Last;
   function Number_Of_CPUs return CPU;
end System.Multiprocessors;
```

A value of subtype CPU denotes a specific processor. Zero indicates don't know or perhaps don't care. The total number of CPUs is determined by calling the function Number_Of_CPUs. This is a function rather than a constant because there could be several partitions with a different number of CPUs on each partition.

Tasks can be allocated to processors by an aspect specification. If we write

```
task My_Task is
  with CPU => 10;
```

then My_Task will be executed by processor number 10. In the case of a task type then all tasks of that type will be executed by the given processor. The expression giving the processor for a task can be dynamic. The aspect can also be set by a corresponding pragma CPU. (This is an example of a pragma born obsolescent.) The aspect CPU can also be given to the main subprogram in which case the expression must be static.

Further facilities are provided by the child package System.Multiprocessors.Dispatching_Domains. The idea is that processors are grouped together into dispatching domains. A task may then be allocated to a domain and it will be executed on one of the processors of that domain.

Domains are of a type Dispatching_Domain. They are created by a function Create

```
function Create(First, Last: CPU) return
                                   Dispatching_Domain;
```

that takes the first and last numbered CPU of the domain and then returns the domain. All CPUs are initially in the System_Dispatching_Domain. If we attempt to do something silly such as create overlapping domains, then Dispatching_Domain_Error is raised.

Tasks can be assigned to a domain in two ways. One way is to use an aspect

```
task My_Task
  with Dispatching_Domain => My_Domain;
```

The other way is by calling the procedure Assign_Task whose specification is

```
procedure Assign_Task(
        Domain: in out Dispatching_Domain;
        CPU: in CPU_Range := Not_A_Specific_CPU;
        T: in Task_Id := Current_Task);
```

There are a number of other subprograms for manipulating domains and CPUs. An interesting one is Delay_Until_And_Set_CPU which delays the calling task until a given real time and then sets the processor.

The Ravenscar profile is now defined to be permissible with multiprocessors. However, there is a restriction that tasks may not change CPU. Accordingly the definition of the profile now includes the following restriction

```
No_Dependence =>
        System.Multiprocessors.Dispatching_Domains
```

In order to clarify the use of multiprocessors with group budgets the package Ada.Execution_Time.Group_Budgets introduced in Ada 2005 is slightly modified. The type Group_Budget (which is currently just **tagged limited private**) has a discriminant in Ada 2012 giving the CPU thus

```
type Group_Budget(
                CPU: System.Multiprocessors.CPU :=
                  System.Multiprocessors.CPU'First)
                          is tagged limited private;
```

This means that a group budget only applies to a single processor. If a task in a group is executed on another processor then the budget is not consumed. Note that the default value for CPU is CPU'First which is always 1.

Another improvement relating to times and budgets concerns interrupts. Two Boolean constants are added to the package Ada.Execution_Time

```
Interrupt_Clocks_Supported:
        constant Boolean := implementation-defined;
Separate_Interrupt_Clocks_Supported:
        constant Boolean := implementation-defined;
```

The constant Interrupt_Clocks_Supported indicates whether the time spent in interrupts is accounted for separately from the tasks and then Separate_Interrupt_Clocks_Supported indicates whether it is accounted for each interrupt individually. There is also a function

```
function Clocks_For_Interrupts return CPU_Time;
```

This function gives the time used over all interrupts. Calling it if Interrupt_Clocks_Supported is false raises Program_Error.

A new child package accounts for the interrupts separately if Separate_Interrupt_Clocks_Supported is true.

```
package Ada.Execution_Time.Interrupts is
  function Clock(Interrupt: Ada.Interrupts.Interrupt_Id)
                                        return CPU_Time;
  function Supported(
                Interrupt: Ada.Interrupts.Interrupt_Id)
                                         return Boolean;
end Ada.Execution_Time.Interrupts;
```

The function Supported indicates whether the time for a particular interrupt is being monitored. If it is then Clock returns the accumulated time spent in that interrupt handler (otherwise it returns zero). However, if the overall constant Separate_Interrupt_Clocks_Supported is false then calling Clock for a particular interrupt raises Program_Error.

Multiprocessors have an impact on shared variables. The existing pragma Volatile (now the aspect Volatile) requires access to be in memory but this is strictly unnecessary. All we need is to ensure that reads and writes occur in the right

order. A new aspect Coherent was considered but was rejected in favour of simply changing the definition of Volatile.

The final improvement in this section is in the core language and concerns synchronized interfaces and requeue. The procedures of a synchronized interface may be implemented by a procedure or entry or by a protected procedure. However, in Ada 2005 it is not possible to requeue on a procedure of a synchronized interface even if it is implemented by an entry. This is a nuisance and prevents certain high level abstractions.

Accordingly, Ada 2012 has an aspect Synchronization that takes one of By_Entry, By_Protected_Procedure, and Optional. So we might write

```
type Server is synchronized interface;
procedure Q(S: in out Server; X: in Item);
  with Synchronization => By_Entry;
```

and then we are assured that we are permitted to perform a requeue on any implementation of Q.

As expected there are a number of consistency rules. The aspect can also be applied to a task interface or to a protected interface. But for a task interface it obviously cannot be By_Protected_Procedure.

In the case of inheritance, any Synchronization property is inherited. Naturally, multiple aspect specifications must be consistent. Thus Optional can be overridden by By_Entry or by By_Protected_Procedure but other combinations conflict and so are forbidden.

A related change is that if an entry is renamed as a procedure then we can do a requeue using the procedure name. This was not allowed in Ada 95 or Ada 2005.

3.5 General improvements

As well as the major features discussed above there are also a number of improvements in various other areas.

We start with some gentle stuff. Ada 95 introduced the package Ada thus

```
package Ada is
  pragma Pure(Ada);
end Ada;
```

However, a close reading of the RM revealed that poor Ada is illegal since the pragma Pure is not in one of the allowed places for a pragma. Pragmas are allowed in the places where certain categories are allowed but not *in place of them*. In the case of a package specification the constructs are basic declarative items, but "items" were not one of the allowed things. This has been changed to keep Ada legal.

A related change concerns a sequence of statements. In a construction such as

```
if B then
  This;
else
  That;
end if;
```

there must be at least one statement in each branch so if we don't want any statements then we have to put a null statement. If we want a branch that is just a pragma Assert then we have to put a null statement as well thus

```
if B then
  pragma Assert(...); null;
end if;
```

This is really irritating and so the rules have been changed to permit a pragma in place of a statement in a sequence of statements. This and the problem with the package Ada are treated as Binding Interpretations which means that they apply to Ada 2005 as well.

A similar change concerns the position of labels. It is said that gotos are bad for you. However, gotos are useful for quitting an execution of a loop and going to the end in order to try the next iteration. Thus

```
for I in ... loop
  ...
    if this-one-no-good then goto End_Of_Loop; end if;
  ...
<<End_Of_Loop>> null;      -- try another iteration
end loop;
```

Ada provides no convenient way of doing this other than by using a goto statement. Remember that **exit** transfers control out of the loop. The possibility of a continue statement as in some other languages was discussed but it was concluded that this would obscure the transfer of control. The great thing about **goto** is that the label sticks out like a sore thumb. Indeed, a survey of the code in a well known compiler revealed an orgy of uses of this handy construction.

However, it was decided that having to put **null** was an ugly nuisance and so the syntax of Ada 2012 has been changed to permit the label to come right at the end.

There is a significant extension to the syntax of loops used for iteration. This arose out of a requirement to make iteration over containers easier (and this will be described in a later paper) but the general ideas can be illustrated with an array. Consider

```
for K in Table'Range loop
  Table(K) := Table(K) + 1;
end loop;
```

This can now be written as

```
for T of Table loop
  T := T +1;
end loop;
```

The entity T is a sort of generalized reference and hides the indexing. This mechanism can also be used with multidimensional arrays in which case just one loop replaces a nested set of loops.

A minor problem has arisen with the use of tags and Generic_Dispatching_Constructor. There is no way of discovering whether a tag represents an abstract type other than by attempting to create an object of the type which

then raises Tag_Error; this is disgusting. Accordingly, a new function

```
function Is_Abstract(T: Tag) return Boolean;
```

is added to the package Ada.Tags.

There were many changes to access types in Ada 2005 including the wide introduction of anonymous access types. Inevitably some problems have arisen.

The first problem is with the accessibility of stand-alone objects of anonymous access types such as

```
A: access T;
```

Without going into details, it turns out that such objects are not very useful unless they carry the accessibility level of their value in much the same way that access parameters carry the accessibility level of the actual parameter. They are therefore modified to do this.

Programmers have always moaned about the need for many explicit conversions in Ada. Accordingly, implicit conversions from anonymous access types to named access types are now permitted provided the explicit conversion is legal. The idea is that the need for an explicit conversion with access types should only arise if the conversion could fail. A curious consequence of this change is that a preference rule is needed for the equality of anonymous access types.

An issue regarding allocators concerns their alignment. It will be recalled that when implementing a storage pool, the attribute Max_Size_In_Storage_Units is useful since it indicates the maximum size that could be requested by a call of Allocate. Similarly, the new attribute Max_Alignment_For_Allocation indicates the maximum alignment that could be requested.

Another problem is that allocators for anonymous access types cause difficulties in some areas. Rather than forbidding them completely a new restriction identifier is added so that we can write

```
pragma Restrictions(No_Anonymous_Allocators);
```

Another new restriction is

```
pragma Restrictions(No_Standard_Allocators_
                    After_Elaboration);
```

This can be used to ensure that once the main subprogram has started no further allocation from standard storage pools is permitted. This prevents a long lived program suffering from rampant heap growth.

However, this does not prevent allocation from user-defined storage pools. To enable users to monitor such allocation, additional functions are provided in Ada.Task_Identification, namely Environment_Task (returns the Task_Id of the environment task) and Activation_Is_Complete (returns a Boolean result indicating whether a particular task has finished activation).

A new facility is the ability to define subpools using a new package System.Storage_Pools.Subpools. A subpool is a separately reclaimable part of a storage pool and is identified by a subpool handle name. On allocation, a handle name can be given.

Further control over the use of storage pools is provided by the ability to define a default storage pool. Thus we can write

```
pragma Default_Storage_Pool(My_Pool);
```

and then all allocation within the scope of the pragma will be from My_Pool unless a different specific pool is given for a type. This could be done using the aspect Storage_Pool as expected

```
type Cell_Ptr is access Cell
  with Storage_Pool => Cell_Ptr_Pool;
```

A pragma Default_Storage_Pool can be overridden by another one so that for example all allocation in a package (and its children) is from another pool. The default pool can be specified as **null** thus

```
pragma Default_Storage_Pool(null);
```

and this prevents any allocation from standard pools.

Note that coextensions and allocators as access parameters may nevertheless be allocated on the stack. This can be prevented (somewhat brutally) by the following restrictions

```
pragma Restrictions(No_Coextensions);
```

```
pragma Restrictions(No_Access_Parameter_Allocators);
```

A number of other restrictions have also been added. The introduction of aspects logically requires some new restrictions to control their use. Thus by analogy with No_Implementation_Pragmas, we can write

```
pragma Restrictions(No_Implementation_Aspect_
                                Specifications);
```

and this prevents the use of any implementation-defined aspect specifications. The use of individual aspects such as Default_Value can be prevented by

```
pragma Restrictions(No_Specification_of_Aspect =>
                                Default_Value);
```

Implementations and indeed users are permitted to add descendant units of Ada, System and Interfaces such as another child of Ada.Containers. This can be confusing for maintainers since they may be not aware that they are using non-standard packages. The new restriction

```
pragma Restrictions(No_Implementation_Units);
```

prevents the use of such units.

In a similar vein, there is also

```
pragma Restrictions(No_Implementation_Identifiers);
```

and this prevents the use of additional identifiers declared in packages such as System.

A blanket restriction can be imposed by writing

```
pragma Profile(No_Implementation_Extensions);
```

and this is equivalent to the following five restrictions

```
No_Implementation_Aspect_Specifications,
No_Implementation_Attributes,
No_Implementation_Identifiers,
No_Implementation_Pragmas,
No_Implementation_Units.
```

Finally, the issue of composability of equality has been revisited. In Ada 2005, tagged record types compose but untagged record types do not. If we define a new type (a record type, array type or a derived type) then equality is defined in terms of equality for its various components. However, the behaviour of components which are records is different in Ada 2005 according to whether they are tagged or not. If a component is tagged then the primitive operation is used (which might have been redefined), whereas for an untagged type, predefined equality is used even though it might have been overridden. This is a bit surprising and so has been changed in Ada 2012 so that all record types behave the same way and use the primitive operation. This is often called composability of equality so that we can say that in Ada 2012, record types always compose for equality. Remember that this only applies to records; components which are of array types and elementary types always use predefined equality.

3.6 Standard library

The main improvements in the standard library concern containers. But there are a number of other changes which will be described first.

In Ada 2005, additional versions of Index and Index_Non_Blank were added to the package Ada.Strings.Fixed with an additional parameter From indicating the start of the search. The same should have been done for Find_Token. So Ada 2012 adds

```
procedure Find_Token(Source: in String;
                     Set: in Maps.Character_Set;
                     From: in Positive;
                     Test: in Membership;
                     First: out Positive;
                     Last: out Natural);
```

Similar versions are added for bounded and unbounded strings to the corresponding packages.

New child packages of Ada.Strings are added to provide conversions between strings, wide strings, or wide wide strings and UTF8 or UTF16 encodings. They are

Ada.Strings.UTF_Encoding – declares a function Encoding to convert a String into types UTF_8, UTF_16BE, or UTF_16LE where BE and LE denote Big Endian and Little Endian respectively.

Ada.Strings.UTF_Encoding.Conversions – declares five functions Convert between the UTF schemes.

Ada.Strings.UTF_Encoding.Strings – declares functions Encode and Decode between the type String and the UTF schemes.

Ada.Strings.UTF_Encoding.Wide_Strings – declares six similar functions for the type Wide_String.

Ada.Strings.UTF_Encoding.Wide_Wide_Strings – declares six similar functions for the type Wide_Wide_String.

Further new packages are Ada.Wide_Characters.Handling and Ada.Wide_Wide_Characters.Handling. These provide classification functions such as Is_Letter and Is_Lower and conversion functions such as To_Lower for the types Wide_Character and Wide_Wide_Character in a similar way to the existing package Ada.Characters.Handling for the type Character.

Experience with the package Ada.Directories added in Ada 2005 has revealed a few shortcomings.

One problem concerns case sensitivity. Unfortunately, common operating systems differ in their approach. To remedy this the following are added to Ada.Directories

```
type Name_Case_Kind is (Unknown, Case_Sensitive,
                        Case_Insensitive, Case_Preserving);

function Name_Case_Equivalence(Name: in String)
                               return Name_Case_Kind;
```

Calling Name_Case_Equivalence enables one to discover the situation for the operating system concerned.

Another problem is that the basic approach in Ada.Directories is a bit simplistic and assumes that file names can always be subdivided into a directory name and a simple name. Thus the existing function Compose is

```
function Compose(Containing_Directory: String := "";
                 Name: String;
                 Extension: String := "") return String;
```

and this requires that the Name is a simple name such as "My_File" with possibly an extension if one is not provided.

Accordingly, an optional child package is introduced, Ada.Directories.Hierarchical_File_Names, and this adds the concept of relative names and a new version of Compose whose second parameter is a relative name and various functions such as Is_Simple_Name and Is_Relative_Name.

Programs often need information about where they are being used. This is commonly called the Locale. As an example, in some regions of the world, a sum such as a million dollars is written as $1,000,000.00 whereas in others it appears as $1.000.000,00 with point and comma interchanged. An early attempt at providing facilities for doing the right thing was fraught with complexity. So Ada 2012 has adopted the simple solution of enabling a program to determine the country code (two characters) and the language code (three characters) and then do its own thing. The codes are given by ISO standards. Canada is interesting in that it has one country code ("CA") but uses two language codes ("eng" and "fra").

The information is provided by a new package Ada.Locales which declares the codes and the two functions Language and Country to return the current active locale (that is, the locale associated with the current task).

And finally, we consider the container library. Containers were a major and very valuable addition to Ada 2005 but again, experience with use has indicated that some enhancements are necessary.

We start with a brief summary of what is in Ada 2005. The parent package Ada.Containers has six main children namely Vectors, Doubly_Linked_Lists, Hashed_Maps, Ordered_Maps, Hashed_Sets, and Ordered_Sets. These manipulate definite types.

In addition there are another six for manipulating indefinite types with names such as Indefinite_Vectors and so on.

There are also two packages for sorting generic arrays, one for unconstrained types and one for constrained types.

There are four new kinds of containers in Ada 2012

- bounded forms of the existing containers,
- a container for a single indefinite object,
- a container for multiway trees, and
- a number of containers for queues.

In addition there are a number of auxiliary new facilities whose purpose is to simplify the use of containers.

We will start by briefly looking at each of the new kinds of containers in turn.

The existing containers are unbounded in the sense that there is no limit to the number of items that can be added to a list for example. The implementation is expected to use storage pools as necessary. However, many applications in high integrity and real-time areas forbid the use of access types and require a much more conservative approach. Accordingly, a range of containers is introduced with bounded capacity so that there is no need to acquire extra storage dynamically.

Thus there are additional packages with names such as Containers.Bounded_Doubly_Linked_Lists. A key thing is that the types List, Vector and so on all have a discriminant giving their capacity thus

```
type List(Capacity: Count_Type) is tagged private;
```

so that when a container is declared its capacity is fixed. A number of consequential changes are made as well. For example, the bounded form has to have a procedure Assign

```
procedure Assign(Target: in out List; Source: in List);
```

because using built-in assignment would raise Constraint_Error if the capacities were different. Using a procedure Assign means that the assignment will work provided the length of the source is not greater than the capacity of the target. If it is, the new exception Capacity_Error is raised.

Moreover, a similar procedure Assign is added to all existing unbounded containers so that converting from a bounded to an unbounded container or vice versa is (reasonably) straightforward.

Conversion between bounded and unbounded containers is also guaranteed with respect to streaming.

There are no bounded indefinite containers; this is because if the components are indefinite then dynamic space allocation is required for the components anyway and making the overall container bounded would be pointless.

In Ada, it is not possible to declare an object of an indefinite type that can hold any value of the type. Thus if we declare an object of type String then it becomes constrained by the mandatory initial value.

```
S: String := "Crocodile";
```

We can assign other strings to S but they must also have nine characters. We could assign "Alligator" but not "Elephant". (An elephant is clearly too small!)

This rigidity is rather a nuisance and so a new form of container is defined which enables the cunning declaration of an object of a definite type that can hold a single value of an indefinite type. In other words it is a wrapper. The new package is Ada.Containers.Indefinite_Holders and it takes a generic parameter of the indefinite type and declares a definite type Holder which is tagged private thus

```
generic
  type Element_Type (<>) is private;
  with function "="(Left, Right: Element_Type)
                                  return Boolean is <>;
package Ada.Containers.Indefinite_Holders is
  type Holder is tagged private;
  ...                -- various operations
end Ada.Containers. Indefinite_Holders;
```

The various operations include a procedure Replace_Element which puts a value into the holder and a function Element which returns the current value in the holder.

Another new container is one for multiway trees. It might have been thought that it would be easy to do this using the existing containers such as the list container. But it is difficult for various reasons concerning memory management. And so it was concluded that a new container for multiway trees should be added to Ada 2012.

The package Ada.Containers.Multiway_Trees has all the operations required to operate on a tree structure where each node can have multiple child nodes to any depth. Thus there are operations on subtrees, the ability to find siblings, to insert and remove children and so on.

Finally, there is a group of containers for queues. This topic is particularly interesting because it has its origins in the desire to provide container operations that are task safe. However, it turned out that it was not easy to make the existing containers task safe in a general way which would satisfy all users because there are so many possibilities.

However, there was no existing container for queues and in the case of queues it is easy to see how to make them task safe.

There are in fact four queue containers and all apply to queues where the element type is definite; these come in both bounded and unbounded forms and for synchronized and priority queues. We get (writing AC as an abbreviation for Ada.Containers)

- AC.Unbounded_Synchronized_Queues,
- AC.Bounded_Synchronized_Queues,
- AC.Unbounded_Priority_Queues,
- AC.Bounded_Priority_Queues.

These in turn are all derived from a single synchronized interface. This is a good illustration of the use of synchronized interfaces and especially the aspect Synchronization discussed earlier (see Section 3.4). First there is the following generic package which declares the type Queue as a synchronized interface (writing AC as an abbreviation for Ada.Containers and ET for Element_Type)

```
generic
  type ET is private; -- element type for definite queues
package AC.Synchronized_Queue_Interfaces is
  pragma Pure(...);
  type Queue is synchronized interface;

  procedure Enqueue(Container: in out Queue;
                    New_Item: in ET) is abstract
      with Synchronization => By_Entry;

  procedure Dequeue(Container: in out Queue;
                    Element: out ET) is abstract
      with Synchronization => By_Entry;

  function Current_Use(Container: Queue)
                    return Count_Type is abstract;
  function Peak_Use(Container: Queue)
                    return Count_Type is abstract;
end AC.Synchronized_Queue_Interfaces;
```

Then there are generic packages which enable us to declare actual queues. Thus the essence of the unbounded synchronized version is as follows (still with abbreviations AC for Ada.Containers, ET for Element_Type)

```
with System; use System;
with AC.Synchronized_Queue_Interfaces;
generic
  with package Queue_Interfaces is new
              AC.Synchronized_Queue_Interfaces(<>);
  Default_Ceiling: Any_Priority := Priority'Last;
package AC.Unbounded_Synchronized_Queues is
  pragma Preelaborate(...);

  package Implementation is
    -- not specified by the language
  end Implementation;

  protected type Queue(Ceiling: Any_Priority :=
                                  Default_Ceiling)
              with Priority => Ceiling
                is new Queue_Interfaces.Queue with
    overriding
    entry Enqueue(New_Item: in Queue_Interfaces.ET)
    overriding
    entry Dequeue(Element: out Queue_Interfaces.ET);

    overriding
    function Current_Use return Count_Type;
    overriding
    function Peak_Use return Count_Type;

  private
    ...
  end Queue;

private
  ...
end AC.Unbounded_Synchronized_Queues;
```

The discriminant gives the ceiling priority and for convenience has a default value. Remember that a protected type is limited and when used to implement an interface (as here) is considered to be tagged. In Ada 2012, defaults are allowed for discriminants of tagged types provided they are limited as mentioned in Section 3.3.

Note that the Priority is given by an aspect specification. Programmers who are allergic to the multiple uses of **with** could of course use the old pragma Priority in their own code.

The need for the package Implementation will be explained in a later paper. However, this need not bother the user because the above text is all part of Ada 2012. Now to declare our own queue of integers say we first write

```
package My_Interface is new
  AC.Synchronized_Queue_Interfaces(ET => Integer);
```

This creates an interface for dealing with integers. Then to obtain an unbounded queue package for integers we write

```
package My_Q_Package is new
  AC.Unbounded_Synchronized_Queues(My_Interface);
```

This creates a package which declares a protected type Queue. Now at last we can declare an object of this type and perform operations on it.

```
The_Queue: My_Q_Package.Queue;
...
The_Queue.Enqueue(37);
```

The various calls of Enqueue and Dequeue are likely to be in different tasks and the protected object ensures that all is well.

The other generic queue packages follow a similar style. Note that unlike the other containers, there are no queue packages for indefinite types. Indefinite types can be catered for by using the holder container as a wrapper or by using an access type.

In Ada 2005 there are two generic procedures for sorting arrays; one is for constrained arrays and one is for

unconstrained arrays. In Ada 2012, a third generic procedure is added which can be used to sort any indexable structure. Its specification is

```
generic
  type Index_Type is (<>);
  with function Before(Left, Right: Index_Type)
                                        return Boolean;
  with procedure Swap(Left, Right: Index_Type);
procedure Ada.Containers.Generic_Sort
                              (First, Last: Index_Type'Base);
pragma Pure(Ada.Containers.Generic_Sort);
```

Note that there is no parameter indicating the structure to be sorted; this is all done indirectly by the subprograms Before and Swap working over the range of values given by First and Last. It's almost magic!

A frequent requirement when dealing with containers is the need to visit every node and perform some action, in other words to iterate over the container. And there are probably many different iterations to be performed. In Ada 2005, this has to be done by the user defining a subprogram for each iteration or writing out detailed loops involving calling Next and checking for the last element of the container and so on. And we have to write out this mechanism for each such iteration.

In Ada 2012, after some preparatory work involving the new package Ada.Iterator.Interfaces it is possible to simplify such iterations hugely. For example, suppose we have a list container each of whose elements is a record containing two components of type Integer (P and Q say) and we want to add some global X to Q for all elements where P is a prime. In Ada 2005 we have to write the laborious

```
C := The_List.First;       -- C declared as of type Cursor
loop
  exit when C = No_Element;
  E := Element(C);
  if Is_Prime(E.P) then
   Replace_Element(C, (E.P, E.Q + X));
  end if;
  C := Next(C);
end loop;
```

Not only is this tedious but there is lots of scope for errors. However, in Ada 2012 we can simply write

```
for E of The_List loop
  if Is_Prime(E.P) then E.Q := E.Q + X; end if;
end loop;
```

The mechanism is thus similar to that introduced in the previous section for arrays.

There are also a number of minor new facilities designed to simplify the use of containers. These include the introduction of case insensitive operations for comparing strings and for writing hash functions.

4 Conclusions

This overview of Ada 2012 should have given the reader an appreciation of the important new features in Ada 2012. Some quite promising features failed to be included partly because the need for them was not clear and also because a conclusive design proved elusive.

Further papers will expand on the six major topics of this overview in more detail.

It is worth briefly reviewing the guidelines (see Section 2 above) to see whether Ada 2012 meets them.

The group A items were about extending the advantages of Ada and specifically mentioned containers, contracts and real-time. There are many new features for containers, pre- and postconditions have been added and so have facilities for multiprocessors.

The group B items were about eliminating shortcomings, increasing safety and particularly mentioned improvements to access types and storage management. This has been achieved with corrections to accessibility checks, the introduction of subpools and so on.

It seems clear from this brief check that indeed Ada 2012 does meet the objectives set for it.

Finally, I need to thank all those who have helped in the preparation of this paper and especially Randy Brukardt, Ed Schonberg and Tucker Taft.

References

[1] ISO/IEC JTC1/SC22/WG9 N498 (2009) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of Amendment 2 to ISO/IEC 8652*.

[2] ISO/IEC TR 24718:2004 *Guide for the use of the Ada Ravenscar profile in high integrity systems.*

[3] ISO/IEC 8652:1995/COR 1:2001, *Ada Reference Manual – Technical Corrigendum 1.*

[4] ISO/IEC 8652:1995/AMD 1:2007, *Ada Reference Manual – Amendment 1*.

[5] S. T Taft et al (eds) (2007) *Ada 2005 Reference Manual*, LNCS 4348, Springer-Verlag.

The Implementation of High Integrity Data Structures

Phil Thornley
SPARKSure, 32 Purbeck Drive, Bolton, BL6 4JF, UK.; Tel: +44 1204 695923; email: phil@sparksure.com

Abstract

An initial attempt to prove the correctness of a simple linked data structure written in SPARK failed when the proof artefacts became unmanageable. This paper discusses the reasons for the failure and describes an alternative approach that has succeeded in completing the proofs. Finally a broader conclusion about the place of proof within the software process is suggested.

Keywords: SPARK, proof of correctness, reachability, linked data structure, data structure invariant, Ada.

1 Introduction

SPARK implementations of simple data structures (stack, queue) have simple proofs, but for even the simplest linked data structure, it has been found that the proofs can become very elaborate and unmanageable.

The natural approach to the proofs is to define an abstract data structure invariant for the public view and a detailed invariant for the internal view. The proofs are then completed by showing that the detailed invariant is maintained by all operations that modify the data structure.

However, experience with this approach shows that it generates an excessive number of proof artefacts (annotations and proof rules).

- Since SPARK programs cannot use a heap, each data structure must define and manage its own set of free elements, and the data structure invariant must include the correct management of those elements.
- Proof of correctness for linked data structures requires proofs about the reachability of the elements, but reachability cannot be fully proved using first-order logic. Consequently the user has to provide a large number of detailed rules about how reachability changes as the data structure is modified.

An alternative method of completing the proofs is illustrated using a singly linked ordered list of integers. The method can be tailored to different levels of integrity – the proof rules can be validated manually or they can be shown to follow from an established axiomatization of reachability.

All the files for the examples used in this paper (code, rules, Proof Checker scripts) are available from the Data Structures page at www.sparksure.com.

2 A simple data structure

2.1 The specification

SPARK data structures must be bounded, so the capacity is defined:

```
Capacity : constant := 100;
```

The list type is declared:

```
type T is private;
```

Ordered is the (abstract) data structure invariant.

```
--# function Ordered(L : T) return Boolean;
```

Set_Of_List is the abstraction function for a list and Set_Of_One converts an integer to a set containing just that integer:

```
--# function Set_Of_List(L : T) return Set;
--# function Set_Of_One(X : Integer) return Set;
```

Empty and Full are query functions on the list:

```
function Empty(L : T) return Boolean;
--# pre Ordered(L);

function Full(L : T) return Boolean;
--# pre Ordered(L);
```

Initialize creates an empty list and establishes the invariant for the list:

```
procedure Initialize(L :  out T);
--# post Ordered(L)
--# and Empty(L);
```

Insert is one of the operations that manipulates the list; it must maintain the data structure invariant:

```
procedure Insert(L : in out T;
                 X : in    Integer);
--# derives L from *, X;
--# pre  Ordered(L)
--# and (not Member(X, Set_Of_List(L)) -> not Full(L));
--# post Ordered(L)
--# and Set_Of_List(L) = Union(Set_Of_List(L~),
--#                            Set_Of_One(X));
```

Other operations (Delete, Contains) are also provided, this paper uses Insert as its example.

2.2 The implementation

A marker value is required to indicate the last item in a list:

```
List_End : constant := 0;
```

Subtypes for the link values and the array indices:

```
type Index is range 0 .. Capacity;
subtype List_Index is Index range 1 .. Capacity;
```

The array types:

```
type Link_T is array(List_Index) of Index;
type Data_T is array(List_Index) of Integer;
```

The implementation of the list type:

```
type T is
 record
  Head : Index;
  Free : Index;
  Next : Link_T;
  Data : Data_T;
 end record;
```

Note that the links between items (stored in Next) and the data values (stored in Data) are in separate arrays rather than stored in a single array of records. The reasons for this are explained later.

2.3 The data structure invariant

The data structure invariant requires reachability to be defined:

```
--# function Reachable(L : T;
--#                    I, J : Index) return Boolean;
-- Reachable(L, I, J) is defined as:
--   I = J -> True
--   I /= J and I = List_End -> False
--   I /= J and I /= List_End -> Reachable(L, L.Next(I), J)
```

The detailed (concrete) data structure invariant states that:

- both the data list and the list of free items terminate (i.e. List_End is reachable from L.Head and L.Free)
- each item is exclusively in the list or is a free item
- every item in the list, except the last, has a data value that is lower than its successor.

```
-- Ordered(L) <->
--  (Reachable(L, L.Head, List_End)
--   and
--   Reachable(L, L.Free, List_End)
--   and
--   (for all I in List_Index => (Reachable(L, L.Head, I)
--                    xor
--                    Reachable(L, L.Free, I)))
--   and
--   (for all I in List_Index =>
--         ((Reachable(L, L.Head, I)
--          and
--          L.Next(I) /= List_End)
--         -> L.Data(I) < L.Data(L.Next(I)))))
```

3 The initial attempt

The natural way to approach the correctness proofs is to use the detailed version of the invariant in the refined pre- and post-conditions in the package body.

The refinement verification conditions will then be satisfied by a proof rule that defines the correspondence of the two versions of the invariant. The main proof task becomes the demonstration, for each operation, that its refined post-condition follows from its refined pre-condition.

However, completion of the proof of the refined post-conditions becomes very difficult. The factors that contribute to this difficulty include the following.

- There are three main components to be developed - code, annotations and proof rules.
- These components are very strongly interrelated and changing any one of them requires changes to the other two.
- Because of the way that the Simplifier applies user defined proof rules, most rules are specific to a particular conclusion within a particular verification condition. Therefore a large number of rules, each slightly different to all the others, must be provided by the user.
- Many of the user defined proof rules will be about changes to reachability of items as the Next array is updated. These rules are quite complicated to write but must be shown to be valid – that they follow from the definition of reachability. However, the validation of these rules is quite a lengthy process, and it would be wasteful to do this before the code and annotations (and hence the set of rules) are in their final forms.
- Inevitably, errors and omissions are found in some proof rules when they are formally validated. This triggers further changes to the code and annotations and, consequently, to the proof rules, some of which may already have been formally validated but are now no longer required or required in a changed form.

The problems of managing the interaction of the changes to code, annotations and proof rules makes the direct completion of the proofs of the post-conditions an impractical approach.

4 An alternative approach

4.1 Defining a single rule

Since each operation makes well-defined changes to the structure of the list, an alternative approach is to consider the overall effect of each change and encapsulate this in a single rule.

The simplest way to see the overall change to a structure is to draw the before and after diagrams.

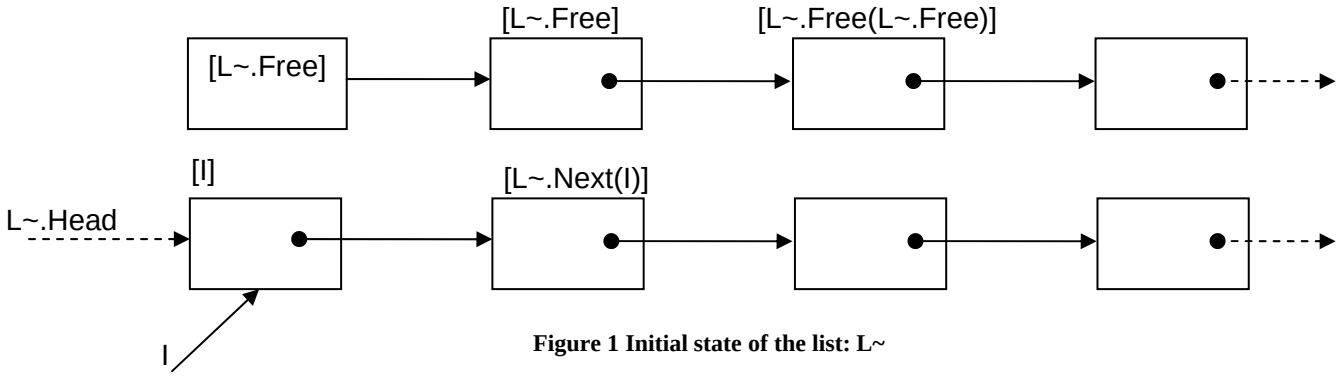

Figure 1 Initial state of the list: L~

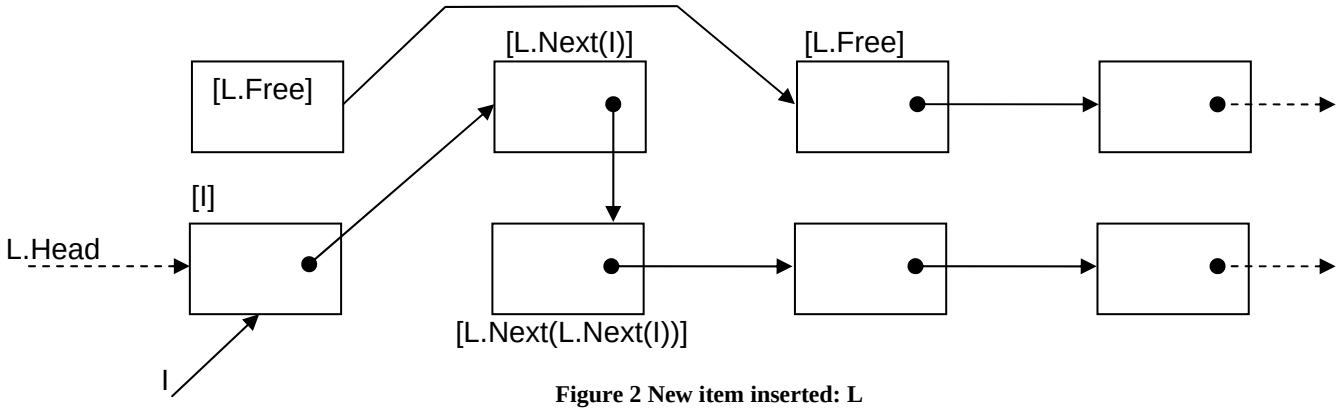

Figure 2 New item inserted: L

Figure 1 above shows the state of the links in the imported list for a call of Insert and Figure 2 shows the links in the exported list after a new item has been added following the item pointed at by the index I.

The expressions in square brackets appended to some of the items are the expressions that access those items in each of the lists.

Since the items in the two figures correspond, it is simple to write down the equalities that define the required changes to be made:

```
L.Next(I) = L~.Free
L.Free = L~.Next(L~.Free)
L.Next(L.Next(I)) = L~.Next(I)
```

It is now straightforward to write a single proof rule that determines whether the code of Insert has correctly added the required item into the list. The rule must include the correct initial conditions (to ensure that it is only applied where it is valid) and constraints to ensure that only the required changes are made.

For the Insert operation the rule must also ensure that the new item is correctly located in the list.

Within the rule the required changes (the above equalities) are translated into FDL (the language of the SPARK proof tools). L_Old is used in the rule for L~.

```
ordered(L)

  may_be_deduced_from

[ /* The correct initial conditions            */
  ordered(L_Old),
  reachable(L_Old, head, I),
  fld_free(L_Old) <> list_end,

  /* The required changes                      */
element(fld_next(L), [I]) = fld_free(L_Old),
fld_free(L) = element(fld_next(L_Old),
              [fld_free(L_Old)]),
element(fld_next(L), [element(fld_next(L), [I])])
          = element(fld_next(L_Old), [I]),

/*  No other changes are made to the Next array */
fld_next(L) = update(update(fld_next(L_Old),
                  [fld_free(L_Old)],
                  Y),
                [I],
                Z),

/*  Head is unchanged.                        */
fld_head(L) = fld_head(L_Old),

/* The correct value is written into the Data array */
fld_data(L) = update(fld_data(L_Old),
              [fld_free(L_Old)],
              X),

/* The new item is in the correct position     */
X > element(fld_data(L_Old), [I]),
element(fld_next(L_Old), [I]) = list_end
 or X < element(fld_data(L_Old),
          [element(fld_next(L_Old), [I])]) ] .
```

(The checks that values are within their subtype ranges have been omitted in this paper.)

There are several important features to this rule.

- Every component of the exported list is fully defined.
- The update expressions for Next and Data are in separate sideconditions – this is one reason for

defining them as two separate arrays in the list record type. If a single array had been used then there would be a single, much more complex update expression.

- The rule simply states the overall effect required by the code, so it is linked very loosely to the form of the code (in contrast to the very tight linkage found in the initial approach). The above form does require the two updates to the Next array to be in a specified sequence, but (if required) this could be avoided by using additional wild cards as the indexes of the two updates. It is noted in the Conclusions section that this sequence is, in fact, a deliberate choice.
- A second rule with the conclusion:
  Set_Of_List(L) = Union(Set_Of_List(L_Old),
  Set_Of_One(X))

  and identical side-conditions is required to fully prove the post-condition on Insert.
- Similar rules can be created for the Delete operation. (But all other operations, including Initialize, are proved conventionally.)

For a certain level of integrity it would be sufficient to validate the rules for Insert and Delete, developed in this way, by manual inspection.

For a higher level of integrity it would be desirable to validate the rules more formally. This can be done by creating verification conditions that correspond to the rules and proving these using the SPADE Proof Checker. Since the development of these proofs is likely to be quite a lengthy process it would be helpful to first reduce the number of the rules that need to be validated.

4.2 A modified list definition

The rule shown above is specific to the insertion of an item within the list and a slightly different rule is required for the addition of an item as the first in the list.

The need for a second set of rules specific to the first item in the list can be avoided by creating an additional element within the Next array that is used in place of the Head component.

```
Head   : constant := Capacity + 1;
List_End : constant := 0;
type Extended_Index is range 0 .. Head;
subtype Extended_List_Index is
      Extended_Index range 1 .. Head;
subtype Index is
      Extended_Index range 0 .. Capacity;
subtype List_Index is
      Extended_List_Index range 1 .. Capacity;
type Link_T is array(Extended_List_Index) of Index;
type Data_T is array(List_Index) of Integer;
type T is
  record
    Free : Index;
    Next : Link_T;
    Data : Data_T;
  end record;
```

Note that the two arrays now have different index subtypes – this is the second reason for keeping the links and data values in separate arrays. A spurious data value for the Head element would introduce substantial complications into the proofs.

The only changes to the rule as given above are to remove the references to fld_head and to change the test that X is greater than the I^{th} element to:

```
I = head
  or X > element(fld_data(L_Old), [I])
```

4.3 Validating the rule

It is simple to create the verification condition that validates a rule: convert the side-conditions to hypotheses with wildcards in the rule (the names with upper-case characters) replaced by variables of the appropriate type (these are declared in an associated fdl file, which is not shown here).

The verification condition (VC) that proves the modified rules for Insert is:

```
H1:  ordered(l_old) .
H2:  reachable(l_old, head, i) .
H3:  fld_free(l_old) <> list_end .
H4:  element(fld_next(l), [i]) = fld_free(l_old) .
H5:  fld_free(l) = element(fld_next(l_old),
                 [fld_free(l_old)]) .
H6:  element(fld_next(l), [element(fld_next(l), [I])])
           = element(fld_next(l_old), [i]) .
H7:  fld_next(l) = update(update(fld_next(l_old),
                      [fld_free(l_old)],
                      y),
                 [i],
                 z) .
H8:  fld_data(l) = update(fld_data(l_old),
                 [fld_free(l_old)],
                 x) .
H9:  i = head
     or x > element(fld_data(l_old), [i]) .
H10: element(fld_next(l_old), [i]) = list_end
     or x < element(fld_data(l_old),
              [element(fld_next(l_old), [i])]) .
 ->
C1:  ordered(l) .
C2:  set_of_list(l) = union(set_of_list(l_old),
                  set_of_one(x)) .
```

Definitions of the various proof functions in the VC (ordered, reachable, union, equality of sets, etc.) are provided in further defining rules (not given in this paper).

A proof of this VC using the Proof Checker will formally validate the rule. We know that this proof will require further rules about reachability, as a proof that uses only the definition of reachability is not possible using first-order logic. However we cannot predict exactly what rules may be required until the proof is attempted, so attempting the proof is the easiest way to identify the further rules that are required.

When this is done for the rules for both Insert and Delete, about 10 further rules are identified. Examples of these rules are:

```
reachable(L, I, J)
  may_be_deduced_from
[ reachable(L, I, K),
  reachable(L, K, J) ] .

reachable(L, I, J) <-> not reachable(L, J, I)
  may_be_deduced_from
[ reachable(L, K, I),
  reachable(L, K, J),
  reachable(L, K, list_end),
  I <> J ] .

reachable(L2, I, J)
  may_be_deduced_from
[ reachable(L1, I, J),
  fld_next(L2) = update(fld_next(L1), [K], X),
  not reachable(L1, I, K) ] .
```

Some of these rules, such as the first of the above, could be validated manually but others are more complex and less easy to validate manually. Fortunately the basis for formal validation of these rules is already available.

Nelson [1] provides an axiomatization of reachability which, along with the other results proved in [1], can be converted to Proof Checker rules. Using just these rules and other defining rules that, for example, follow from the declarations in the Ordered_Lists package, the verification conditions corresponding to all the additional rules can be proved using the Proof Checker.

There is one significant addition to Nelson's analysis that is required – the definition of list termination. This is not defined in [1], but clearly must be taken into account in these proofs. In particular this means that, for Nelson's function *f*, *f(u)* is not defined if *u = list_end*.

4.4 Binary tree example

For this approach to be practicable, the rules generated should not increase rapidly in size and complexity as the number of changes to the data structure increases. The most complex application of this method to date is the deletion of elements from a binary search tree.

There are several different cases for deletion and it helps if there is a rule for each case. The following example is for the deletion of an internal node that is the left child of its parent.

Figures 3 and 4 are the before and after diagrams for this case. The node to be deleted is replaced by its immediate successor.

There are four item pointers:

- I points to the item to be deleted,
- P points to its parent,
- Succ is the immediate successor of I
- Succ_P is its parent.

In this case there are five changes to the links, compared to two changes in the list example.

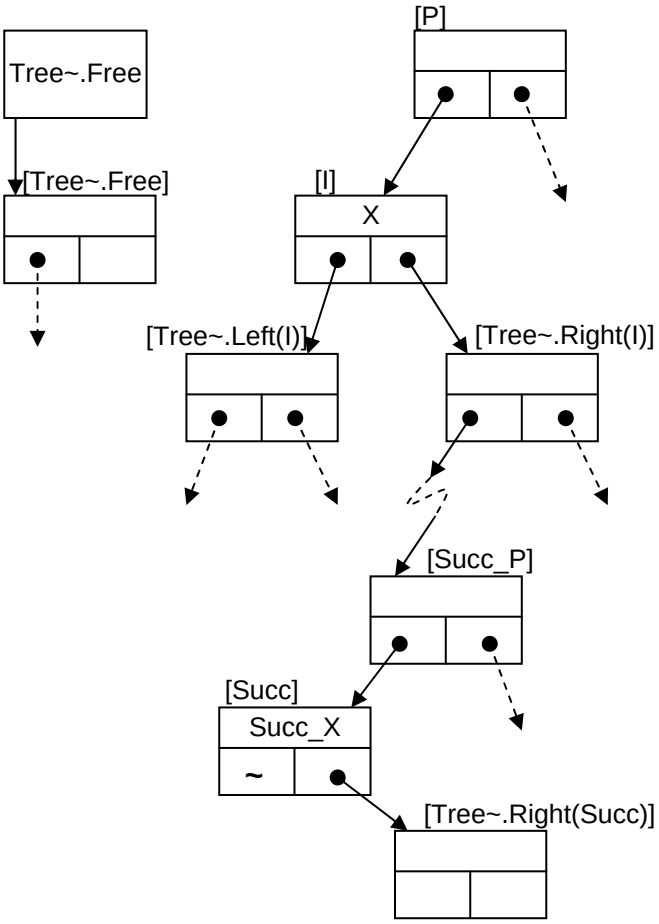

Figure 3 Before deletion of item I

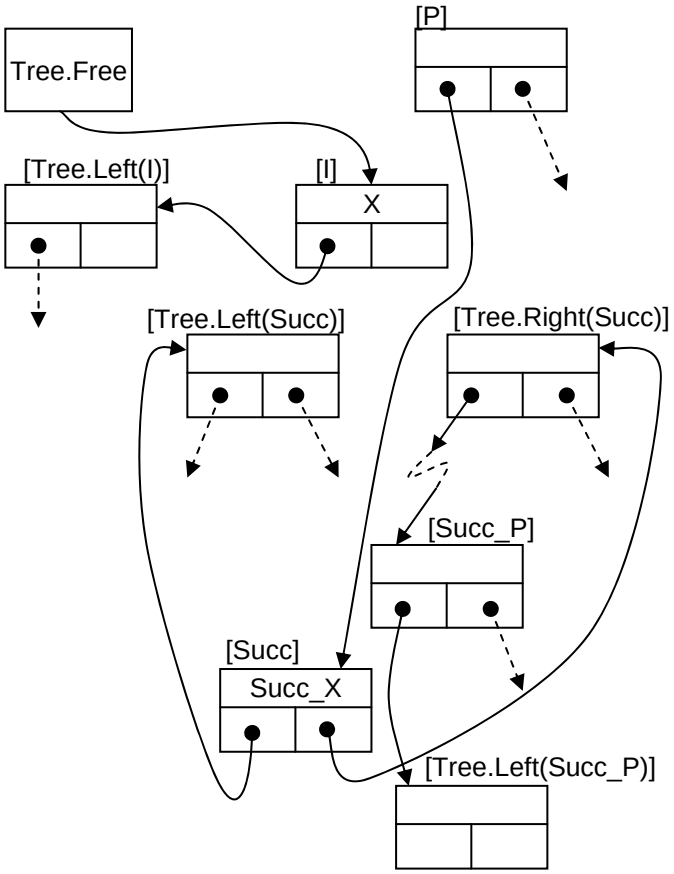

Figure 4 After deletion of item I

This example indicates that the rate of increase in the size and complexity of the rule is not worse that linear with the number of changes to the data structure.

The rule for this action is:

```
deleted_internal_left(Tree_Old, Tree, P, I)

 may_be_deduced_from

[ /* The correct initial conditions                */
 binary_tree(Tree_Old),
 ancestor(Tree_Old, root, P),
 I = element(fld_left(Tree_Old), [P]),
 element(fld_left(Tree_Old),  [I]) <> leaf,
 element(fld_right(Tree_Old), [I]) <> leaf,
 element(fld_left(Tree_Old),
       [element(fld_right(Tree_Old), [I])]) <> leaf,
 immediate_successor(Tree_Old, I, Succ),
 Succ = element(fld_left(Tree_Old), [Succ_P]),

 /* The correct changes are made to the tree      */
 fld_free(Tree) = I,
 element(fld_left(Tree), [I]) = fld_free(Tree_Old),
 element(fld_left(Tree), [P]) = Succ,
 element(fld_left(Tree),  [Succ]) =
            element(fld_left(Tree_Old), [I]),
 element(fld_right(Tree), [Succ]) =
            element(fld_right(Tree_Old), [I]),
 element(fld_left(Tree),  [Succ_P]) =
            element(fld_right(Tree_Old), [Succ]),

 /* No other changes are made to the tree        */
 fld_left(Tree) =
   update(update(update(update(fld_left(Tree_Old),
                       [Succ_P],
                       Y1),
                  [Succ],
                  Y2),
            [P],
            Y3),
        [I],
        Y4),
 fld_right(Tree) = update(fld_right(Tree_Old),
                [Succ],
                Z),

 /* The Data array is unchanged                  */
 fld_data(Tree) = fld_data(Tree_Old) ] .
```

The complete analysis for a binary tree is also available. In this case the rules have not been validated further (i.e. by proving the related verification conditions with the Proof Checker).

5 Conclusions

The main conclusion to be drawn is that the natural approach to the proof of correctness for linked data structures is unlikely to succeed, however the alternative approach described here may be sufficiently rigorous for the integrity required.

A broader conclusion, strongly supported by the work reported here, concerns the use of proof within a software development process:

> *The completion of proofs should not be separated from the software design and code activities.*

The completion of proofs is, typically, quite difficult and time-consuming. Furthermore the difficulties experienced are strongly affected by design and code details that will often appear irrelevant to design and code engineers who are not concerned with the proofs. There are several instances of this in the ordered list data structure.

- The structure of the list record, with separate arrays for the links and data is unlikely to be the implementation chosen by most software engineers, and the creation of a data value for the Head element is unlikely to be seen as a problem.
- In the discussion of the rule for Insert in Section 4, it was noted that the two updates to the Next array were required to be in a specific sequence. This sequence is, in fact, required by the formal validation of the rule. When writing the code each possible sequence was considered and the choice made as the one that was thought likely to lead to easier proofs than the alternatives. An engineer not concerned with proof would not have considered this.
- The code of the Insert operation has an unusual and slightly inefficient form (with a test that is not required in the more obvious form). This makes the proofs easier to create and validate, but would never be the choice if proof were not being taken into account.

Projects that delay work on the proofs until after completion of design and code are therefore likely to experience more problems with those proofs.

This conclusion should be taken into account by anyone using code already developed to evaluate the use of proof; they are unlikely to see the full potential benefits unless they are prepared to modify the code during that evaluation.

References

[1] Greg Nelson (1983), *Verifying Reachability Invariants of Linked Structures*, in Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages.

The text of this reference is available via the Google scholar link at: http://www.informatik.uni-trier.de/~ley/db/conf/popl/popl83.html

"Crimeville" – using Ada inside an on-line multi-user game

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation, Vesterbrogade 148 K, 1620 København V, Danmark; email: jacob@jacob-sparre.dk

Abstract

This paper presents an Open Source language server developed for the children's on-line multi-user game "Crimeville". The language server was developed in Ada using the POSIX Ada API. The language server is to the author's knowledge the first time Ada has been used when implementing a commercial computer game, and we are thus pushing the boundaries of where Ada is an accepted programming language.

The language server is an example of how the POSIX Ada API can be used to launch external applications, and handle communications with them through POSIX pipes.

1 Introduction

The children's game universe "Crimeville" challenges the players to solve detective riddles cooperatively. In the on-line version of the game this means that the players in each session of the game can chat with each other.

To help the children write better – and to limit them being naughty – the chat is going through a language server. The language server is thus a part of the game developers' attempt to assure that the on-line game is "safe" for children.

The language server is written in Ada using the POSIX Ada API [1] [1]. It uses existing Ispell [3] compatible Open Source spell-checkers and their associated dictionaries as an integrated part of the system. The language server is implemented such that it should be possible to reuse it "as-is" for future games, and has since the Ada Europe 2011 conference been available as Open Source software.

The paper is structured roughly following the development of the language server going from requirements (section 2) over the specification (section 3) to architectural decisions (section 4) and seclected implementation details (section 5). Following that I will present information on source code reuse (section 6), and finally a few words on performance (section 7) and a conclusion.

2 Requirements

When Art of Crime, the developers of "Crimeville", contacted me, they described the task as helping the players write correctly, and limit how much they insult each other. They wanted to do this on the level of whole words, and were interested in leveraging existing Open Source spell-checking dictionaries for this purpose.

In addition to this, it was expressed that the solution had to be "sufficiently" [2] fast.

3 Specification

We decided to solve the task by implementing a TCP/IP server responsible for checking words in a specific language. Behind the scenes this server should run two instances of the Aspell [2] spell-checker; one with an ordinary dictionary of known (polite) words, and one with a specialised dictionary of foul words. The Aspell instances should be easily substitutable with another Ispell compatible spell-checker, in case tests showed that Aspell was too slow.

We mapped out the categorisation rules going from spell-checker responses:

- Correct / Misspelled / Unknown correct word
- Correct / Misspelled / Unknown foul word

to categories used by the in-game chat:

- Correct word – allowed
- Foul word – forbidden
- Misspelled word – to be corrected
- Unknown word – allowed

This mapping is shown in figure 1.

A protocol for the communications with the language server was defined. The protocol is text-based (ISO-8859-1) with messages terminated by Ada.Characters.Latin_1.LF (i.e. line-based). The client sends a request formatted as:

```
Request_Key  Space  Word
         {   Space  Word } LF
```

[1] In practise using the Florist [4] implementation of the standard as distributed with Debian GNU/Linux.

[2] Later this was revealed to be relative to known bottlenecks in the game, and the speed of the selected production hardware.

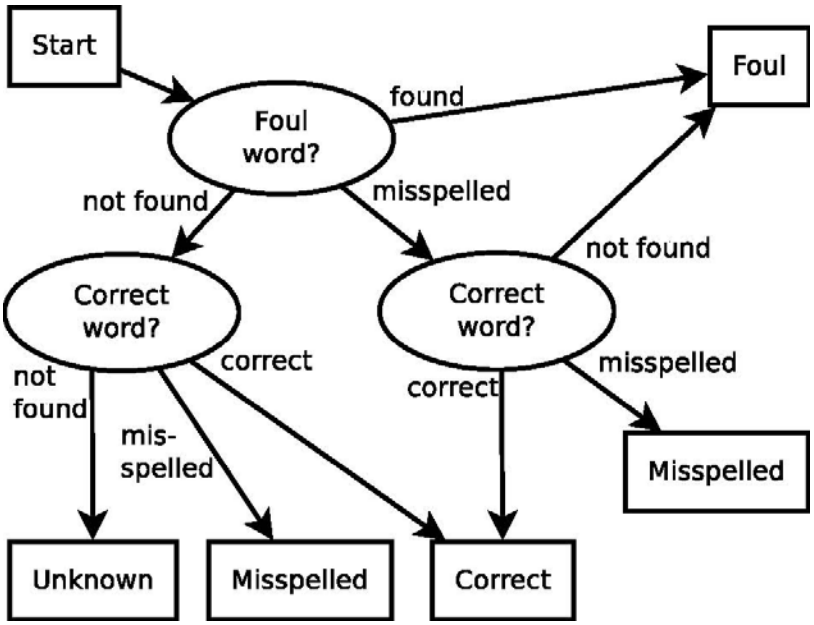

Figure 1 Diagram showing how it is decided which category a word should be put in. This diagram was drawn together with Art of Crime staff as a part of the specification of the system.

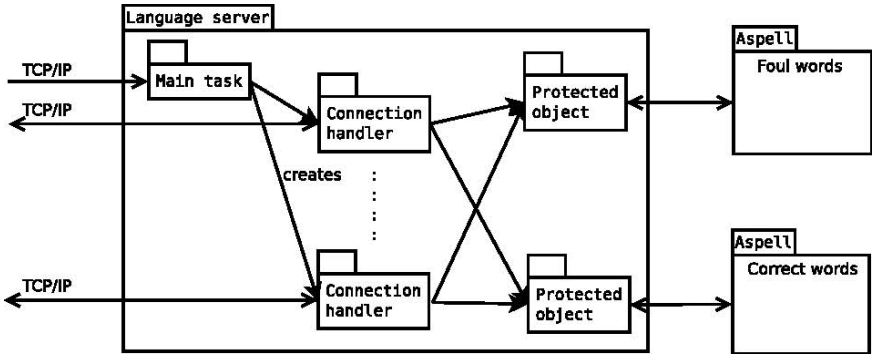

Figure 2 Diagram showing the architecture of the language server. The two "aspell" processes are launched by the protected objects in the language server. The connection handler tasks are created by the main task.

The matching answer from the server is formatted as:

```
Request_Key   Space  Category_ID   Word
       {      Space  Category_ID   Word } LF
```

where the "Category_ID" is "+" for a correct word, "-" for a foul word, " " for a misspelled word and "?" for an unknown word [3].

We agreed to postpone any optimisations of the system until we had it working and could compare its speed with the requirements.

4 Architecture

The specification lead to an architecture (figure 2), where the main task is responsible for setting up the system and listening for incoming TCP/IP connections. Whenever a client connects to the server, the main task creates a new connection handling task and hands over the client to that task.

When a connection handler needs to check a word in one of the two dictionaries it calls a procedure in a protected object, which encapsulates two POSIX pipes connected to a spell-checker process [4].

The incoming connections from the clients are encapsulated in a package, hiding the actual protocol inside two types representing messages respectively from and to the clients.

[3] This is actually the second iteration of the protocol. In the first iteration, only one word could be sent to the language server in each message.

[4] Technically I/O is a potentially blocking operation, and thus a bounded error, when used from a protected object. Since it works in practise, and is a less complicated construction, than having a task doing the I/O, I have decided to keep it like this for now.

5 Implementation

In the following sections I will present selected parts of the source code for the Crimeville language server. In section 5.1 I will present the core of the system making the decisions about categorising words. In section 5.2 I will show how one can use the POSIX Ada API to launch an external application with POSIX pipes connected to "Standard_Input", "Standard_Output" and "Standard_Error" of the external process. Finally, in section 5.3 I will show how the external protocol of the language server is encapsulated.

5.1 Application logic

One of the great benefits of using Ada is that it is designed for source code to be read and understood. One of my goals when implementing a system is that my customers should be able to understand enough of the source code, to make it possible for them to verify that the system is likely to do what they intend it to do [5]. It is not my intent that any customer necessarily should be able to read my source code unaided, but I should be able to talk them through the core of the system; in this case the bit of logic deciding which category a word falls in. The core application logic of the Crimeville language server reads like this:

```
Foul_Words.Check (Word => Word,
                  Class => Class ) ;
case Class is
   when Aspell.Found =>
      return Game_Communication.Foul_Word;
   when Aspell.Misspelled =>
      Dictionary.Check (Word => Word,
                        Class => Class ) ;
      case Class is
         when Aspell.Found =>
            return Game_Communication.
                                   Correct_Word;
         when Aspell.Misspelled =>
            return Game_Communication.
                                    Misspelled_Word;
         when Aspell.Not_Found |
           Aspell.Timeout | Aspell.Error =>
            return Game_Communication.
                                    Foul_Word;
      end case ;
   when Aspell.Not_Found | Aspell.Timeout |
    Aspell.Error =>
      Dictionary.Check (Word => Word,
                Class => Class ) ;
      case Class is
         when Aspell.Found =>
            return Game_Communication.
                                   Correct_Word;
         when Aspell.Misspelled =>
            return Game_Communication.
                                   Misspelled_Word;
         when Aspell.Not_Found | Aspell.Timeout |
              Aspell.Error =>
            return Game_Communication.
                                   Unknown_Word ;
      end case;
end case;
```

"Foul_Words" and "Dictionary" are the two protected objects encapsulating the Aspell processes. The "Check" operation looks up a word in Aspell, and returns the class Aspell puts it in. In practise we group together an unexpected result ("Error") or no result ("Timeout") from Aspell with words Aspell cannot find in the dictionary.

It is worth noting that the code structure with two levels of case statements in part is like that to match the specification diagram (figure 1)[6]. If the specification of the classification rules had been in tabular form, it would have made more sense to create the mapping using a constant two-dimensional array.

5.2 Launching an Aspell process

When we launch an Aspell process to take care of phonetic mapping and dictionary look-up, the first step is to set up the POSIX pipes to be used to communicate with the Aspell process. The second step is to "fork" the running process, i.e. create a clone of it, where the only difference of importance to us is the result from the "fork" function [7]. The third step is to close the unused ends of the pipes. Finally – in the cloned process – we move the pipes to the required positions ("Standard_Input", "Standard_Output" and "Standard_Error"), before we substitute the language server program with the aspell program using the "Exec_Search" procedure.

Altogether this is the core of the subroutine "Pipe_Fork_Exec_Search":

```
POSIX.IO.Create_Pipe (Write_End => To_Child,
                      Read_End => From_Parent ) ;
POSIX.IO.Create_Pipe (Write_End => To_Parent,
                      Read_End => From_Child ) ;
POSIX.IO.Create_Pipe (Write_End =>Errors_To_Parent,
                Read_End => Errors_From_Child ) ;
case Fork is
   when Parent =>
     POSIX.IO.Close (From_Parent ) ;
     POSIX.IO.Close (To_Parent ) ;
     POSIX.IO.Close (Errors_To_Parent ) ;
```

[5] I have already once, on a previous project, had great benefit from insisting on this kind of code review. My customer noticed that the program – although implemented to the specfication – didn't do exactly what was intended, and we caught the error before we started testing the system.

[6] There is also the hope that it may avoid a few procedure calls to the "Dictionary" protected object.

[7] The other difference is that only the POSIX thread calling "fork" is active in the cloned process.

Unit	Compilation units	Subroutines	Lines	Characters
Standardised	18	26	6297	205858
Vendor-provided	2	9	3480	111864
Reused	5	5	111	3920
Reusable	7	20	344	10934
Project-only	6	13	485	16592
Total	38	73	10717	349168

Figure 3 Levels of source code reuse measured in different units. Standard library line and character counts are for the GNAT and Florist implementations. Note that for standard and vendor-provided packages only explicitly withed packages and called subroutines are included in the counts. Since the standard and vendor-provided packages provides relatively more unused functionality, their line and character counts are much relatively larger than for in-house produced source code.

```
when Child =>
   POSIX.IO.Close (To_Child );
   POSIX.IO.Close (From_Child ) ;
   POSIX.IO.Close (Errors_From_Child ) ;
   Move (From => From_Parent,
         To => POSIX.IO.Standard_Input ) ;
   Move (From => To_Parent ,
         To => POSIX.IO.Standard_Output ) ;
   Move (From => Errors_To_Parent ,
         To => POSIX.IO.Standard_Error ) ;
   POSIX.Unsafe_Process_Primitives.Exec_Search (
       Program_Name,
       Arguments ) ;
end case ;
```

In the source above, the procedure "Move" is a local procedure, which encapsulates the POSIX subroutines "Duplicate_And_Close" and "Close" with appropriate error handling.

5.3 Client query protocol

The protocol used by clients (i.e. the Crimeville game server) to ask the language server to classify the words in a sentence is encapsulated in the package "Game_Communication". The public part of the package specification is:

```
package Game_Communication is
   type Paragraph is private;
   procedure Get ( Item : out Paragraph;
      From : in out Buffered_IO.File) ;
   function More_Words ( Source : in Paragraph )
      return Boolean ;
   procedure Get_Next_Word (
      Source : in out Paragraph;
      Word : out Ada.Strings.Unbounded.
                 Unbounded_String ) ;
   type Classifications is private;
   type Word_Classifications is (Correct_Word,
                                 Misspelled_Word,
                                 Unknown_Word,
                                 Foul_Word);
   procedure Copy_Key ( Source : in Paragraph;
      Target : out Classifications);
   procedure Append ( Target : in out Classifications;
      Word : in Ada.Strings.Unbounded.
                                 Unbounded_String ;
      Classification : in Word_Classifications);
   procedure Put ( Item : in Classifications ;
      To : in Buffered_IO.File) ;
private
```

The type "Paragraph" represents a query from a client, containing a query key and a sentence to be classified. The primitive operations of this type allow the user of the package to read a query from a le (in practise a TCP/IP socket) and to pull out the individual words in the query.

The type "Classifications" represent a response from the language server. The primitive operations of this type allow the user of the package to compose a response of words and word classifications and write it to a file (in practise a TCP/IP socket).

6 Reuse

Just as the language server application has been developed with the intent that it can be reused in other systems than Crimeville, the development of the system has also been done with attention to reuse of existing source code (and data) in mind. First of all, we reuse the existing Aspell spell-checker and related dictionaries and phonetic rules.

When it comes to the source code for the language server, I have used both standardised packages, compiler vendor provided libraries, and some of my own pre-existing packages. In addition to this I have developed some packages which I hope to be able to reuse in the future. The table in figure 3 gives an overview of the distribution of source code use on these categories in terms of compilation units of withed packages, called subroutines, and lines and characters in the source files of the withed packages.

One of the not yet reused compilation units is "Pipe_Fork_Exec_Search", the core of which is shown in

section 5.2. In earlier projects I have written this commonly used idiom from scratch every time I have needed it.

The compilation units I expect to reuse, even though they have been written in connection with this project are:

- "Buffered_IO": Adds a minimal "Ada.Text_IO"-like interface on top of "POSIX.IO"
- "Daemon": Imports the C function "daemon", which is used to disconnect a process from its terminal and parent process.
- "Logging": Simple logging package. Encapsulates an "Ada.Text_IO" file in a protected object, which only allows writing whole lines.
- "Pipe_Fork_Exec_Search": Launches an external program with POSIX pipes to its standard input, output and error files.

7 Performance

The capacity of the language server was measured on the production hardware chosen by Art of Crime, and was found suficient. This means that it wasn't found necessary to optimise the initially delivered system.

We also measured the distribution of the CPU use between the actual language server process (the part written in Ada) and the aspell processes (written in C++). The language server uses approximately 5% of the used CPU resources, while the aspell processes use the other 95%.

8 Conclusion

Altogether I consider this project a success. I have created a working Ada application which – without any problems – has been put in production in an environment outside of what is typical for Ada.

Some specific points which I believe have been positive for the project and my customer's confidence in the solution:

- Solving the task as a stand-alone TCP/IP server allowed me to
  - use the best programming language for the task, independently of what was used for other parts of the complete system.
  - make an easily reusable system
  - have a well-defined boundary between my responsibilities and those of my customer

  and in general made the language server independent of the actual Crimeville game server.
- Using existing Open Source spell-checkers allows us to reuse existing language data such as dictionaries and phonetic rules.
- Using the Ispell pipe protocol to communicate with the spell-checker allows us to switch between different spell checkers with only a small modification of the system.

The complete source code for the language server can be downloaded from

http://www.jacob-sparre.dk/spelling/crimeville.zip.

References

[1] *POSIX Ada95 Bindings for Protocol Independent Interfaces* (P1003.5c).

[2] Kevin Atkinson. *Aspell*, 2001.

[3] Geoff Kuenning. *International Ispell*.

[4] Students and faculty of the Florida State University Department of Computer Science. *Florist*. http://www.cs.fsu.edu/~baker/florist.html.

Ada User Guide for LEGO MINDSTORMS NXT

Peter J. Bradley, Juan A. de la Puente, Juan Zamorano
Universidad Politécnica de Madrid, Madrid, Spain; http://polaris.dit.upm.es/str

Abstract

The purpose of this guide is to introduce the robotics kit LEGO MINDSTORMS NXT to the Ada community. All the steps required to complete a working Ada application running under the LEGO MINDSTORMS NXT are covered.

Keywords: LEGO, MINDSTORMS, Ada, Ravenscar, Real-Time, Embedded, Robotics.

1 Introduction

The LEGO MINDSTORMS NXT (from now on NXT) is a simple and flexible robotics kit that allows Ada programmers to develop applications that interact with the "outside world" by means of sensors, actuators, etc. The dynamic features associated to this interaction with the physical environment require that the actions of the control software are executed at a specified time rate. Therefore, real-time constraints must be generally met. Ada's concurrency and real-time integrated features together with the use of the Ravenscar profile [1] makes it the ideal language for the NXT.

This guide is organised as follows. The first section is this introduction. Then, the second section shows some fundamental aspects of the NXT hardware that should be kept in mind for NXT Ada development. Section three briefly introduces Ada programming for the NXT taking into consideration the Ravenscar compliant NXT runtime system and the NXT Ada drivers library. The fourth section gives an overview of the development environment with a description of the tools required to work with the NXT. As an example, the development of a prototype vehicle is presented in section five. Finally, section six describes how the internal JTAG interface of the NXT is accessed and used to debug Ada programs.

Throughout this guide the AdaCore GNAT GPL for LEGO MINDSTORMS NXT 2011 hosted in GNU/Linux for x86 (available from http://polaris.dit.upm.es/str/projects/mindstorms) will be used but note that the Windows version is also available (http://libre.adacore.com/libre/tools/mindstorms).

2 MINDSTORMS NXT

2.1 Architecture overview

The NXT kit comes with a programmable controller, also called Intelligent Brick. This Brick (see figure 1 for its block diagram) features a 32-bit ARM main processor (AT91SAM7S256) with 64 KB of RAM and 256 KB of Flash memory that runs at 48 MHz. To assist the main processor an 8-bit AVR co-processor (ATmega48) is also included. Main processor and co-processor periodically communicate through an I^2C bus.

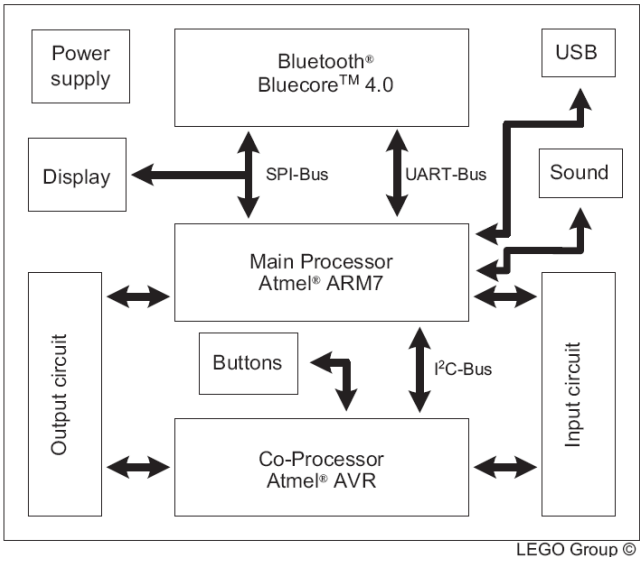

Figure 1 NXT block diagram

It also has three output ports, which are bidirectional, to connect and control actuators such as electrical motors or linear actuators and four input ports that support both digital and analog sensors.

Communications with the Brick are possible using either USB, via a full-speed USB 2.0 port, or Bluetooth, available through a CSR BlueCore 4 chip that is connected to the ARM's USART. The USB 2.0 port is usually used to connect to a PC and Bluetooth to communicate with other NXT Bricks or any other Bluetooth enabled devices such as smartphones, tablets, etc.

On the top of the Brick there is a 100 x 64 pixel LCD display connected to the main processor via a SPI bus (serial peripheral interface bus), and four rubber buttons, controlled by the co-processor, to interact with the Brick.

The NXT Brick also comes with an audio amplifier, connected to the ARM PWM (pulse-width modulation) controller, and a 16 Ω speaker with a bandwidth of 2 -16 KHz.

For schematics and further information refer to *LEGO MINDSTORMS NXT Hardware Developer Kit* [2].

2.2 Processor and co-processor

The AVR co-processor handles the following low-level tasks for the main processor:

- **Power management**. Turns the NXT Brick off and wakes it up when the center orange button is pressed. It

also monitors the battery status sending information to the ARM processor.

- **PWM generation**. Generates pulses for the three output ports at a frequency of 8 KHz with the duty cycle specified by the ARM processor.
- **A/D conversion**. Performs a 10 bit digital conversion of the analog signals at the input ports every 3 ms.
- **Button decoding**. Decodes the buttons so that the main processor is able to tell which buttons are pressed and which are not. Note that the co-processor does not carry out any button debouncing. If it is not handled at driver level the programmer should take care of it.

To handle all of the above it is necessary for main processor and co-processor to periodically exchange information. The communication between the two microcontrollers is set up as two memory allocations that, on the original LEGO firmware, are updated on both microcontrollers every 2 ms. The communication interface operates at 380 Kbit/s using the I^2C hardware interface in both microcontrollers with the ARM main processor functioning as master.

2.3 Output ports

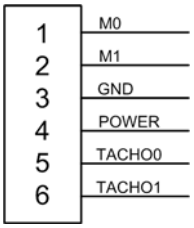

Figure 2 Output port generic schematic

All of the three output ports work in the same manner, see figure 2. They have a ground (GND) and a 4.3 V supply output (POWER). Two output signals (M0 & M1) that come from an internal H-bridge motor driver that controls the motor standby, forward, reverse or brake modes. This motor driver is governed by the PWM pulses generated by the co-processor. It also has two input signals (TACHO0 & TACHO1) that are connected to the main processor's parallel input/output controller (PIO) using a Schmitt trigger for noise suppression. Within the Ada drivers these two last signals are used for the motor encoder. The encoder has a resolution of 360 counts per revolution. When the motor rotates the ARM processor receives an interrupt in order to update the encoder counter through the parallel I/O controller. Notice that clockwise and counterclockwise operation is detected by the counter's increments or decrements.

2.4 Input ports

Depending on the type of sensor connected to the NXT Brick the input ports behave differently. The input ports allow both digital and analog interfaces, see figure 3.

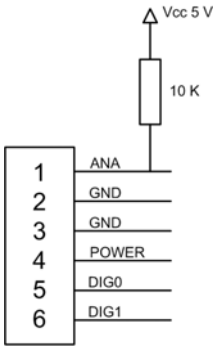

Figure 3 Input NXT generic schematic

LEGO considers three types of sensors:

- **Active sensors**. This kind of sensors belongs to the previous version of LEGO MINDSTORMS, the RCX. They require an NXT adapter cable. NXT firmware provides the same functionality available in the RCX Bricks by using an extra current source. This current source delivers power (approximately 18 mA) to the active sensors. It supplies power to the sensor through the analog pin (ANA) during 3 ms and then measures the analog value during the following 0.1 ms. The AVR sends the 10 bit digital conversion of the analog value to the main processor using the scheme presented in section 2.2.

  When using this kind of sensors (e.g. RCX light sensor, RCX rotation sensor) be sure to set the appropriate input power settings by calling `Set_Input_Power(sensor_id,RCX_9V)` from `NXT.AVR` driver package where `sensor_id` is the input port used for the active sensor.

- **Passive sensors**. These are analog sensors that do not need the special power/measurement timing of the active sensors. The power needs of these sensors are not covered via the analog pin (ANA) but via a specific pin (POWER). Note that the sampling of all the AVR A/D converters occurs simultaneously so active and passive sensors must be sampled at the same rate, 333 Hz.

  All of the sensors packed with the LEGO MINDSTORMS NXT are passive with the exception of the ultrasonic sensor.

- **Digital sensors**. These sensors contain all the necessary logic and processing resources to work independently. Thus, they perform their function autonomously and send or receive information to/from the ARM via an I^2C channel (DIGI0 & DIGI1) running at 9600 bit/s where the ARM functions as master. These sensors are mapped as external memory areas from/to which the programmer can read or write to control the behaviour of the sensor and harvest data. For a memory arrangement that optimises read and write access refer to *LEGO MINDSTORMS NXT Hardware Developer Kit* [2].

  The ultrasonic sensor is the only digital sensor packed in the NXT kit.

If a higher sampling rate is required by an analog input the hardware allows configuring DIGI1 as an analog input.

Port 4 can also function as a high-speed communication port. It has a RS485 IC that allows for high-speed-bi-directional multipoint communications.

2.5 Bluetooth features

The NXT Brick can be connected using Bluetooth to any other Bluetooth device that implements the Serial Port Profile (SPP), a serial cable emulation profile. The effective working Bluetooth range for the NXT Brick is approximately 10 m (Bluetooth Class II device).

The NXT Brick provides a master/slave communication scheme with four channels. Channel 0 is used when working as slave and the other three when working as master. The NXT Brick can either work as master or slave. This means that when the NXT Brick works as master it can communicate with three more devices.

The CSR BlueCore 4 firmware is implemented as a virtual machine with an integrated command interpreter. Thus, communication between the main ARM processor and the Bluetooth chip is handled by a set of defined commands and data streams that are exchanged through the USART channel. Refer to *LEGO MINDSTORMS NXT ARM7 Bluetooth Developer Kit* [3] for a full specification.

3 Ada programming for NXT

3.1 NXT run-time system

The AdaCore GNAT GPL for LEGO MINDSTORMS NXT 2011 cross-compiler toolchain relies on a Ravenscar small footprint run-time system (Ravenscar SFP). It is really a superset of the zero footprint profile. It adds the specification of a secondary stack mechanism for unconstrained objects and the Ravenscar tasking features to the zero footprint profile. This means that Ada applications for the NXT should comply with the Ravenscar profile for tasking purposes. Also, as it is targeted for use with embedded systems, it uses a sequential Ada subset where not all language features are available. For example, attributes 'Image and 'Value are not included. Moreover, there is no exception propagation. Unhandled exceptions jump to a "last chance handler" that can be reprogrammed as desired as long as the application then terminates (it must not return to the caller). Note that you must explicitly include the package NXT.Last_Chance, using a with-clause, for it to be part of your application. If you do not, a default handler is included that only displays an address for the exception on the NXT LCD screen. For a full description of the Ravenscar SFP profile refer *to GNAT User's Guide "Supplement for High-Integrity Edition Platforms"* [4].

The purpose of the Ravenscar profile is to restrict the use of many tasking facilities so that the outcome of the program is predictable. For this purpose, the profile is restricted to a fixed priority and pre-emptive scheduling. With fixed priority pre-emptive scheduling, the scheduler ensures that at any given time, the processor executes the highest priority task of all those tasks that are currently ready to be executed. Also, the Immediate Ceiling Priority Protocol (ICPP) is enforced by the Ravenscar profile. This means that when a task locks the resource, its priority is temporarily raised to the priority ceiling of the resource, thus no task that may lock the resource is able to get scheduled. This allows execution of a low priority task deferring execution of higher-priority tasks, thus minimizing priority inversion. More information can be found in *Annex D: Real-Time Systems of the Ada 2005 Reference Manual* [5].

When writing an Ada application for NXT you should bear in mind that only the Ada subset defined by the Ravenscar profile can be used for tasking. These are some of the restrictions:

- requeue statement.
- abort statements.
- Task entries.
- Dynamic priorities.
- Relative delays.
- Protected types with more than one entry.
- Protected entries with barriers other than a single boolean variable declared within the same protected type.
- Entry calls to a protected entry with a call already queued.
- select statements.
- Task termination.

For a full and detailed list refer to *Guide for the use of the Ada Ravenscar Profile in high integrity systems* [1].

3.2 NXT Ada drivers

The NXT drivers developed by AdaCore are completely coded in Ada. These drivers are based on those of the LeJOS Project. The LeJOS Project is a tiny Java virtual machine ported to the NXT Brick in 2006 (http: //lejos.sourceforge.net).

These drivers have undergone major updates in the last two versions of GNAT GPL for MINDSTORMS (2010 & 2011) so 2010 programs might not compile with the 2011 compiler. Unfortunately, AdaCore does not supply API documentation with the drivers. It is convenient to revise the drivers' code to understand how they work. A full description of the drivers is out of the scope of this guide.

For every Ada NXT program the NXT.AVR package must always be imported even if its functions are not required. The body of this package contains a periodic task called Pump, with the highest priority, executed every 20 ms, that handles the co-processor communications (explained in subsection 2.2) using a circular buffer. By adding a with-clause to the main program and importing NXT.AVR the execution of this task within the program is guaranteed. It is also advisable to import NXT.Display and NXT.Last_Chance for exception handling.

High-level access to motors and sensors is available through a series of object oriented interfaces that provide a tagged type, a constructor function and some operations. NXT.Motors and NXT.I2C_Sensors packages provide abstract types and primitive operations. This object oriented structure eases extending the code with new drivers for third-party sensors. For AVR connected peripherals (analog sensors, motors, buttons, etc.) the low-level package NXT.AVR can also be used.

Note that these drivers provide user-transparent button debouncing through the NXT.Filtering package.

Both AVR and Bluetooth interfaces perform checksum analysis for all data exchanged with the main processor to discard inconsistent data.

When using the concurrency features available with the Ravenscar profile it must be considered that the display and AVR drivers do not implement a thread-safe environment. LCD data and the circular buffer with the outgoing messages to the AVR are defined as global variables with no access control. For concurrent access to the display the NXT.Display.Concurrent package provided can be used. For AVR concurrent access a thread-safe solution must be provided by the user to avoid race conditions when calling Power_Down, Set_Power and Set_Input_Power procedures. Notice, that because of the periodic task that handles ARM-AVR communications, every time a motor is used or a power down to the NXT is set, race condition issues are present. The 2010 GNU/Linux GNAT version provided modified drivers that addressed this issue but since the 2011 GNU/Linux version changed its interface the solution has not yet been adapted.

4 Development Environment

4.1 Tools overview

A cross-compiler toolchain is a set of tools (essentially a compiler, an assembler and a linker) that create executable code for a platform, in this case the NXT main processor (ARMv3 architecture), other than the one on which the tools run, that is, GNU/Linux x86. Cross-compiler toolchains are used to compile code for a platform upon which it is not feasible to do the compiling. AdaCore has ported the GNAT compiler toolchain to the ARM architecture by porting part of the LEON-based Open Ravenscar Real-Time Kernel (ORK+) [1] developed by a team of the Department of Telematics Engineering from the Technical University of Madrid (DIT/UPM) [6].

4.2 Compiling a program

The NXT's original firmware for the main processor is completely removed (this invalidates the warranty) and replaced by a binary image of the user's Ada application that is executed from RAM. Flash memory is not used. This means that every time a program is executed it must first by uploaded to RAM.

Instead of using the widespread ELF as executable file format the EABI format is used by the GNAT cross-toolchain. EABI has been created as a common binary interface so that object code and libraries created with one toolchain can be linked to a project created with a different one.

To generate an executable NXT file from the user's Ada application the GNAT cross-toolchain needs first to compile and then link to RAM all compiled code using kernel_samba.ld linker script. The code that needs to be compiled is the user's Ada code, the run-time system, the Ada NXT required drivers, nxt main() C function (main.c), a low-level routine to initialise the system (init.s), a low-level interrupt handler routine (irq.s), a vector table that is remapped to RAM (vectors.s) by init.s and the elaboration code generated by the *GNAT binder*.

A *GNU make* script (Makefile.inc) is in charge of building the binary image that is uploaded. This script compiles the run-time libraries every time since precompiled library units are not used.

4.3 Uploading a program

With no firmware, when the orange button of the NXT Brick is pressed the ARM main processor executes the default Boot Program (SAM-BA Boot Assistant) located in the first two sectors of the Flash memory. The SAM-BA Boot Assistant supports serial communications through the USB Device Port.

LibNXT is a utility library for communicating with the NXT Brick from a POSIX-compliant host computer using USB. When the ARM processor is in SAM-BA mode, *LibNXT* is able upload the binary image file of the NXT executable to RAM and then execute it. For Windows host platforms, the Atmel SAM-BA software is available.

5 Vehicle Prototype

This section describes the steps to have a working NXT vehicle prototype using Ada [2].

5.1 Functionality

The vehicle has a front castor wheel, free to turn, and two back wheels, each driven by an independent motor. To control the vehicle a hardwired joystick made with a touch sensor to start/stop drive and a motor encoder to control operation is used. Depending on the angle of the joystick encoder, different speed commands are sent to the vehicle motors, thus controlling vehicle motion, see figure 4.

[1] ORK+ is an open source real-time kernel that implements the Ravenscar profile for the GNAT compiler system on a bare LEON2 processor.

[2] Example modified from Bradley et al. [7].

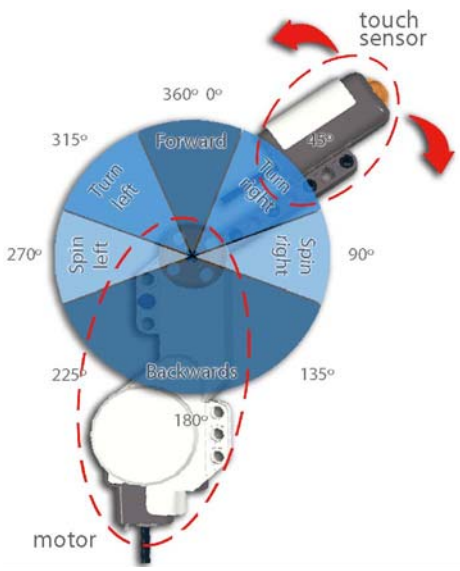

Figure 4 Vehicle's joystick

5.2 Design and assembly

Next step is to assemble a prototype that achieves the above mentioned functionality. The best way to do so, especially if dealing with a complex design, is to model it using a CAD tool. LEGO offers a freeware software to develop NXT models, *LEGO Digital Designer* [3] [8]. The vehicle prototype for this guide was modelled with *LDD*, see figure 5.

Although it can initially be somehow frustrating, using this kind of tools decreases assembly time by allowing the development of several prototypes. It lists the bricks used and generates a step-by-step building guide for the model. Figure 6 shows the vehicle prototype fully assembled using the generated building guide from the LDD model.

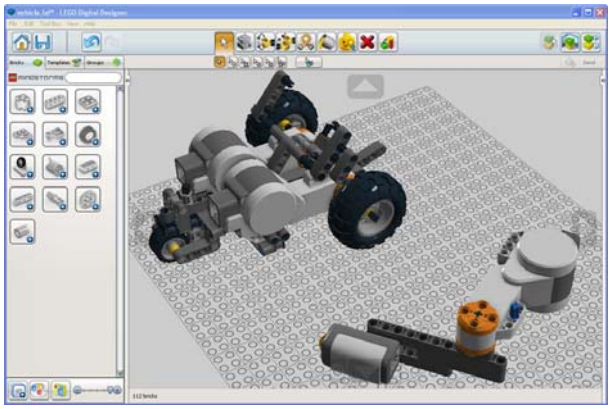

Figure 5 LDD model for the vehicle prototype

5.3 Software Architecture

The following are the tasks involved in the software architecture of the vehicle prototype:

- **Control Task**: Periodic task that executes every 20 ms. It checks if the touch sensor is pressed (a 20 ms period to detect a man operated touch sensor is considered sufficient). In case it is, it gets the value of the joystick motor encoder to determine the speed commands that are then stored in the circular buffer. These speed commands are later sent to the AVR by the Pump task. The Control_Task task takes the position of the joystick motor at the beginning of its execution as reference point. It also checks if the orange button is pressed to switch off the NXT Brick.
- **Display Task**: Periodic task that executes every 500 ms with a lower priority than Control_Task. This task shows the joystick's position, the execution time and the battery's mV on the LCD screen.
- **Background procedure**: This is just a background procedure that executes every time the ARM processor is free.

Although the application performs as expected, the circular buffer global variable used for the ARM-AVR communications is not thread-safe and a race condition exists. This race condition may or may not happen, and if it happens, it does not necessarily mean the performance of the vehicle will be affected. Nevertheless, it is not a good programming practice to rely on non controlled access to a global variable.

There is a thread-safe vehicle version using the 2010 modified AVR drivers that can be downloaded from http://polaris.dit.upm.es/str/projects/mindstorms/2010.

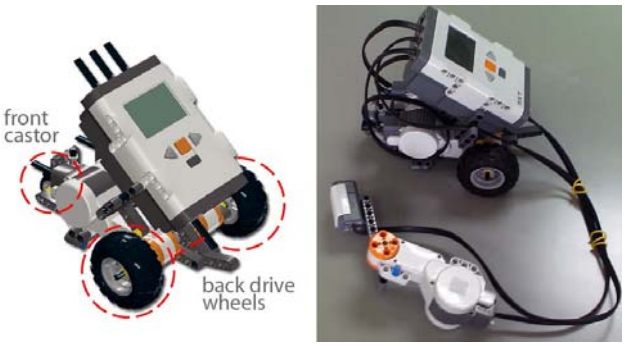

Figure 6 Vehicle prototype fully assembled

5.4 Software Implementation

Three compilation units are used for the Ada vehicle application: The main procedure (vehicle.adb) that calls Background procedure, a package declaration (tasks.ads) and its body (tasks.adb). The Tasks package includes the two control tasks (Control_Task and Display_Task), the empty procedure (Background) and some auxiliary functions. Listing 1 shows a fragment of tasks.adb containing the declaration of the two tasks and the background procedure. When declaring a task, besides using pragma Priority to establish the static priority, pragma Storage_Size is used. Pragma Storage_Size specifies the amount of memory to be allocated for the task stack. Notice that this pragma is required because of the small amount of memory available, 64KB of RAM memory. The stack size

[3] This software is available for Windows and Mac OS. LDraw and LeoCAD are other CAD software alternatives.

must not be exceeded. If it does, a Storage_Error will be raised. If this Storage_Size pragma is not used, a compiling error about RAM overflowing could be prompted.

It must be remembered that the clock resolution defined by the run-time system is of 1 ms.

```
----------------------------
-- Background task --
----------------------------
procedure Background is
begin
    loop
        null ;
    end loop;
end Background;
--------------
-- Tasks --
--------------
task Control_Task is
    pragma Priority (System.Priority ' First + 2);
    pragma Storage_Size (4096);
end Control_Task;

task Display_Task is
    pragma Priority (System.Priority ' First + 1);
    pragma Storage_Size (4096);
end Display_Task;
```

Listing 1: Specification of tasks.

6 Debugging Solution

A remote debugger is an extremely useful tool for an embedded system developer. It can drastically decrease development time. There is no open source Ada/C debugging solution for the NXT. In this section we describe a way to remotely debug Ada/C programs for the NXT using the *GNU debugger* (*GDB*) and the ARM EmbeddedICE (In-circuit Emulator) technology. The ARM EmbeddedICE is a JTAG [4]-based debugging channel available on the ARM main processor. Debugging the NXT from a host computer through the available JTAG interface is therefore possible. RAM and Flash programming is also available using this method.

This solution has been adapted to work on GNU/Linux x86 hosts but it could be easily ported to a Windows platform.

6.1 Overview

The JTAG-based debugging channel provides real-time access to memory addresses and data dependent watchpoints, step-by-step execution, full control of the central processing unit and other related debugging features. It requires no use of memory unlike debugging monitor solutions.

[4] JTAG, as defined by the IEEE Std.-1149.1 standard, is an integrated method for testing interconnects on printed circuit boards (PCBs) that are implemented at the integrated circuit (IC) level.

The ARM featured EmbeddedICE-compatible macrocell from the NXT includes an ARM7 core, a small amount of control logic, a TAP [5] (Test Access Port) controller for the JTAG interface and an EmbeddedICE macrocell, see figure 7. This EmbeddedICE macrocell has two real-time watchpoint registers as well as control and status registers. Each watchpoint register can be configured as a watchpoint for data access or a breakpoint for instruction fetch. If a match occurs between the values programmed into the watchpoint registers and the values of the address bus and data busses or some specific control signal, the ARM7 core ceases to read instructions from the data bus and isolates itself from the memory system entering debug state. Access to the processor's state and memory system is then possible through the JTAG interface using the TAP controller.

GDB provides the remote serial protocol (RSP) for remote debugging. RSP is a *GDB* protocol used to send debugging commands through a serial or Ethernet link. Using a localhost TCP connection on the developer's host computer an *OpenOCD* daemon processes the commands issued by *GDB*.

OpenOCD (The Open On-Chip Debugger) is an open source tool initially developed by Dominic Rath as part of his diploma thesis at the University of Applied Sciences Augsburg [9]. This software provides debugging, in-system programming and boundary-scan testing for embedded targets such as the NXT. *OpenOCD* essentially allows *GDB* to talk through a JTAG adapter to the EmbeddedICE-compatible macrocell on the NXT.

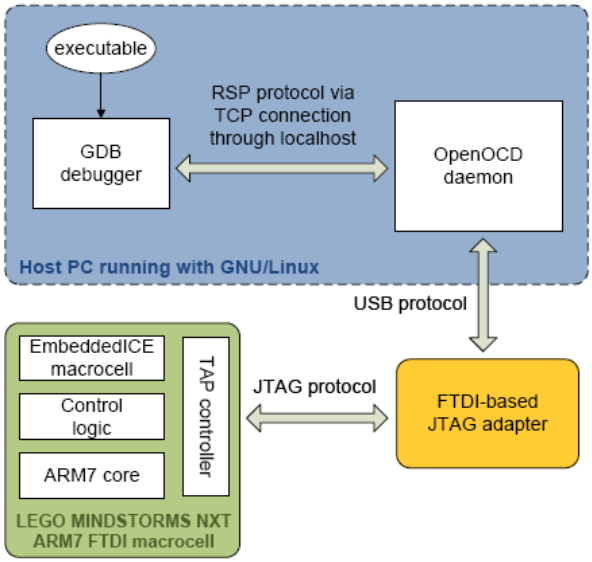

Figure 7 ICE debugging solution for NXT

A JTAG adapter is a piece of hardware that connects the host computer with the JTAG interface of the remote target. The JTAG adapter is in charge of adapting the serial

[5] a TAP is the core of the JTAG standard. It is a finite state machine that controls JTAG operations.

electric signalling received from *OpenOCD*, using, in this case, an FTDI [6] chip, to send the JTAG operations to the TAP controller. Figure 7 shows the debugging scheme.

6.2 Modifying the NXT Brick

To connect *GDB* in the host computer with the JTAG interface of the NXT a JTAG adapter is required. Also, The NXT Brick PCB has the provision for mounting a JTAG connector but this has not been mounted to save cost. The NXT Brick must be opened in order to access the JTAG interface. Note that by performing this modification warranty will be lost.

6.2.1 FTDI-based JTAG adapter

An FTDI-based JTAG adapter that is both compatible with *OpenOCD* and the main processor of the NXT (AT91SAM7S256) is required. For this guide the ARM-USB-TINY-H adapter by Olimex (http://www.olimex.com) was used. *Open On-Chip Debugger: OpenOCD User's Guide* [10] offers other vendor options.

6.2.2 Tools and materials

- Small Philips head screwdriver.
- Fine wire cutter.
- Wire stripper.
- Soldering iron with a fine tip and solder.
- De-soldering pump.
- Magnifying glass.
- Drill with 4 mm diameter bit.
- Digital multimeter.
- 20 pin 2.54 pitch ribbon cable male connector (ARM JTAG connector).
- 30 SWG single core polyurethane insulated cable.

6.2.3 NXT Brick disassembly

Take out the battery pack or batteries to gain access to the four Philips head screws. Unscrew them and remove the front cover. Remove the silicon rubber buttons' assembly.

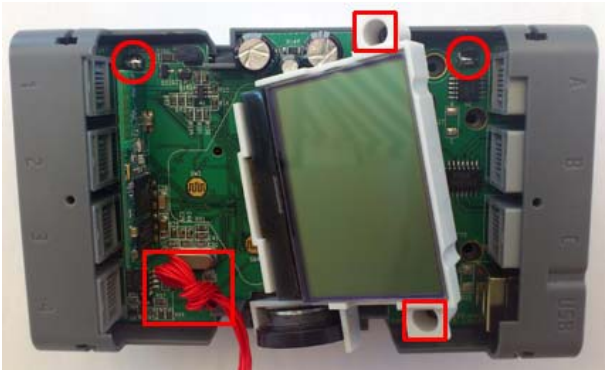

Figure 8 NXT without front cover

Find the two screws that hold down the LCD display, located on each side of it (the two small squares of figure 8). Loosen these screws and carefully lift the LCD display to get access to the battery terminals that are soldered to the main PCB. Note that the LCD display cannot be removed from the PCB board on some models.

Once the two display screws have been removed the two battery terminals must be de-soldered (the two small circles of figure 8). To do this, remove the solder with the soldering iron and the de-soldering pump. When the terminals are free of solder separate the PCB from the battery case and remove the input and output connector supports. Note that there is a small silicone rubber push-button between the battery case and the PCB.

6.2.4 JTAG connection

Since there was no short delivery 1.27 pitch connectors at the time, the hard-wired option presented below was used.

Cut 8 equal lengths, at least 100 mm, of the single core cable and strip 3 mm of insulation on one side. Identify both ends with an indelible marker. The JTAG interface (J17 on the PCB) is located below the loudspeaker beside the quartz crystal (the big square of figure 8). Pin 1 has a square pad and the remaining pins have round pads. Insert one by one the stripped ends of the 8 cables in pins 1 -8 and solder them to the board. This type of wire is used because, unlike PVC insulation, it supports high temperatures (155ºC) and makes soldering easy. With the magnifying glass inspect each solder for bridges between pins. See left picture from figure 9 for the final result.

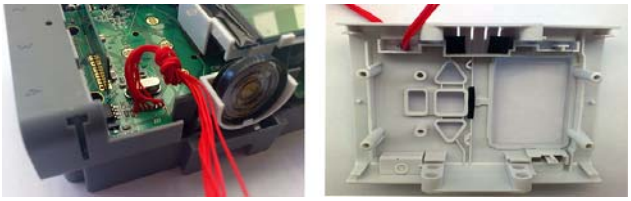

Figure 9 Soldered JTAG interface & front cover drilled hole

Drill a 4 mm hole on the front cover of the NXT Brick directly above the J17 connection as shown in the right picture of figure 9. As a strain relief bundle the eight wires together and tie a knot with them 20 mm from the PCB. Take them through the hole of the front cover and cut them to length for the connection to the ribbon cable connector according to figure 10. As the wire used has a smaller gauge than the connector it is advisable to solder the connections after inserting them. Therefore, strip the wires, insert them and solder them. Try to use as little solder as possible to allow inserting the header in the connector.

[6] Hardware solution to interface with USB peripherals.

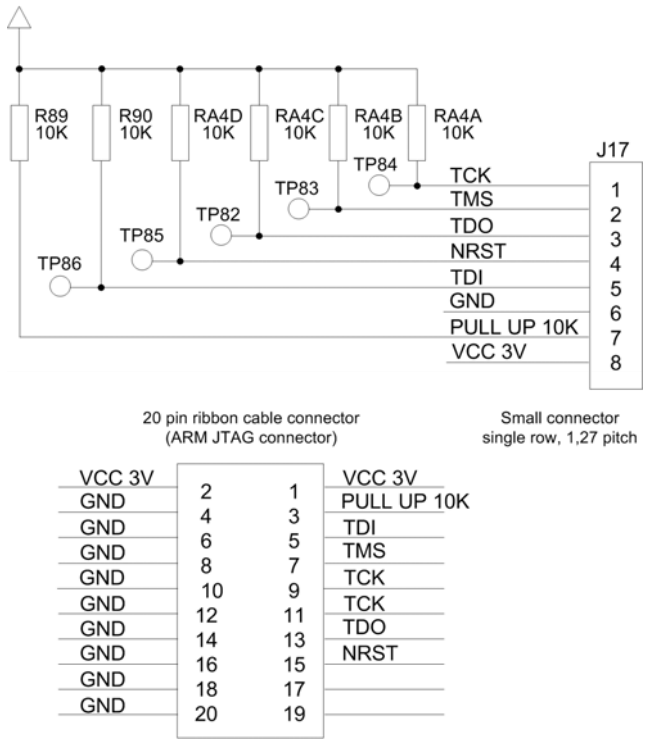

Figure 10 NXT JTAG hardware schematic

Note that the GND connection is only connected to pin 6 because the JTAG adapter used has all the GND pins internally connected.

6.2.5 Testing the connections

Locate on the NXT Brick PCB resistor R89, check for continuity with the multimeter in Ω between the top of R89 and pin 2 of the ribbon cable connector (VCC 3V). Check that the other end of R89 is connected to pin 3 of the ribbon cable connector (PULL UP 10K). Next, check the GND connection between pin 6 of the ribbon cable connector and the negative battery terminal PCB connection (J5). Locate test points TP82-TP86 on the solder side of the PCB and check with the multimeter for continuity between them and the corresponding pins of the ribbon cable connector. Also check for short-circuits between connections.

Finally, once the connections have been checked, re-assemble the NXT Brick.

For a more graphical guide on the modification of the NXT Brick refer to *Installing the JTAG connector* [11].

6.3 A debugging session

In order to remotely debug programs under GNU/Linux libusb-0.1, libusb-dev, libftdi1 and libftdi-dev are required. The FTDI module with the JTAG adapter information will probably have to be loaded also, once it is plugged in:

```
$ sudo modprobe -v ftdi_sio vendor=0x... product=0x...
```

When the NXT has no firmware the orange button must be pressed. Then, when a clicking sound is heard, the JTAG adapter must be plugged to the NXT. Next, arm-eabi-openocd must be run with a specific configuration script:

```
$ arm-eabi-openocd -f debug-ram.cfg
```

This configuration file is a setup for *OpenOCD* that establishes communications with the NXT EmbeddedICE macrocell. The script usually contains the daemon configuration that establishes communications with *GDB*, the configuration for the adapter, the board, the target and some init commands. JTAG adapter vendors usually provide this *OpenOCD* script and in case they do not, the share/openocd/scripts folder from the install directory contains generic configuration files. For further information refer to *Open On-Chip Debugger: OpenOCD User's Guide* [10].

When *OpenOCD* handshakes with the NXT successfully *GDB* must be run with the executable as parameter, not with the binary image:

```
$ arm-eabi-gdb executable_name
```

Any breakpoints should be added at this point. After, the gdbinit script, see listing 2, must be run:

```
gdb> source gdbinit
```

Cross-debugging is now possible.

```
# Init command
target remote localhost:3333

# OpenOCD command to halt the processor
# and wait
monitor soft_reset_halt

# OpenOCD command to select the core state
monitor arm core_state arm

# set flash wait state (AT91C_MC_FMR)
monitor mww 0xffffff60 0x00320100

# watchdog disable (AT91C_WDTC_WDMR)
monitor mww 0xfffffd44 0xa0008000

# enable main oscillator (AT91C_PMC_MOR)
monitor mww 0xfffffc20 0xa0000601

# wait 100 ms
monitor sleep 100

# set PLL register (AT91C_PMC_PLLR)
monitor mww 0xfffffc2c 0x00480a0e

# wait 200 ms
monitor sleep 200

# set master clock to PLL (AT91C_PMC_MCKR)
monitor mww 0xfffffc30 0x7

# wait 100 ms
monitor sleep 100

# enable user reset AT91C_RSTC_RMR
monitor mww 0xfffffd08 0xa5000401

# force a peripheral RESET AT91C_RSTC_RCR
monitor mww 0xfffffd00 0xa5000004

# toggle the remap register to place RAM
# at 0x00000000
monitor mww 0xffffff00 0x01

# set the PC to 0x00000000
```

```
monitor reg pc 0x00000000

# enable use of software breakpoints
monitor gdb_breakpoint_override soft
monitor arm7_9 dbgrq enable

# upload the application
load

# resume execution from reset vector
continue
```

Listing 2 *GDB* init script

This *GDB* script basically sets the ARM processor to execute the application and set some debugging features. The script used is a modified version of that presented in *Using Open Source Tools for AT91SAM7S Cross Development* by James P. Lynch [12].

7 Conclusions

This guide shows the basics for Ada development using LEGO MINDSTORMS NXT. The Ravenscar profile run-time system offers concurrency Ada programming while making possible a schedulability analysis of the system. Ada development on the NXT presents a whole perspective of an embedded system with real-time constraints.

At a reasonable price the NXT kit offers all kinds of sensors and mechanisms to work with, even custom-made sensors can be developed.

Development and sharing of Ada projects with the NXT would be of great interest, in the same way as other complex models like Rubik's cube solvers, Segway robots, scanners, etc. have been developed using other programming languages and shared.

The Ada community is encouraged to use this development platform that, besides the fun, can be an interesting teaching asset.

It is important to note that all of the tools used, except *LDD*, are open source and therefore there is no dependence on software vendors. All of the source code is available and can by modified.

Acknowledgements

The authors would like to thank AdaCore for their work adapting the Ravenscar run-time system and developing the Ada drivers for the LEGO MINDSTORMS NXT platform.

References

[1] Burns A., Dobbing B., Vardanega T. Guide for the use of the Ada Ravenscar Profile in high integrity systems. Ada Letters 2004 June;XXIV:1–74. Available from: http://doi.acm.org/10.1145/997119. 997120.

[2] LEGO. LEGO MINDSTORMS NXT Hardware Developer Kit; Version 1.00. Available from: http://mindstorms.lego.com.

[3] LEGO. LEGO MINDSTORMS NXT ARM7 Bluetooth Developer Kit; Version 1.00. Available from: http://mindstorms.lego.com.

[4] AdaCore. GNAT Pro User's Guide, Supplement for High-Integrity Edition Platforms; 2011. The GNAT Ada Compiler. GNAT GPL Edition, Version 2011 Document revision level 175263.

[5] Std. 8652:1995/Amd 1:2007 — Ada 2005 Reference Manual. Language and Standard Libraries; 2007. Published by Springer-Verlag, ISBN 978-3-540-69335-2.

[6] de la Puente J. A., Ruiz J. F., Zamorano J. An Open Ravenscar Real-Time Kernel for GNAT. In: Proceedings of the 5th Ada-Europe International Conference on Reliable Software Technologies. Ada-Europe '00. London, UK: Springer-Verlag; 2000. p. 5-15. Available from: http://portal.acm.org/citation.cfm?id=646579.697613.

[7] Bradley P. J., de la Puente J. A., Zamorano J. Real-time system development in Ada using LEGO MINDSTORMS NXT. In: Proceedings of the ACM SIGAda annual international conference on SIGAda. SIGAda'10. New York, NY, USA: ACM; 2010. p. 37-40. Available from: http://doi.acm.org/10.1145/1879063.1879077.

[8] LEGO. LEGO Digital Designer 4.1 User Manual; 2011. Available from: http://ldd.lego.com.

[9] Rath D. Open On-Chip Debugger. Design and Implementation of an On-Chip Debug Solution for Embedded Target Systems based on the ARM7 and ARM9 Family. University of Applied Sciences Augsburg; 2005.

[10] Brownell D. Open On-Chip Debugger: OpenOCD User's Guide; 2011. Available from: http://openocd.berlios.de.

[11] IAR. Installing the JTAG connector. IAR Kick-Start for LEGO MINDSTORMS NXT; 2009. Available from: http://www.iar.com/website1/1.0.1.0/1483/1.

[12] Lynch J. P. Using Open Source Tools for AT91SAM7S Cross Development. Grand Island, New York, USA; 2007. Revision C.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o K.U. 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

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

Ada-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

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: ada@white-elephant.ch
URL: www.ada-switzerland.ch