ADA USER JOURNAL

Volume 32
Number 2
June 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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

The second issue of Volume 32 of the Ada User Journal is being finalized in the exceptional environment of the Ada Connection event, which brings together the 16th International Conference on Reliable Software Technologies – Ada-Europe 2011 and the Ada Conference UK 2011, in the beautiful city of Edinburgh, UK. I hope the Journal readers had the opportunity to experience and enjoy this environment; if not I urge you to consider attending next year's Ada-Europe 2012, which will take place in Stockholm. The Ada-Europe conference is returning to Sweden 14 years after being in Uppsala, in 1998.

As for the highlights of this year's event, I would like to point out not only the scientific program of the conference, as usual with a set of high-quality papers, but also the impressive industrial track of the conference, with a set of high quality presentations (of which we start to present derived papers in this issue). The conference also featured two very interesting panels on *Programming Languages Meet Multi-Core* and *DO178C and Object-Orientation for Critical Systems*, and three keynote talks: Peter Bernard Ladkin (Causalis Ltd) on the *Future of Software Safety Standards*, Pippa Moore (UK CAA) on the topic of *Hippocrates and DO-178B*, and Jeff O'Leary (USA FAA) on *Assuring Software Reliability While Using Web Services and Commercial Products*. A special note also to the pre-dinner presentation by Professor Les Hatton, providing (with a humorous view) a few important remarks on the development of safe software. Undoubtedly a major networking and promotion event for the world-wide Ada community.

As for the contents of this issue, it starts with the information of the News and Calendar sections, by Marco Panunzio and Dirk Craeynest, their respective editors. The Forthcoming Events section provides an announcement of the 2011 SIGAda conference, which will take place next November in Denver, Colorado, USA, and the preliminary call for papers for the 17th International Conference on Reliable Software Technologies – Ada-Europe 2012 that, as noted, will take place June 2012 in Stockholm, Sweden.

The technical part of the issue provides a paper from the Industrial Track of the Ada-Europe 2011 conference, from a group of authors from Eurocopter and ESG, Germany, discussing the Software Product Line approach to the development of the NH90 helicopter avionics software, in Ada 83. To finalise, the Ada Gems section provides a set of gems with the trail Code Archetypes for Real-Time Programming, by Marco Panunzio, from the University of Padua, Italy.

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

Quarterly News Digest

Marco Panunzio

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

Contents

Ada-related Organizations

Submission for "The Ada Way" programming contest

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Tue, 26 Apr 2011 21:35:50 +0000 UTC
Subject: The Ada Way programming contest now accepts submissions
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

Ada-Europe's first Annual Student Programming Contest "The Ada Way" is now accepting submissions.

Last September, Ada-Europe [1] kicked off its annual student programming contest "The Ada Way" [2]. The challenge for the 2010-2011 competition was to build a software simulator of a football match. The submission of entries has now been opened: they will be accepted through May 15th.

The submission shall be made as a single compressed archive of all items listed at the contest web site. Dropbox [3] is used to handle the submission process online. Details are available on the contest web site.

The winning submission shall be announced on the contest web site and at the Ada-Europe 2011 Conference [4], held June 20-24 in Edinburgh, UK. This edition of the contest is sponsored by AdaCore, Atego and Ada-Europe.

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of the Ada programming language in academia, research and industry in Europe. Ada-Europe has member organizations all over the continent, in Belgium, Denmark, France, Germany, Spain, Sweden, Switzerland, as well as individual members in many other countries.

[1] http://www.ada-europe.org/

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

[3] http://www.dropbox.com/

[4] http://www.ada-europe.org/conference2011/

Ada-Belgium Spring 2011 - Debian packaging workshop

From: Dirk Craeynest <dirk@vana.cs.kuleuven.be>
Date: Tue, 10 May 2011 21:37:18 +0000 UTC
Subject: Ada-Belgium Spring 2011 Event incl. Debian packaging workshop
Newsgroups: comp.lang.ada, fr.comp.lang.ada,be.comp.os.linux, be.comp.programming

---

A d a - B e l g i u m S p r i n g 2 0 1 1 E v e n t

Sunday, May 22, 2011, 12:00-19:00

Brussels, Belgium

including at 14:30

2011 Ada-Belgium General Assembly

and at 15:30

Workshop on Creating Debian Packages of Ada Software

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

Announcement

The next Ada-Belgium event will take place on Sunday, May 22, 2011 in Brussels.

For the fourth year in a row, Ada-Belgium decided to organize their "Spring Event", which starts at noon, runs until 7pm, and includes an informal lunch, a key signing party, the 18th General Assembly of the organization, and a workshop on packaging Ada software for Debian hosted by Ludovic Brenta, principal maintainer of Ada in Debian.

Schedule

- 12:00 welcome and getting started (please be there!)
- 12:15 informal lunch
- 14:30 Ada-Belgium General Assembly
- 15:15 key signing party
- 15:30 workshop on creating Debian packages of Ada software
- 19:00 end

Participation

Everyone interested (members and non-members alike) is welcome at any or all parts of this event.

For practical reasons registration is required. If you would like to attend, please send an email before Tuesday, May 17, to Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be> with the subject "Ada-Belgium Spring 2011 Event", so you can get precise directions to the place of the meeting.

If you are a member but have not renewed your affiliation yet, please do so by paying the appropriate fee before the General Assembly (you have also received a printed request via normal mail). If you are interested to become a new member, please register by filling out the 2011 membership application form[1] and by paying the appropriate fee before the General Assembly.

After payment you will receive a receipt from our treasurer and you are considered a member of the organization for the year 2011 with all member benefits[2]. Early renewal ensures you receive the full Ada-Belgium membership benefits (including the Ada-Europe indirect membership benefits package).

As mentioned at earlier occasions, we have a limited stock of documentation sets and Ada related CD-ROMs that were distributed at previous events. Most important are back issues of the Ada User Journal[3]. These will be available on a first-come first-served basis at the General Assembly for current and new members.

[1] http://www.cs.kuleuven.be/~dirk/ada-belgium/forms/member-form11.html

[2] http://www.cs.kuleuven.be/~dirk/ada-belgium/member-benefit.html

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

Informal lunch

The organization will provide food and beverage to all Ada-Belgium members. Non-members who want to participate at the lunch are also welcome: they can choose to join the organization or pay the sum of 15 Euros per person to the Treasurer of the organization.

General Assembly

All Ada-Belgium members have a vote at the General Assembly, can add items to the agenda, and can be a candidate for a position on the Board[4]. See the separate official convocation[5] for all details.

[4] http://www.cs.kuleuven.be/~dirk/ada-belgium/board/

[5] http://www.cs.kuleuven.be/~dirk/ada-belgium/events/11/110522-abga-conv.html

Key Signing Party

Wouldn't it be nice if a majority of people used GPG to sign their email every day so that you could send all non-signed email into the spam bin? To make that dream come true, please join and expand the global Web of Trust [6]!

What you should bring with you:

- an official ID card issued by your national government;
- your GPG key fingerprint (i.e. the output of gpg --fingerprint) on small paper slips; a dozen copies or so should be enough.

What you will go home with:

- signatures from all other participants;
- automatic inclusion in the global Web of Trust;
- the ability to digitally sign or encrypt anything you like.

[6] http://en.wikipedia.org/wiki/Web_of_Trust

Workshop:
Packaging Ada Software for Debian

Debian [7], "The Universal Operating System", is simply the best platform for the enthusiast Ada developer. The features that distinguish Debian from the rest are:

- a binary distribution that avoids the need to recompile Florist, ASIS, GtkAda and all other Ada packages;
- a large number of packages intended for Ada developers;
- a clear and consistent policy[8] making all packages integrate seamlessly and interoperate;
- outstanding support for the Ada part of the GNU Compiler Collection (GCC) with unique innovations like libgnatvsn and libgnatprj not found anywhere else;
- backports of bug fixes from the bleeding edge of GCC development into the safe and stable compiler used for all Debian packages;
- support for more hardware architectures than any other Ada distribution: alpha, amd64, hppa, i386, ia64, kfreebsd-i386, powerpc, s3980 and sparc (with mips, mipsel and ppc64 added recently).
- a choice between "stable", "testing" and "unstable" versions of Debian to suit personal preferences;
- Debian is the mother of Ubuntu, Knoppix and dozens of other distributions which sometimes incorporate the Ada packages.

The goal of the workshop is to help people participate in this effort to bring even more Ada software to Debian, or to help maintain the existing packages.

What you should bring with you:

- your computer, already installed with Debian unstable or with an unstable chroot already created (see below);
- network cables (or WiFi already configured);
- monitor and keyboard, if your computer is not a laptop;
- power cables;
- some Ada software you would like to see in Debian but is not there (not necessarily software that you wrote; any software with a license permitting redistribution in source and binary form will do).

Note 1: if your computer does not run Debian as its main operating system, you can install Debian in a virtual machine (VMWare or other), in a jail on a FreeBSD system (Debian kfreebsd-i386), or in a chroot on any other distribution. Danny Beullens will offer help and assistance to those who would like to install Debian in a VMWare virtual machine.

Note 2: if you would like to install Debian as your main operating system but are uncomfortable doing so by yourself, please get in touch with your nearest Linux User Group (e.g. http://www.bxlug.be in Brussels).

What Ludovic Brenta will do for you:

- set up a local Debian mirror, so you can install or upgrade packages necessary for Ada package development;
- explain how to package Ada software for Debian;
- help you package your own program or library;
- answer questions about GNAT, GCC, Debian, etc.;
- if your package is suitable for inclusion in Debian, sponsor it for you.

What you will go home with:

- your own .deb packages installed on your computer;
- better understanding of how packaging works;
- better understanding of the Debian Policy for Ada;
- if your package is suitable, your name on the Debian Package Tracking System and your package on the next Debian DVD or CDROM distribution.

[7] http://www.debian.org/

[8] http://people.debian.org/~lbrenta/debian-ada-policy.html

Directions

To permit this more interactive and social format, the event takes place at private premises in Brussels. As instructed above, please inform us by e-mail if you would like to attend, and we'll provide you precise directions to the place of the meeting. Obviously, the number of participants we can accommodate is not unlimited, so don't delay…

Looking forward to meet many of you in Brussels!

Dirk Craeynest, President Ada-Belgium

Dirk.Craeynest@cs.kuleuven.be

Acknowledgements

We would like to thank our sponsors for their continued support of our activities: AdaCore, Katholieke Universiteit Leuven (K.U.Leuven), and Université Libre de Bruxelles (U.L.B.).

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]

SIGAda 2011 Call for Papers

From: Greg Gicca <greggicca@gmail.com>
Date: Wed, 23 Mar 2011 07:57:27 -0700 PDT
Subject: SIGAda 2011 conference Call for Papers
Newsgroups: comp.lang.ada

Group,

An important announcement for Ada and high reliability software developers:

SIGAda 2011

Denver, Colorado, USA

November 6 - 10, 2011

Call for Technical Contributions

ACM Annual International Conference on Ada and Related Technologies:

Engineering Safe, Secure, and Reliable Software

Submission Deadline: June 30, 2011

Sponsored by ACM SIGAda

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

SUMMARY: Reliability, safety, and security are among the most critical requirements of contemporary software. The application of software engineering methods, tools, and languages all interrelate to affect how and whether these requirements are met. Such software is in operation in many application domains. Much has been accomplished in recent years, but much remains to be done. Our tools, methods, and languages must be continually refined; our management process must remain focused on the importance of reliability, safety, and security; our educational institutions must fully integrate these concerns into their curricula.

The conference will gather industrial and government experts, educators, software engineers, and researchers interested in developing, analyzing, and certifying reliable, safe, long-lived, secure software. We are soliciting technical papers and experience reports with on these development topics. A direct relationship or comparison with the Ada language is preferred.

[…]

HOW TO SUBMIT: Send contributions by June 30, 2011, in Word, PDF, or text format as follows:

Technical Articles, Extended Abstracts, Experience Reports, and Panel

Session Proposals to: Program Chair,

Lt. Col. Jeff Boleng (jeff.boleng@usafa.edu)

Tutorial Proposals to: Tutorials Chair, Dr. Robert Pettit

(rpettit@gmu.edu)

Industrial Presentations Proposals to: Industrial Committee Chair,

Prof. Liz Adams (adamses@cs.jmu.edu)

[…]

ANY QUESTIONS?

Please submit any questions to the conference chair, Ricky E.(Ranger) Sward (rsward@mitre.org).

Ada-related Resources

How to build a GNAT cross-compiler for QNX

From: Piotr Trojanek <piotr.trojanek@gmail.com>
Date: Sat, 23 Apr 2011 14:17:34 -0700 PDT
Subject: GNAT Ada cross compiler for QNX
Newsgroups: comp.os.qnx, comp.lang.ada

Dear all,

recently I have completed an updated document about how to build a GNAT Ada Linux->QNX cross-compiler:

https://github.com/ptroja/qnx-cross/wiki/Ada-Annex-E-with-PolyORB-under-QNX

In addition instructions about the cross build of the the PolyORB distributed middleware (implementation of the Distributed System Annex) have been added.

Hope this will be helpful to other porters and of course all the QNX and Ada community.

AdaCore @ YouTube

From: Thomas Løcke <tl@ada-dk.org>
Date: Fri, 25 Mar 2011
Subject: AdaCore @ YouTube
URL: http://ada-dk.org/?page=news&news_id=290

I'm one of those people who enjoy tech-talks on video, so when I found out that AdaCore have a channel @ YouTube, it made my day.

Yes, I'm that easy to please. :o)

It's a fairly busy channel, with regular updates. Some of the subjects dealt with are:

- What is SPARK?
- GPS 5 Auto Highlighting
- Hi-Lite: a Verification Toolkit for Unit Test & Unit Proof

And of course lots more. Be sure to check it out!

[Visit the channel at http://www.youtube.com/adacore05 —mp]

Safe and Secure Software with Ada 2005

From: Thomas Løcke <tl@ada-dk.org>
Date: Fri, 18 Mar 2011
Subject: Safe and Secure Software With Ada 2005
URL: http://ada-dk.org/?page=news&news_id=285

John Barnes, who is probably known to most Ada programmers, have written a booklet on how to use Ada 2005 to build safe and secure software.

It is, to no surprise, a good read.

The booklet is divided into 13 chapters, each in its own PDF file:

- Introduction
- Chapter 1 – Safe Syntax
- Chapter 2 – Safe Typing
- Chapter 3 – Safe Pointers
- Chapter 4 – Safe Architecture
- Chapter 5 – Safe Object Oriented Programming
- Chapter 6 – Safe Object Construction
- Chapter 7 – Safe Memory Management
- Chapter 8 – Safe Startup
- Chapter 9 – Safe Communication
- Chapter 10 – Safe Concurrency
- Chapter 11 – Certified Safe with SPARK
- Chapter 12 – Conclusion

The entire booklet is available here.

[http://www.adacore.com/home/ada_answers/ada_2005/safe_secure —mp]

I'd like to end this news item with a quote from the booklet:

> In terms of software, the languages Ada and C have very different attitudes to freedom. Ada introduces restrictions and checks, with the goal of providing freedom from errors. On the other hand C gives the programmer more freedom, making it easier to make errors. One of the historical guidelines in C was "trust the programmer". This would be fine were it not for the fact that programmers, like all humans, are frail and fallible beings. Experience shows that whatever techniques are used it is hard to write "correct" software. It is good advice therefore to use tools that can help by finding bugs and preventing bugs. Ada was specifically designed to help in this respect. There have been three versions of Ada – Ada 83, Ada 95 and now Ada 2005. The purpose of this booklet is to illustrate the ways in which Ada 2005 can help in the construction of reliable software, by illustrating some aspects of its features. It is hoped that it will be of interest to programmers and managers at all levels.

[…]

Tutorial on Ada streams

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Mon, 28 Feb 2011 09:00:23 -0800 PST
Subject: Ann: Little tutorial about streams
Newsgroups: comp.lang.ada

Dear all,

remembering my initial difficulties with streams (I self-taught Ada, using few tutorials and lots of experiments before landing to the RM), I decided to write a page (my first one, so be patient :-) of the Wikibook with a little stream tutorial

http://en.wikibooks.org/w/index.php?title=Ada_Programming/Input_Output/Stream_Tutorial

As said in the page, the goal is to give to the reader an intuitive idea about how streams work so, in order to not hide the forest with too many leaves, some of the finest details have been omitted. A subpage of the above page has a fairly complex example that is not complete yet, but I plan to complete it soon.

The page is not linked yet with the book body. I was thinking to add a link to it in the "Input Output" page as soon as the subpage with the example is in a good state.

[…]

Ada Tutor tutorial

From: Jeffrey R. Carter <pragmada@pragmada.x10hosting.com>
Date: Wed, 04 May 2011 14:25:00 -0700
Subject: Ada Tutor Available from PragmAda Software Engineering
Newsgroups: comp.lang.ada

John Herro's Ada Tutor Ada-95 tutorial is now available from PragmAda Software Engineering:

http://pragmada.x10hosting.com/

New website Ada-Poland.org

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sat, 19 Mar 2011 03:09:21 -0700 PDT
Subject: Ada-Poland.org
Newsgroups: comp.lang.ada

I'm pleased to announce the mini-launch of new website:

http://www.ada-poland.org/

The purpose of this site is to gather mutual contacts to people and companies in Poland who are interested in the use of Ada in their projects, promotion of their own work and representation in the international community.

This simple web page is a seed and currently contains some useful basic links, as well as the following important information:

Ada-Poland.org is currently *not* associated to Ada-Europe, but its intent is to get the (snow)ball rolling and build the critical mass that will eventually make it happen.

Please feel free to promote this site among Polish software engineers.

Ada-related Tools

Mathpaqs, February 2011

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Mon, 21 Feb 2011 10:43:29 -0800 PST
Subject: Ann: Mathpaqs, release Feb. 2011
Newsgroups: comp.lang.ada

Hello, there is a new release of Mathpaqs @ http://sf.net/projects/mathpaqs/

What's new:

- *new* Discrete_random_simulation package (this is for simulating any discrete random distribution)
- updated the Finite_distributed_random function (this is for simulating a random distribution for an enumerated type)
- cleanup of ConjGrad (Conjugate gradient iterative methods for solving the matrix equation Ax=b)

[see also "Mathpaqs - November 28, 2010" in AUJ 32-1 (Mar 2011), p.11 —mp]

Ada 2005 Math Extensions 20110320

From: Simon Wright <simon.john.wright@gmail.com>
Date: Sun, 20 Mar 2011 10:24:50 -0700 PDT
Subject: ANN: Ada 2005 Math Extensions 20110320
Newsgroups: comp.lang.ada

This release has just been made at Sourceforge:

https://sourceforge.net/projects/gnat-math-extn/files/20110320/

Changes in this release:

- An additional overloaded procedure Eigensystem returns the generalized eigenvalues and eigenvectors of a pair of non-hermitian complex matrices.
- The program Test_Extensions is renamed to Demo_Extensions, and includes a demonstration of the real generalized eigensystem code.
- The library project file is renamed src/gnat_math_extensions.gpr.

[…]

[see also "Ada 2005 Math Extensions" in AUJ 32-1 (Mar 2011), p.11 —mp]

Ada support in *BSD and Android distributions

From: John Marino <dragonlace.cla@marino.st>
Date: Thu, 3 Mar 2011 12:07:22 -0800 PST
*Subject: Ann: DragonLace and Ada Support on *BSD / *Solaris / Android*
Newsgroups: comp.lang.ada

For a long time, support for Ada was poor to non-existent for the four major BSD platforms of FreeBSD, NetBSD, DragonFlyBSD, and OpenBSD.

GNAT would not build from source, although FreeBSD i386 was pretty close. Over the last year, I've developed a substantial set of patches and now GNAT from GCC-4.6 passes the entire ACATS and gnat.dg testsuite without failure on FreeBSD, DragonFlyBSD, and NetBSD on both the AMD64 and i386 platforms. The version is called "GNAT-AUX" and it also builds on OpenBSD with only a single failure the same two platforms.

Three of the BSDs have brand new Ada software in the package/ports system. I would like to give many thanks to Wen Helping for his extensive testing and tweaking of the following software (just) now available in the FreeBSD ports system:

lang/gnat-aux	[GNAT compiler based on gcc-4.6]
devel/gps	[GNAT Programming Studio]
devel/gprbuild-aux	[GPRBuild for GNAT-AUX]
devel/gnatpython	[Needed for AWS test suite]
textproc/xmlada	[XML library needed by GPS and AWS]
x11-toolkits/gtkada	[gtk bindings needed by GPS]
www/aws	[Ada Web Server]

I would like to extend the same gratitude to Matthias Drochner who also spent an incredible effort bringing the same packages to NetBSD and DragonFlyBSD through their mutual package system "pkgsrc":

lang/gnat-aux

devel/gps

devel/gprbuild-aux

devel/gnatpython

textproc/xmlada

x11/gtkada

www/aws

Progress about Ada support the *BSD and *Solaris is found at the http://www.dragonlace.net web site. There's a blog/RSS feed there, and we've started a mailing list that people can use to talk about any of the projects they may have interest in.

I've also successfully built GNAT-AUX which passes flawlessly on OpenSolaris. I also appreciate the Solaris-based Operating system, so support for Illumos based systems will likely arrive in the future. It's not on my immediate priority list, but maybe interest from others will affect that.

OPENBSD

=======

As previously mentioned, GNAT-AUX builds and performs seemingly just fine on OpenBSD, already passing almost every test. However, I do not plan on creating any Ada ports for OpenBSD. There appears to be zero interest for Ada ports for OpenBSD, even from the Ada users I know that are fans of OpenBSD. That said, anybody can just copy the Makefiles from FreeBSD to tweak them for OpenBSD. I won't do it, but I'm happy to give advise to anyone else that wants to tackle this.

DRACO

=====

A long-term project of mine is to graft the GNAT front end of GCC to LLVM by replacing the "GiGi" module. The resulting native Ada compiler for LLVM will be called DRACO. I've spent a fair amount of time on this in the past and currently the project is mothballed, but not for forever.

ANDROID

=======

I've recently built a GNAT-AUX cross-compiler for Android, and verified that helloworld runs on the Android SDK emulator. I'm looking to buy a Honeycomb Android tablet. After that I'll look to fully test GNAT-AUX on the platform and hopefully blaze the trail for creating native Ada applications for Android devices.

There are other projects brewing. As appropriate, details will appear on DragonLace. If you have interest in any of these projects aimed at *BSD, *Solaris, or Android platforms, then feel free to help kick off use of the DragonLace mailing lists and possibly contribute to some of these projects. If nothing else, consider that no less than 3 BSD platforms are now excellent Ada development environments!

[…]

From: John Marino <dragonlace.cla@marino.st>
Date: Wed, 27 Apr 2011 11:23:11 -0700 PDT
*Subject: Ann: GNAT-AUX upgrade in *BSD, includes C++ now*
Newsgroups: comp.lang.ada

GNAT-AUX is heavily patched version of GCC 4.6 which builds perfectly on FreeBSD, NetBSD, and DragonFlyBSD among other platforms.

The first versions published to FreeBSD ports and NetBSD pkgsrc were pre-release. GCC 4.6.0 was released on 25 March, so GNAT-AUX has been synced to this version. In addition, the C++ language is built by default, although this can be switched off.

The FreeBSD port was published today. The NetBSD/DragonFlyBSD pkgsrc package will probably be published within a week as the committer has been busy traveling.

In any case, if you have multiple language (Ada,C,C++) projects and use *BSD, then you definitely want to get this new version. The next release of GNAT-AUX will coincide with GCC 4.6.1 release.

[…]

[See also "Ada support in *BSD distributions" in AUJ 32-1 (Mar 2011), p.13 and "GNAT AUX ported to Android" in the same AUJ issue, p.10 —mp]

Zip-Ada v.40

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Wed, 2 Mar 2011 01:51:48 -0800 PST
Subject: Ann: Zip-Ada v.40
Newsgroups: comp.lang.ada

Hello!

A new version of the Zip-Ada library, @ http://unzip-ada.sf.net/ , is out.

Latest changes are:

- New package Zip.Compress.Deflate, with a first (weak but straightforward) compression technique for the Deflate format - more to come!
- Some improvements in the ReZip and UnZipAda tools

Zip-Ada is a library for handling, decompressing and creating Zip archives.

Some features:

- full sources are in Ada (no binding)
- decompression for all Zip formats up to BZip2
- compression for all Zip formats up to Deflate
- unconditionally portable
- input and output can be any stream (file, buffer,…) for archive creation as well as data extraction.
- task safe
- endian-neutral

URL: http://unzip-ada.sf.net/

The zipada40.zip archive contains:

- The full library sources inside one directory, Zip_Lib, in pure Ada 95+
- Some command-line demo / tools:
  - o ZipAda, a zipping tool
  - o UnZipAda, an unzipping utility
  - o Comp_Zip, compares two Zip files
  - o Find_Zip, searches a text string through contents of a Zip file
  - o ReZip.adb, optimizes compression of Zip archives

[see also "Zip-Ada v.38" in AUJ 31-1 (Mar 2010), p.12 —mp]

Excel Writer v.07

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sun, 6 Mar 2011 07:22:57 -0800 PST
Subject: Ann: Excel Writer v.07
Newsgroups: comp.lang.ada

Hello!

There is a new release of Excel Writer.

What's new:

- Cell merging on a row has been implemented, at least in a very light way
- Some unexpected display of built-in numeric formats (%'s) on some versions of Excel with some language settings has been fixed

URL for download:
http://excel-writer.sf.net

[…]

[see also "Excel Writer v.05" in AUJ 31-1 (Mar 2010), p.13 —mp]

AXMPP, an XMPP implementation for Ada

From: coopht <coopht@gmail.com>
Date: Mon, 7 Mar 2011 13:32:25 -0800 PST
Subject: Announce: AXMPP library. XMPP protocol implementation for Ada.
Newsgroups: comp.lang.ada

Hi all. I'm pleased to announce the first release of AXMPP library.

AXMPP is an implementation of the XMPP protocol for the Ada programming language.

The first release of AXMPP library - AXMPP-0.0.1 includes the following features:

- Secure connection support with agnutsl (ada-binding to gnutls) library.
- Base XMPP protocol implementation:
  - o Roster management support.
  - o Messaging support.
  - o Multi-user chat support.
  - o Presence support.
  - o Resource management support.
  - o Raw IQ Support
- Version extension support.

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

Storage pool with bindings to Apache Runtime Pools library

From: Brad Moore <brad.moore@shaw.ca>
Date: Thu, 24 Mar 2011 08:00:04 -0600

Subject: ANN: Storage pool for Ada 2005 with bindings to Apache Runtime Pools library
Newsgroups: comp.lang.ada

This is the initial release of a storage pool for Ada 2005 called Deepend, that binds to the Apache Runtime Pools library.

Key features

- Pool may deallocate all storage all at once, rather than having to perform Unchecked_Deallocation one object at a time.
- No need to call Unchecked_Deallocation with this pool. It is essentially a NO-OP.
- Provides Subpool capabilities, where a pool object may be a subpool of another pool object. The lifetime of the subpool object extends to the lifetime of the ultimate top-level pool object. Subpools may in turn also have subpools.
- Fast storage management, should be more efficient than garbage collection strategies used in other languages.

The latest stable release and older releases may be downloaded from;

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

For those who want the current development versions of the source they can download using git (http://git-scm.com/) by issuing the following commands

mkdir sandbox

cd sandbox

git clone git://deepend.git.sourceforge.net/gitroot/deepend/deepend

The current development version typically will correspond to the latest stable release, but may at times be unstable when new features are being worked on.

Low-level Bindings to the Apache Runtime Pools library were recently used for a submission to the Computer Language Benchmarks game, binary tree benchmark, and moved Ada into the number 2 spot behind C. On my machine, the Ada version actually runs 10% faster than the C version, but for some reason the benchmark has C ahead of Ada.

It may be that the number of worker threads isn't tuned correctly for the benchmark hardware, or compiler version differences, or other differences related to the target platform.

See

http://shootout.alioth.debian.org/u64q/benchmark.php?test=binarytrees&lang=all

Although the submission code does not use Deepend, the submission code has been reworked to use deepend to see if performance is impacted by using Ada's storage pool mechanism, and no noticeable performance impacts were found.

From: Brian Drummond <brian_drummond@btconnect.com>
Date: Thu, 24 Mar 2011 15:59:48 +0000 UTC
Subject: Re: ANN: Storage pool for Ada 2005 with bindings to Apache Runtime Pools library
Newsgroups: comp.lang.ada

[…]

Great work, and certainly blows the doors off my puny efforts!

You may be right about tuning the number of threads; on my (AMD Phenom) system, my version (#3) gave the same runtime for 4 or 8 tasks, but on the test system (Intel Q6600) 8 tasks was about 10% slower than 4. (The memory footprint was doubled, suggesting memory or cache limitations on the Intel system).

It may be worth posting the Deepend version - either there, or is there a place on Rosetta for it? - as a demonstration of the flexibility of Ada's storage pools.

From: Brad Moore <brad.moore@shaw.ca>
Date: Thu, 24 Mar 2011 15:25:44 -0600
Subject: Re: ANN: Storage pool for Ada 2005 with bindings to Apache Runtime Pools library
Newsgroups: comp.lang.ada

[…]

Thanks for your version also, in particular, the output generation from your version saved me from having to fiddle around with getting the output to come out right.

I actually set the number of workers to 5, which was a bit surprising to me. I believe there are 9 iterations, which is why the number of workers doesnt come out to an even number. On my system, an AMD Quadcore, 5 workers gave me the best time. I was thinking 9 would have been the best number.

It may be that 4 is a better number on their machine. I should maybe ask the maintainers of the benchmarks to try running with 4 to see if that runs any better.

I was thought about posting the Deepend version, (there are actually two versions, one that uses nested access types that relies on Ada's ability to clean up objects when access types get finalized, using the new operator, and the second version that uses calls to Deepend's generic allocate procedure that lets you use a single access type with different pool objects. The reason I decided against posting the result was more that the one that was there involves less source code, and might be better for language comparisons.

I'm not aware of Rosetta. I'll see if I can find that site.

From: Brad Moore <brad.moore@shaw.ca>
Date: Thu, 24 Mar 2011 23:25:53 -0600
Subject: Re: ANN: Storage pool for Ada 2005 with bindings to Apache Runtime Pools library
Newsgroups: comp.lang.ada

[…]

Actually, thinking about it some more, it makes sense to me that 5 workers would be the best choice for 9 iterations and 4 processors.

At t=0, 5 workers should proceed at the same rate, (assuming that processor affinity is not set on the tasks). The 5 workers should migrate as needed between the 4 processors to ensure fair sharing of the processing resources.

Four of the workers will be given two iterations, while one will be given a single iteration.

The worker with a single iteration will finish first. At that time the other workers should have roughly one full iteration left.

At that point there are four workers with even work loads, and four processors.

The workers proceed until all the work is complete, and all processors were fully loaded for the entire processing.

KDF9 emulator in Ada 2005

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Sat, 16 Apr 2011 01:44:34 +0100
Subject: KDF9 emulator in Ada 2005
Newsgroups: comp.lang.ada

The first public release of ee9, my KDF9 emulator, is now available via:

http://www.findlayw.plus.com/KDF9/#Emulator

The zip file includes a Mac OS X binary, and full Ada 2005 source code.

Courtesy of David Holdsworth, a Linux/FreBSD binary is available at:

http://sw.ccs.bcs.org/KDF9/ee9.zip

(Many thanks to Simon Wright for building a 64-bit GNAT for MacOS X last year, which rescued me from an over-hasty adoption of Snow Leopard.)

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Sat, 16 Apr 2011 11:08:44 +0200
Subject: Re: KDF9 emulator in Ada 2005
Newsgroups: comp.lang.ada

[…]

To see a computer of roughly my own age having used highlighting, in color, making form follow function, and serving intuition ("it's that simple"…) is amazing.

I apologize if this is its least noteworthy feature.

It is spectacular.

It's got separate stacks for separate purposes!

Another of the few insights into the history of wonderful inventions that market forces have not improved. Thank you!

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Sat, 16 Apr 2011 13:29:00 +0100
Subject: Re: KDF9 emulator in Ada 2005
Newsgroups: comp.lang.ada

[…]

You're welcome! KDF9 does seem to inspire that sort of admiration in people, Even 50 years after it was designed. 8-)

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Thu, 21 Apr 2011 03:01:28 +0100
Subject: ee9 available in Snow Leopard, PPC Tiger and Intel Linux ports.
Newsgroups: comp.lang.ada

See

http://www.findlayw.plus.com/KDF9/emulation/emulator.html

Mike Hore, who built the PPC version for me, said:

"I must say this whole process went amazingly smoothly. (Having ported a few other projects in my time :-) I must have a closer look at Ada."

Ahven 1.9

From: Tero Koskinen's blog
Date: Tue, 19 Apr 2011
Subject: Ahven 1.9 released
URL: http://tero.stronglytyped.org/2011/04/ahven-19

I finally managed to release Ahven 1.9 […]

[http://sourceforge.net/projects/ahven/files/ahven/Ahven%201.9/ahven-1.9.zip/download —mp]

This is a small bug fix release only to allow Ahven compile with GNAT GPL 2010 also.

If everything goes as planned, the next 2.0 will include new features, like timeouts, improved documentation, and possibly an ability to skip some tests.

[see also "Ahven 1.8" in AUJ 31‑3 (Sep 2010), p.159 —mp]

Ada-related Products

AdaCore — GNAT Pro 6.4

From: AdaCore Press Center
Date: Tue, 01 Mar 2011
Subject: AdaCore Releases GNAT Pro 6.4
URL: http://www.adacore.com/2011/03/01/adacore-releases-gnat-pro-6-4/

Ada development environment brings new features, platforms, and tool support

NEW YORK, PARIS and NUREMBERG, Germany, March 1, 2011 – Embedded World Conference – AdaCore, a leading supplier of Ada development tools and support services, today announced the release of a new major version of its GNAT Pro development environment. GNAT Pro 6.4, the latest annual release of the company's flagship product, evidences AdaCore's continuing commitment to regularly scheduled technology updates with new features and improvements (many based on user suggestions), support for new platforms, and integration with new tools. As with all AdaCore products, GNAT Pro is Freely-Licensed Open Source Software (FLOSS).

"In the FLOSS world, you can't just sit on your hands, do nothing, and expect to succeed," said Robert Dewar, AdaCore President and CEO. "You have to constantly innovate. One of the rewarding things about AdaCore is that we have been able to keep up a high pace of exciting innovation. Our regular update release of GNAT Pro 6.4, which marks the 10th anniversary of our regular release schedule, is no exception. This is a major update of the product, including significantly improved performance and major new features.

When a new version of Ada is on the way, in this case Ada 2012, you don't have to wait for the formal release to try it out. GNAT Pro 6.4 already has all the important stuff from Ada 2012, in 2011!"

The new GNAT Pro 6.4 features include the following enhancements:

- Ada 2012 preview, including most of the currently finalized Ada Issues (AIs) and in particular conditional/case/parameterized/quantified expressions, aspect specifications (including pre/postconditions and type invariants), subtype predicates, and improved support for multiprocessors
- Improved code generator based on GCC 4.5
- New switch to generate cross reference information for C and C++
- More detailed exception messages
- New rules for gnatcheck (coding standard enforcement tool)
- New warnings
- Better debugger performance
- More flexible and more efficient project manager in gnatmake/gprbuild
- More aggregates recognized as static
- Support for GNATbench 2.5 Integrated Development Environment, which will work with a new version of Eclipse

GNAT Pro 6.4 is now available for a new embedded platform, Wind River's VxWorks MILS 2.1.x, and for updated versions of VxWorks 6 Cert (6.6.2) and SYSGO's ElinOS (5.1). On the native side, GNAT Pro 6.4 supports Red Hat Enterprise Linux 6 on x86 (32- and 64-bit).

New tools now supported by GNAT Pro, and available separately, include GNATemulator and GNATcoverage. Used together, these are especially useful for High-Integrity applications where safety certification is required, such as DO-178B for avionics. The tools' supported architectures include PowerPC and LEON.

GNATemulator, based on QEMU technology, offers an efficient and flexible emulator solution for Ada, C and C++ applications. It allows developers to compile code directly for their target architecture and run it on their host platform, through an approach that translates from the target object code to native instructions on the host.

The GNATcoverage tool, known earlier as XCov, performs coverage analysis on both object code (including branch coverage) and source code. Source code analysis includes both decision coverage and Modified Condition/Decision Coverage (MC/DC). The tool does not require instrumentation of the executable code. Instead, the tests can be run either on a version of GNATemulator instrumented for collecting coverage data, or directly on a board with a suitable debugger interface.

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial software solutions for Ada, a modern 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 worldwide customer base;

see http://www.adacore.com/home/company/customers/ for further information.

Ada and GNAT Pro continue to see 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 — CodePeer 2.0

From: AdaCore Press Center
Date: Tue, 12 Apr 2011
Subject: AdaCore Releases Major New Version of CodePeer Source Code Analysis Tool

URL: http://www.adacore.com/2011/04/12/codepeer2/

More efficient, generates fewer "false positive" messages

NEW YORK and PARIS, April 12, 2011 – AdaCore, a leading supplier of Ada development tools and support services, today announced the release of CodePeer 2.0, the advanced source code analysis tool that helps developers detect potential run-time and logic errors in Ada programs. CodePeer 2.0 also comes with a number of complementary static analysis tools common to the GNAT Pro technology – a coding standard verification tool (GNATcheck), a program metric generator (GNATmetric), a semantic analyzer, and a document generator that can be invoked through the GNAT Programming Studio (GPS) Integrated Development Environment (IDE).

CodePeer 2.0 introduces many enhancements to the technology, most driven by customer feedback, including:

- Support for subprogram calls via pointers.
- Much more efficient intermediate format (SCIL) generation, with faster processing and simpler (and fewer) SCIL files. In addition, CodePeer requires fewer partitions by default to perform an analysis.
- Support for parallel SCIL generation on multiple cpus/cores, via the gnatmake "-j" switch.
- New, "useless self assignment" warning when an assignment does not modify the destination variable.
- Fewer "false positives" (false alarms).
- Improved integration with the GPS IDE.

"CodePeer 2.0 brings maturity to our static analyzer technology and allows processing of large applications easily, taking particular advantage of multi-core computers," said Arnaud Charlet, CodePeer Project Manager at AdaCore. "CodePeer is not just a bug finding tool; its ability, in particular, to generate and display annotations in a human readable form is a unique capability."

Webinar

A webinar introducing the CodePeer 2.0 features will be presented by Tucker Taft (SofCheck) on May 5, 2011, at 11:00 am (EDT) / 5:00 pm (GMT). For more information, or to register, please visit http://www.adacore.com/home/gnatpro/webinars/.

About CodePeer

Serving as an efficient and accurate code reviewer, CodePeer identifies constructs that are likely to lead to run-time errors, such as buffer overflows, and it flags legal, but suspect, code typical of logic errors.

Going well beyond the capabilities of typical static analysis tools, CodePeer also produces a detailed analysis of each subprogram, including pre- and post-conditions. Such an analysis makes it easier to find potential bugs and vulnerabilities early: if the implicit specification deduced by CodePeer does not match the component's requirements, a reviewer is alerted immediately to a likely logic error. CodePeer can be used both during system development − to prevent errors from being introduced, or as part of a systematic code review process to dramatically increase the efficiency of human review − and retrospectively on existing code, to detect and remove latent bugs.

CodePeer was developed jointly by AdaCore and SofCheck.

Inspirel — YAMI4 1.3.0

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 8 Mar 2011 05:34:34 -0800 PST
Subject: YAMI4 1.3.0 released
Newsgroups: comp.lang.ada

I am pleased to announce that the 1.3.0 version of YAMI4 is available for download:

http://www.inspirel.com/yami4/

This new release is an important milestone and includes a rich Ada-oriented content:

The Ada-Ravenscar variant of the core interface was introduced for systems that expect compliance with the Ravenscar language profile.

The original API was retained as well - the two versions are very similar and differ only in those places where the original interface violated the restrictions on implicit use of dynamic memory.

Three central services were added to complement the peer-to-peer capabilities of the YAMI4 library. Contrary to some other messaging solutions, these services are not necessary components, but are provided as optional utilities that extend the YAMI4 suite with wider set of performance and deployment characteristics:

1. Name Server that helps to build easy to configure and manageable systems.
2. Message Broker that supports heavy-duty publish-subscribe messaging with very powerful and flexible message routing engine.
3. Cache that allows easy data sharing between nodes in a single distributed system.

All three central services were implemented in Ada on top of the Ada-Ravenscar interface. The reason for this was to demonstrate that the provided API is functionally complete and supports efficient implementation of non-trivial distributed systems. The binary versions of these services are provided for Windows and Linux as turn-key components for instant deployment.

Some minor tweaks have been applied to Makefiles and .gpr files to ensure proper compilation with recent GNAT versions and to include FreeBSD in the set of supported systems.

Those programmers who work with multi-language systems will also appreciate the complete .NET implementation in C# - the principal intent of this library is to integrate Windows-oriented GUI applications written for the .NET framework with other components in a single distributed system.

Of course, the YAMI4 book was revised to cover all these new features.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 8 Mar 2011 12:56:49 -0800 PST
Subject: Re: YAMI4 1.3.0 released
Newsgroups: comp.lang.ada

In addition to the original announcement, I have the pleasure to post the following:

The tutorial on YAMI4 (or rather the Ada perspective of the whole suite) will be given at the Ada Connection 2011 conference:

http://conferences.ncl.ac.uk/adaconnection2011/tutorials/T3.html

I would like to kindly invite all those Ada programmers who will attend the conference and who are interested in distributed systems.

Don't hesitate to contact me in advance if you have any related questions.

[see also "Inspirel — YAMI4 v. 1.2.1 and 1.2.2" in AUJ 32-1 (Mar 2011), p.15 —mp]

XGC Technology — LEON Ada 1.8

From: XGC website
Date: Tue, 10 May 2011 [fetched]
Subject: LEON Ada Version 1.7 -- now updated to 1.8
URL: http://www.xgc.com/news/news.html

We announce the first version of LEON Ada. Based on ERC32 Ada, this is a restricted Ada 95 compilation system for the new LEON microprocessor from ESTEC and Atmel. Two spacecraft products, the Atmel AT695E and AT695F are explicitly supported. See the technical summary.

[http://www.xgc-tek.com/manuals/pdf/leon-ada-ts.pdf —mp]

An evaluation copy of LEON Ada is available now.

Ada and GNU/Linux

Binding to ncurses available in Debian unstable

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sun, 17 Apr 2011 00:27:53 +0200
Subject: Ada in Debian: libncursesada -- Ada binding for the ncurses text user interface library
Newsgroups: comp.lang.ada

Thanks to the initiative and efforts of Nicolas Boulenguez, I am happy to announce that the Ada binding to ncurses is now part of Debian unstable, with the same level of quality as all other Ada packages.

The following packages are now but an "aptitude install" away:

libncursesada-doc - Ada binding to the ncurses text interface library: documentation

libncursesada1 - Ada binding to the ncurses text interface library: shared library

libncursesada1-dbg - Ada binding to the ncurses text interface library: debug symbols

libncursesada1-dev - Ada binding to the ncurses text interface library: development

These packages add to the already extensive complement of libraries present in Debian for the Ada developer.

SPARK GPL 2010 available in Debian unstable

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 19 Apr 2011 05:19:05 -0700 PDT
Subject: SPARK on Debian!
Newsgroups: comp.lang.ada

It is my pleasure once again to announce a new addition to Debian relevant to Ada programmers.

Eugeniy Meshcheryakov, a Debian Developer, has completed the initial work in packaging the SPARK GPL 2010 toolset for Debian with help from Florian Schanda of Altran Praxis. The following new package is therefore available in Debian unstable:

spark - SPARK programming language toolset

Please test and report any problems you find in the Debian bug tracking system.

Note that Eugeniy does not follow comp.lang.ada but he is reachable on the mailing list debian-ada@lists.debian.org, where highly detailed and technical discussions are welcome.

From: Rod Chapman <roderick.chapman@googlemail.com>
Date: Tue, 19 Apr 2011 11:04:19 -0700 PDT
Subject: Re: SPARK on Debian!
Newsgroups: comp.lang.ada

Nice work guys. I'm very pleased to see the GPL edition of SPARK reach the Debian community.

New Debian maintainer and Debian packages

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 10 May 2011 01:31:57 -0700 PDT
Subject: Nicolas Boulenguez is a new Debian Maintainer
Newsgroups: comp.lang.ada

Hello

After Xavier Grave[1], Stephen Leake[2], Reto Buerki[3], it is my pleasure to announce the continued growth of the Ada in Debian team. Nicolas Boulenguez has been accepted as an official Debian Maintainer and has already made his first upload. Nicolas has been my Padawan for more than a year now and has adopted the libtexttools package from me as well as added new ones. He currently maintains:

libgmpada -- Ada binding to the GNU MultiPrecision library*

libncursesada -- Ada binding to the ncurses text interface library*

libtexttools -- Ada and C++ library for writing console applications

oolite -- space sim game, inspired by Elite

Nicolas is also the upstream author of libgmpada. The packages marked * are new in Debian and would not be there at all if it were not for Nicolas.

In addition, Nicolas is a member of the team that maintains:

gnat-gps -- The GNAT Programming System - advanced IDE for C and Ada

Nicolas has also made contributions to the packaging of GNAT, the Ada compiler.

Now that he is a Debian Maintainer, he has the right to upload new versions of his packages to the Debian archive without supervision from a sponsor.

In time, I do hope he will grow into a full Debian Developer with voting rights, the right to upload any package and to sponsor packages for his own Padawans :)

[1] http://lists.debian.org/debian-ada/2010/03/msg00000.html

[2] http://lists.debian.org/debian-ada/2010/04/msg00004.html

[3] http://lists.debian.org/debian-ada/2010/05/msg00000.html

Ada Inside

Ada used for the development of the nEUROn Unmanned Aircraft

From: AdaCore Press Center
Date: Tue, 1 Mar 2011
Subject: EADS CASA Selects AdaCore Toolset for nEUROn Unmanned Aircraft
URL: http://www.adacore.com/2011/03/01/neuron-unmanned-aircraft/

GNAT Pro supports high-integrity systems on European unmanned air combat vehicle

NEW YORK, PARIS and NUREMBURG, Germany, March 1, 2011 – Embedded World Conference – AdaCore, provider of tools and expertise for mission-critical, safety-critical, and security-critical software development, today announced that EADS CASA is using the GNAT Pro High-Integrity Edition to implement the data exchange and air-to-ground data links systems for the nEUROn Unmanned Combat Air Vehicle (UCAV) demonstrator. The GNAT Pro High-Integrity Edition, which encompasses AdaCore's development environment and accompanying support services, helps develop Ada systems that need to achieve the highest levels of safety and/or security certification. EADS CASA selected AdaCore and GNAT Pro based on the Ada programming language's suitability for developing critical systems and AdaCore's previous experience with high-integrity projects, including the Eurofighter, MRTT, Airbus A400M, and Barracuda projects.

[…]

The delta wing nEUROn UCAV project began in 2006 and is a technology demonstrator for future European combat aircraft. It is one of the largest and most advanced unmanned air vehicles in the world, with a similar airframe size to that of some of the existing fighter aircrafts. Led by Dassault Aviation of France, the nEUROn project is a pan-European co-operation among EADS CASA (Spain), HAI (Greece), Saab (Sweden), RUAG (Switzerland), Alenia Aeronautica (Italy), and Thales (France). Test flights will take place in France, Sweden and Italy. The maiden flight is scheduled for mid-2012.

EADS CASA, which began software development planning at the end of 2007, is responsible for the nEUROn wing, ground control, and data link segments of the project. The company selected AdaCore and the GNAT Pro High Integrity Edition for DO-178B to develop the critical ground stations software and the data link management software, which will run on the Wind River VxWorks 653 Platform.

The finished project will create over 500,000 lines of code.

With the GNAT Pro High-Integrity Edition for DO-178B, the EADS CASA development team benefits from a complete toolsuite that simplifies compliance with the various certification levels of the DO-178B avionics standard on the same hardware. This includes code standard verification (GNATcheck), static stack size analysis (GNATstack), and a choice of several certifiable Ada run-time libraries.

Smooth and industry-proven integration with Wind River's Workbench platform means an environment oriented towards the needs of the safety-critical industry, whether the application is developed in Ada 83, Ada 95, Ada 2005, or a combination of Ada, C or C++.

[…]

Use of Ada in Singo's Blaze call management system

From: AdaCore Press Center
Date: Tue, 19 April 2011
Subject: AdaCore Helps Singo Solution Implement Scalable, Robust Call Management System
URL: http://www.adacore.com/2011/04/19/singo/

Improved quality, reliability, and performance in new Blaze product

NEW YORK and PARIS, April 19, 2011 – AdaCore, a leading supplier of Ada development tools and support services, today announced that Singo Solution, Inc., a global provider of specialized call-center solutions, selected the Ada programming language and GNAT Pro Development Environment to build Blaze – one of the largest, most innovative call management systems. Blaze is a comprehensive, scalable, web-enabled VoIP dialing platform that manages customer interactions via phone lines and a variety of other electronic methods of communication. The system is already being used in both traditional and non-traditional call center applications around the world, including healthcare, cable and telecommunications, utilities, insurance and many others.

When work started on the Blaze product, Singo realized that its own C-based call detection algorithm, and the many complicated contact management functions, were too time consuming and costly to maintain, and that enhancing them would not be practical, so the company decided to start from scratch. They knew that the programming language choice was critical, and decided early on that Ada was the best available language for producing code with the expressiveness, speed, and – most importantly – the reliability needed for a call center system of Blaze's magnitude.

"We know we made the right decision in choosing Ada, because it is simply more robust than any other language available," said Sieu Ngo, President and CEO, Singo Solution, Inc. "The code is efficient, but it is also extremely readable and maintainable. It helped us get our code done right the first time."

As a strongly typed language, Ada detects at compile time many errors that would only be found during testing and integration in a language such as C. This allows developers to spend their time enhancing the core software with new features and functions, versus trying to figure out why things aren't working. One of Ada's distinguishing features is its support for concurrency.

Concurrency is inherent in the Singo Blaze call center software logic: many different calls come in, are routed, and are handled all at the same time.

Singo found Ada's tasking a natural solution for the system's concurrency requirements, and was able to achieve key objectives of efficiency and reliability.

- Efficiency

Ada's tasking implementation automatically takes advantage of multi-core platforms.

As the number of cores increases, the operating system can automatically exploit the additional processing power by allocating tasks to the new cores, with a resultant speedup in system performance.

-Reliability

With Ada's tasking model, data corruption errors can be prevented by defining the shared data structures as protected objects. The Ada tasking implementation ensures that when a protected object is being modified, this operation is executed with mutual exclusion.

To implement the Blaze Call Management System, Singo selected AdaCore's GNAT Pro Ada Development Environment. The GNAT Pro product offers a modern, professional software build environment and tool-chain that scale up to handle very large systems. Most importantly, GNAT Pro is backed by expert support services provided by the product developers themselves. GNAT Pro is available on more native and embedded platforms than any other Ada environment, and is widely used to write applications with the most demanding reliability, safety and/or security requirements.

In developing the Blaze system, Singo used several AdaCore products that complement the GNAT Pro toolset. One is the Ada Web Server (AWS) technology, which supports development of web-based graphical user interfaces (GUIs). The new GUI allowed new users to be instantly productive where past systems almost always required some training prior to use. Another AdaCore product heavily used in Blaze is PolyOrb, which supports CORBA distributed communications. Much like the basic architecture where Ada tasking can take advantage of the number of cores, Blaze can handle an increasing number of phone lines, nodes within the CORBA implementation, without needing to be rebuilt. If the number of lines increase as a call center grows, new memory can be added to the hardware to support new nodes.

Thus Blaze is highly scalable, handling a range from just a few call lines to upwards of hundreds of thousands if required.

[…]

Use of Ada in Rockwell Collins' new avionics display system

From: AdaCore Press Center
Date: Mon, 2 May 2011
Subject: Rockwell Collins Selects GNAT Pro for Advanced Avionics Display System
URL: http://www.adacore.com/2011/05/02/rockwell-avionics-display/

GNAT Pro High-Integrity Edition for DO-178B used for EFIS/EICAS upgrade and modernization

SAN JOSE, Calif., NEW YORK and PARIS, May 2, 2011 – Embedded Systems Conference – AdaCore, a leading supplier of Ada development tools and support services, today announced that Rockwell Collins has adopted AdaCore's GNAT Pro High-Integrity Edition for DO-178B to implement the Electronic Flight Instrument System / Engine Indication and Crew Alert System (EFIS/EICAS) Interface Unit, model EIU-7001. These are key components of an advanced avionics display system that is being deployed on major jet aircraft. Rockwell Collins is using the GNAT Pro High-Integrity Edition for bare board PowerPC ELF as the cross-compilation environment for developing this new-generation software.

Rockwell Collins selected the GNAT Pro tool chain to port the existing Ada codebase from the aircraft's previous display system to a modern processor, as the most efficient and lowest risk approach to system modernization. Using the Ada language and GNAT Pro has allowed Rockwell Collins to easily integrate the proven legacy codebase while adding several new features from other Rockwell Collins products. By using this approach, the company has been able to quickly port the existing code and get it running on the new platform.

The GNAT Pro High-Integrity Edition for DO-178B development environment

includes several run-time libraries that are fully certifiable to DO-178B Level A requirements. For the EIU-7001display system project, Rockwell Collins has selected the Zero-Footprint (ZFP) run-time library, corresponding to a sequential Ada subset that eliminates both non-deterministic and complex language features. Use of this clearly-defined subset has significantly reduced the Rockwell Collins certification effort.

A core requirement for the display system is the ability to run on processors that can support the many new features required for modern aircraft. The largest part of the EIU-7001 display system development effort has been the migration of the Electronic Flight Instrument System / Engine Indication and Crew Alert System (EFIS/EICAS) Interface Unit (EIU) to the new target processor. The original EIU was designed in the late 1980's using the Rockwell Collins AAMP platform. For the EIU-7001, significantly more processing capability was required to support the modern flight deck. The new EIU-7001 meets the current display system capabilities with spare capacity as additional functionality is required in the future.

[…]

Ada used for the development of the Argos satellite's payload

From: AdaCore Press Center
Date: Mon, 2 May 2011
Subject: Thales Selects AdaCore Toolset for Argos Satellite Project
URL: http://www.adacore.com/2011/05/02/thales-argos-satellite/

GNAT Pro to be used on high-assurance software for global location and data collection system

SAN JOSE, Calif., NEW YORK and PARIS, May 2, 2011 – Embedded Systems Conference AdaCore, provider of tools and expertise for the development of mission-critical, safety-critical, and security-critical software, today announced that Thales Airborne Systems has selected the GNAT Pro High Integrity Edition to develop onboard instrument software for the next generation of the Argos satellite project. Argos is a unique, satellite-based worldwide location and data collection system dedicated to studying and protecting the environment. In addition to these core tasks, the Argos satellite family is also known for its safety and security-related applications, including boat localization, territorial security, and law enforcement. The GNAT Pro High-Integrity Edition, which encompasses AdaCore's development environment and accompanying support services, is focused on Ada systems that need to achieve the highest levels of safety and/or security certification.

In its new mission, Argos will face several challenges. In particular, Argos-4 will simultaneously handle three times as many transmitters as Argos-3 and provide these transmitters with increased operational flexibility. Thales chose AdaCore because of AdaCore's ability to provide a complete solution that includes an efficient development environment (GNAT Pro), a LEON 2 emulator (GNATemulator), and a code coverage tool (GNATcoverage) that does not require code instrumentation.

This combination streamlines the development, testing, and validation of the software as all these tasks can be performed on the host development platform.

Operational since 1978, Argos enables scientists across the globe to gather information on any object equipped with an appropriate transmitter. Messages from these transmitters are recorded by a series of satellites carrying Argos instruments and then relayed to dedicated processing centers.

By measuring temperature, pressure, humidity and sea levels, Argos transmitters provide invaluable information on the planet and Earth's atmosphere. Argos is used for a variety of applications, including volcano monitoring, ship and expedition tracking, fishing management, tracking animal migration, and geophysical data collection. Begun jointly by France and the United States (National Oceanic and Atmospheric Administration – NOAA), Argos is operated globally by Collecte Localisation Satellite (CLS), a subsidiary of the French Centre National d'Etudes Spatiales (CNES) and Ifremer, the French institute of marine research and exploration.

Thales will be using GNAT Pro for LEON ELF and the LEON 2 simulation platform. The LEON2 processor was commissioned by the European Space Agency (ESA), and is designed specifically for use in satellite systems.

The 18 month project is expected to be completed in mid-2011, and is estimated to require approximately 25,000 lines of code. The Argos-4 contract is covered by the Thales corporate-wide software licence with AdaCore.

[…]

NIST report on SPARK

From: AdaCore Press Center
Date: Mon, 2 May 2011
Subject: NIST Report Shows SPARK Most Suitable Language for Secure Programming
URL: http://www.adacore.com/2011/05/02/spark-nist/

SPARK language shown to have fewest vulnerabilities

SAN JOSE, Calif., NEW YORK and PARIS, May 2, 2011 – Embedded Systems Conference – AdaCore, a leading supplier of Ada development tools and support services, today announced that the SPARK language's immunity to many vulnerabilities found in other languages has been corroborated by a recent report published by the National Institute of Standards and Technology (NIST). This report – Source Code Security Analysis Tool Functional Specification Version 1.1 (NIST Special Publication 500-268 v1.1) – examines software assurance tools as a fundamental resource to improve quality in today's software applications. It looks at the behavior of one class of software assurance tool: the source code security analyzer. Because many software security weaknesses are introduced at the implementation phase, using a source code security analyzer should help reduce the number of security vulnerabilities in software.

The report defines a minimum capability to help software professionals understand how a tool can help meet their software security assurance needs.

The example languages studied are C, C++, Java and SPARK. SPARK, originally designed by Praxis, is a subset of Ada augmented with annotations ("contracts") that assist in automated proof of program properties, such as freedom from exceptions. The NIST report identifies the languages' vulnerabilities.

Vulnerability	Applicable to SPARK
Range Errors	
Stack Overflow	No
Heap Overflow	No
Format String Vulnerability	No
Improper Null Termination	No
API Abuse	
Heap Inspection	No
Often Misused: String Management	No
Time and State	
Unchecked Error Condition	No
Code Quality	
Memory Leak	No
Double Free	No
Use After Free	No
Uninitialized Variable	No
Unintentional Pointer Scaling	No
Null Dereference	No

SPARK was designed to aid in safe and secure programming. By preventing vulnerabilities it reduces the cost of

developing safe and secure software applications, by reducing the time spent in finding errors and testing to meet top safety and security standards. Such reductions are hard to quantify, but this report from NIST helps identify the types of security flaws that can be guaranteed to be prevented when SPARK is used.

The full report, Source Code Security Analysis Tool Functional Specification Version 1.1 (NIST Special Publication 500-268 v1.1), is available on the web via the following link:

http://samate.nist.gov/docs/source_code_security_analysis_spec_SP500-268_v1.1.pdf

A detailed study on the SPARK language and vulnerabilities will be published shortly in a separate report, "Software Vulnerabilities Precluded by SPARK," by Dr. Joyce L Tokar and co-authors. This paper will be presented at the High Confidence Software and Systems (HCSS) 11th Annual Conference.

See: http://hcss-cps.org/hcssc.html.

SPARK Pro, an open source tool set for SPARK, is available from AdaCore.

See: http://www.adacore.com/home/products/sparkpro for more information.

[…]

Indirect Information on Ada Usage

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

Job offer [Belgium]: Ada 95 Software Engineer

[…] I am looking for an Ada 95 software engineer who will get involved in the following:

- Detailed Software Design.
- Implementation (design, code & test).
- Participate in test automation preparation and implementation.
- Writing of documentation mainly of technical nature.
- Participation in on-call service, third level (with remote equipment).

Skills required:

- Hardware/Platforms: HP-UX, Linux.
- Communications protocols: knowledge of the communication protocols such as TCP/IP.
- Good knowledge of relational databases (e.g. Oracle).
- Mainly Ada. C++ knowledge is an advantage, Ada 95 knowledge is an advantage.
- Unix Scripting, Emacs, test tools, ClearCase.
- Object-Oriented Analysis and Design (HOOD, Booch, UML, …).
- Experience with tools such as Rational Rose.
- MOTIF, Windows, GTK.

You will be able to absorb large amounts of complex information. In spite of the large degree of abstraction, this is a necessary requirement to be able to maintain and to implement new requirements in a code base of roughly 1.5 M LoC of Ada. Strong algorithmic knowledge and ability to abstract and factorize is essential. Being prepared to invest in learning about ATFM domain.

Due to the long learning curve and investment needed to get and bring somebody to proficient level, it is expected the successful candidate will be willing to stay quite a long time, i.e. years, on the project. Being prepared to participate in a third level on-call roster to provide application support.

[…]

Job offer [Germany]: Hardware/Software Integration Engineer

Skills:

- Software and Hardware Engineering
- Integration
- Good knowledge of Lauterbach Practice and Trace command files
- General Scripting language experience, eg Python, Perl, ACL

Nice to have:

- MC68020 knowledge
- Ability to understand Ada and C programming languages
- General Jet Engine operation
- General electronics and engine actuator knowledge
- Dimensions (Config control tool)
- DO-178B

Job Description:

To support the continued development of the EJ200 DECMU engine controller by providing testing and integration expertise both at an interactive and script based level.

Job offer [United Kingdom]: Ada Embedded Software Engineer

[…] This role requires the successful candidate to work on real-time and mission-critical systems […]

Working within small development teams, you will be involved in the full life cycle of software development from initial requirements capture to post-development support and may be involved in any of the following:

- Team/Project Leading
- Software analysis and design using formal OO methodology (such as UML).
- The development of robust ADA [sic —mp] embedded software.
- Code review.
- Unit and acceptance testing.

Job offer [United Kingdom]: Principal Ada Software Test Engineer

[…]

Role

To undertake a full range of software engineering activities in line with relevant processes, quality and other requirements, supplying specialist advice and support across projects, other business units and customer as appropriate.

Responsibilities include

- Take initiative in evaluating technical issues/solutions in order to develop proposals and technical demonstrators as required
- Perform a full range of technical analyses and investigations, including selection of preferred options to resolve problems and challenges within the project.
- Define test strategy, specify tests and evaluate results to meet the needs of the project.
- Support the identification of problems/issues/concerns arising within the team at any early stage.
- Supplying technical support and expertise to others as required.
- Provide reports on progress and achievement to the team leader as required.
- Support functional goals by identifying, suggesting and trialling improvements in SET capability.

In particular consider improvement towards the goals of a more Agile development process and enhanced test and diagnostic procedures.

[…]

Skills and Knowledge to include

- Degree qualified or equivalent (computing, software, or related)
- Capable of design, development and proving of systems for defence or equivalent complex system applications.
- Domain knowledge of design, development and proving of test and simulation facilities for the proving of software-based defence applications.
- Understanding of associated interfaces with aircraft and ground equipment systems.

[…]

- Sound knowledge of software processes, engineering and quality standards.

[…]

Experience of the following: (The first 2 are ESSENTIAL)

- Ada
- UML Tool or MASCOT
- Dimensions Configuration Control
- C
- Doors requirement management system*
- Matlab(Simulink)

Job offer [United Kingdom]: Software Engineer

[…]

Role Description

2 x Software Engineers are needed to work on the testing of Embedded Real Time avionics software written in Ada using the Rational Apex development environment.

Experience Required

- Experience of the Ada Programming Language
- Experience of Rational TestMate test tool
- Experience of Unix/Solaris operating system
- Experience of Avionics Software

[…]

Job offer [United Kingdom]: Software Engineer

[…] looking to immediately recruit a talented and highly motivated Software Engineer to join a small team developing state-of-the-art on-target verification tools.

These tools, which include code coverage and measurement-based execution time analysis, are used in real-time embedded systems development in the aerospace and automotive electronics industries.

As a Software Engineer, your main responsibilities will include specification, design, development, and testing of […] on-target verification tools.

You will have the opportunity to work on both the graphical user interface programmed in Java and Eclipse and the command line tools written in Ada. You can also expect to be involved in a variety of other activities, including EU research projects. […]

The successful candidate will be highly motivated, hard working, innovative, and self-reliant, with strong interpersonal and communication skills.

They will have previous software engineering experience, including strong programming skills in Java and Eclipse, (experience with C, C++, Ada, HQL/SQL and Unix is also desirable) and excellent degree level qualifications in Computer Science or related discipline (a 2.1 or higher).

Knowledge of real-time systems theory and concepts and embedded programming experience is also desirable.

Previous experience working for a technology start-up is seen as an advantage.

[…]

Ada in Context

Order of evaluation of subprogram parameters and aggregate components

From: Syntax Issues <syntax.issues@gmail.com>
Date: Sat, 19 Mar 2011 10:10:04 -0700 PDT
Subject: Order of execution of subprogram parameters
Newsgroups: comp.lang.ada

Across Ada compilers what is the order of execution for subprogram parameters? Is it left-to-right, right-to-left, and does it matter if I use labels out of order?

Example:

```
type Record_Joint is record
        Name   : String(1..64);
        Flags  : Integer_1_Unsigned;
        Parent : Integer_4_Signed;
        Start  : Integer_4_Signed;
    end record;
package Container_Record_Joint
    is new Ada.Containers.
        Indefinite_Vectors
            (Integer_4_Unsigned,
             Record_Joint);
    …
Animation    : Record_Md5_Animation;
…
Animation.Joints.Append(
    (Name   =>
      Remove_Quotes(
         Next_String(File, Current_Line)),
     Parent =>
      Next_Integer_4_Signed(
               File, Current_Line),
     Flags  =>
      Next_Integer_1_Unsigned(
               File, Current_Line),
     Start  =>
      Next_Integer_4_Signed(
               File, Current_Line));
…
function Next_X
    (File : in out File_Type;
     Line : in out String)
    return X;
```

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 19 Mar 2011 11:58:22 -0700
Subject: Re: Order of execution of subprogram parameters
Newsgroups: comp.lang.ada

[…]

> Across Ada compilers what is the order of execution for subprogram parameters? Is it left-to-right, right-to-left, and does it matter if I use labels out of order?

ARM 6.4 says,

"For the execution of a subprogram call, the name or prefix of the call is evaluated, and each parameter_association is evaluated (see 6.4.1). If a default_expression is used, an implicit parameter_association is assumed for this rule. These evaluations are done in an arbitrary order."

> Animation.Joints.Append(

```
   (Name   =>
   Remove_Quotes(Next_String(File,
   Current_Line)),
   Parent =>
   Next_Integer_4_Signed(File,
   Current_Line),
    Flags  =>
   Next_Integer_1_Unsigned(File,
      Current_Line),
   Start  => Next_Integer_4_Signed(File,
      Current_Line));
```

However, what you have here is not evaluation of subprogram parameters but evaluation of aggregate components. The answer, from ARM 4.3, is the same:

"For the evaluation of an aggregate, an anonymous object is created and values for the components or ancestor part are obtained (as described in the subsequent subclause for each kind of the aggregate) and assigned into the corresponding components or ancestor part of the anonymous object. Obtaining the values and the assignments occur in an arbitrary order."

On the overflow with Integer and Float types

From: Alex Mentis <asmentis@gmail.com>
Date: Thu, 31 Mar 2011 20:25:13 +0000 UTC
Subject: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

The following does not cause a constraint error in my version of GNAT on my system:

```
Integer_Result := (Integer'Last +
                   Integer'Last) / 2;
```

If I understand correctly, this is because the Integer operators are defined for operands of type Integer'Base, which is an unconstrained subtype and allows the operands to be stored in extended-length

registers so that intermediate values in calculations do not overflow.

My questions are:

1) Do I understand correctly what's going on?

2) Does the language make any guarantees about preventing spurious overflow, or am I just getting lucky with my compiler/architecture? If guarantees are made by the language, what are they?

From: Adam Beneschan <adam@irvine.com>
Date: Thu, 31 Mar 2011 14:10:55 -0700 PDT
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

> […] If I understand correctly, this is because the Integer operators are defined for operands of type Integer'Base, which is an unconstrained subtype and allows the operands to be stored in extended-length registers so that intermediate values in calculations do not overflow.

No, it's because all the operands are known at compile time and the compiler can just figure out what the answer is. There is no question about "where operands are stored" or about how registers are used.

The code for this statement should not perform any addition or division (or shift) operations. See 4.9.

From: Simon Wright <simon@pushface.org>
Date: Thu, 31 Mar 2011 22:18:29 +0100
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

[…]

That's a compile-time calculation, and any Ada compiler should work it out using infinite-precision arithmetic.

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Very_Large is
   Integer_Result : Integer;
begin
   Integer_Result := 10**128 / 10**127;
   Put_Line (
        Integer'Image (Integer_Result));
end Very_Large;

$ gnatmake very_large.adb
gcc -c very_large.adb
gnatbind -x very_large.ali
gnatlink very_large.ali
$ ./very_large
 10
```

As against

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Very_Large is
   Integer_Result : Integer;
begin
   Integer_Result := Integer'Last;
   Integer_Result :=   Integer_Result +
                       Integer'Last;
   Integer_Result := Integer_Result / 2;
   Put_Line (
        Integer'Image (Integer_Result));
end Very_Large;

$ gnatmake very_large.adb
gcc -c very_large.adb
very_large.adb:6:37: warning: value
not in range of type "Standard.Integer"
very_large.adb:6:37: warning:
"Constraint_Error" will be raised at run
  time
gnatbind -x very_large.ali
gnatlink very_large.ali
$ ./very_large
raised CONSTRAINT_ERROR :
very_large.adb:6 overflow check failed
```

Note that the compiler knew that was going to happen. If the overflow wasn't visible at compile time, you'd have to tell GNAT to perform run-time integer overflow checks using -gnato. Other compiler writers may have different views about whether run-time integer overflow checks should be off by default :-)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 31 Mar 2011 16:24:54 -0500
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

[…]

> 2) Does the language make any guarantees about preventing spurious overflow, or am I just getting lucky with my compiler/architecture? If guarantees are made by the language, what are they?

The language says effectively that you either will get the right answer or Constraint_Error. But it makes no guarantees about which you will get for values outside of the result subtype. So that is compiler-dependent.

The intent is to be able to use the hardware effectively. To take an example, older Intel X86 processors did all of their floating point calculations in 80-bit registers. The only certain way to use fewer bits was to store the register into memory and then reload it (which forced the needed rounding). Needless to say, this doesn't help performance!

In something like:

```
F := (A * B) / (C * D);
```

you would have two extra store/load pairs. That's awful, thus the rule allowing extra precision.

For float types, Ada actually has an attribute to explicitly discard extra precision (S'Machine). For integer types, you'd have to explicitly store the subexpression into an object and do a validity test on it. (It's not clear to me that a type conversion alone would guarantee a check for a type like Integer where Integer has the same range as Integer'Base. The validity rules always allow delaying a constraint check, so only 'Valid is certain to smoke out overflowing values.)

But both of these operations are expensive, and should only be used when absolute portability is needed.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 31 Mar 2011 23:09:49 +0200
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

[…]

I'm not sure what you mean by "spurious overflow" (as opposed to "overflow") but:

- static constants must be computed without any overflow checks at compile time (ARM 4.9(33)); this means that intermediate values can be arbitrarily large or small (ARM 4.9(35/2)) but the final result must be in the range specified for the constant. If not, the compiler reports an error.
- during execution, there are two kinds of overflow checks.

  Intermediate results must lie within the "base range of the type" which, for all intents and purposes, is the full range of [[Long_]Long_]Integer (ARM 4.5.4(20)). So, if an intermediate value exceeds e.g. Integer'Last you get a Constraint_Error.
- At the end of a computation, the result is either assigned to a variable, a constant, or a subprogram parameter. This assignment involves a conversion to the target subtype, the range of which may be smaller than the base range of the type, and this conversion includes an overflow check (ARM 4.6(51/2)) which must raise Constraint_Error if it fails (ARM 4.6(57)).

For example:

```
type T is range 1 .. 10;
A : T := 95 - 90; -- OK
```

See also

http://en.wikibooks.org/wiki/Ada_Programming/Type_System#Elaborated_Discussion_of_Types_for_Signed_Integer_Types

From: Alex Mentis <asmentis@gmail.com>
Date: Thu, 31 Mar 2011 21:26:10 +0000 UTC
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

> […] I suspect you compiled without the secret -gnato option

No, I compiled with that option enabled. It still ran happily and produced the correct output.

[…] By "spurious overflow" I mean overflow from intermediate results of a calculation in which the correct final result is actually still within the type constraints.

> - during execution, there are two kinds of overflow checks.

> Intermediate results must lie within the "base range of the type" which, for all intents and purposes, is the full range of [[Long_]Long_]Integer (ARM 4.5.4(20)). So, if an intermediate value exceeds e.g. Integer'Last you get a Constraint_Error.

Well, that's my question. In the calculation above, I clearly have an intermediate value that exceeds Integer'Last. And I tried something similar with Long_Long_Integer and still couldn't get an overflow error! So what is the actual limit on the base range of the type? Is it language defined, compiler defined, hardware defined, none of the above?

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 31 Mar 2011 23:36:36 +0200
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

[…]

It is both language- and implementation-defined. I gave you the references to what the language says. The implementation-defined part is the "base range of the type"; a sane implementation would choose a range that matches the hardware.

The reason why you did not get an overflow at runtime is because your computation did not take place at run time. […]

From: Alex Mentis <foo@invalid.invalid>
Date: Thu, 31 Mar 2011 21:51:29 +0000 UTC
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

> […] In something like:

F := (A * B) / (C * D);

you would have two extra store/load pairs. That's awful, thus the rule allowing extra precision.

Using variables gave me the behavior I was expecting. I didn't know Ada did infinite precision arithmetic on static expressions. […]

From: Adam Beneschan <adam@irvine.com>
Date: Thu, 31 Mar 2011 15:18:18 -0700 PDT
Subject: Re: Unconstrained base subtype questions
Newsgroups: comp.lang.ada

[…]

However, the language does specify that whatever base range the implementation chooses, Integer'Last will be the top of the range.

It's not legally possible for the "base range" of Integer to include values that are larger than Integer'Last. (Note that that applies only to Standard.Integer; it "should" apply to other predefined signed integer types in Standard, but it doesn't apply to user-defined integer types.)

The language does say, though (3.5.4(24)), that implementations don't need to raise Constraint_Error for arithmetic operations on signed integers as long as they produce correct results, even if intermediate results are outside the base range---that's what Randy was referring to, I think.

On aliased parameters in Ada 2012

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Mon, 28 Mar 2011 13:47:22 +0200
Subject: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

Hello,

I was looking at what the Ada 2012 reference looks like so far, and especially at all the “Extensions to Ada 2005” sections. In “Subprogram Declarations” there is such an extension.

Quote from Ada 2012:

“Parameters can now be explicitly aliased, allowing parts of function results to designate parameters and forcing by-reference parameter passing.”

I'm not sure I've understood. What was wrong with access type parameters?

I don't see a reason why aliased parameters may be required (and don't feel this can be clean). Well, why not in/out parameter for functions, as things like test-and-set are common idioms, but I really can't figure why this one in particular was required.
I may look at the related AI later, but it's long (so I will deffer it).

A quick overview of good reasons from someone who know ?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Mar 2011 22:04:53 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

> Interesting. How to force by-reference something allocated in a register?

The thing passed has to be aliased or tagged (of course), so the compiler just has to avoid allocating them in a register. Standard stuff.

> Yes one could copy the actual value and then pass a reference to the copy, but that is not what I would call "by-reference."

Neither do we.

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Mon, 28 Mar 2011 04:56:51 -0700 PDT
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

Aliased parameters are meant to cure a deficiency in e.g. containers.

With current Ada, it's awkward to replace an element of a container.

Now if you make the cursor an aliased parameter, the function result may dereference safely the cursor, since the cursor and the result are tightly coupled, i.e. the cursor cannot be changed as long as the function result exists:

```
function Get (Pos: aliased Cursor)
   return access Element;
```

This is the idea. The syntax in this example might not be correct.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Tue, 29 Mar 2011 20:22:51 +0200
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

> Now if you make the cursor an aliased parameter, the function result may dereference safely the cursor, since the cursor and the result are tightly coupled,

But this is already the case in Ada 2005. The lifetime of passed-in objects extends beyond the immediate need for evaluating the expression:

Leaving an execution happens immediately after its completion, except in the case of a _master_: the execution of a body other than a package_body; the execution of a statement; or the evaluation of an expression, function_call, or range that is not part of an enclosing expression, function_call, range, or simple_statement other than a simple_return_statement.

This is from 7.6.1(3/2).

I've written experimental code which depends on this, providing a syntactically convenient and efficient form of variadic subprograms.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Mar 2011 22:16:45 -0500
Subject: Re: Ada 2012 : aliased parameters?

Newsgroups: comp.lang.ada

> […] I'm not sure I've understood. What was wrong with access type parameters?

Two things: (1) need to explicitly write 'Access at every call site, when all you are doing is passing a by-reference parameter; (2) anonymous access requires a run-time accessibility level; that means that passing the wrong thing (a local object, for instance) might cause Program_Error to be raised later. This is an unnecessary hazard for the sorts of uses envisioned for aliased parameters. Aliased parameters move the check to the call site, where it almost always will succeed (there is one case involving function calls inside of allocators where it might fail, and if it does it almost always will fail at compile-time).

> I don't see a reason why aliased parameters may be required […]

The motivating case is to make the containers better. Ada 2012 adds the following to all of the containers:

```
function Reference (
    Container : aliased in out Vector;
    Position : in Cursor)
 return Reference_Type;
```

where Reference_Type is defined as:

```
type Reference_Type (
  Element : not null access
       Element_Type) is private
 with
   Implicit_Dereference => Element;
```

The aspect "Implicit_Dereference" lets you omit "Element.all" from uses of this type, so the call:

```
   My_Vector.Reference
            (My_Cursor).Comp := 10;
```

is legal (presuming the element type has a component "Comp"). And another "feature" allows a form of user-defined indexing, so you actually can write:

```
  My_Vector (My_Cursor).Comp := 10;
```

which is a big improvement over using Update_Element (which requires writing a procedure to do an in-place element update).

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 29 Mar 2011 00:34:49 -0700 PDT
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

Out of curiosity - is it possible to leak the reference this way? I mean - is it possible for the caller to make a copy of returned reference and store it arbitrarily long?

Note that the "copy" might not be obvious, as in:

```
declare
  My_Element :
     Vector_Type.Reference_Type
     renames
     My_Vector.Reference (My_Cursor);
begin
  My_Element.Comp := 10;
  My_Element.Other_Comp := 3.14;
end;
```

The C++ equivalent of this is both a fantastic performance feature and a deadly security hole. How is this solved in Ada?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 29 Mar 2011 19:09:47 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

>Out of curiosity - is it possible to leak the reference this way? […]

No, because the attempt to make the copy will fail the accessibility check.

Specifically, the access discriminant has the lifetime of the containing object. So if the object is short-lived (as most return objects are), the access discriminant cannot be assigned into anything that lives longer. OTOH, if the object is long-lived, there is no problem, because as long as the object lives, attempting to add or remove elements from the container is not allowed and must raise Program_Error.

There is are a couple of small holes that occur by using Unchecked_Deallocation, but no one is going to do that by accident, and if there is any sort of management (or sense) on a project, the end-around will be easily detected.

>Note that the "copy" might not be obvious, as in:

>

```
>declare
>   My_Element :
    Vector_Type.Reference_Type renames
>     My_Vector.Reference (My_Cursor);
>begin
>   My_Element.Comp := 10;
>   My_Element.Other_Comp := 3.14;
>end;
```

This isn't a leak, because the Reference object has to continue to exist until the renames goes away (and thus the reference). Instead, My_Vector is locked against "tampering" so long as that object exists. So any attempt to delete this element in this block body will raise Program_Error.

[…]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 29 Mar 2011 19:12:43 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

This works in your example because you avoid the type system completely and thus the accessibility checks. It also would work if you don't care that it can raise Program_Error if someone passes in a local variable. That runtime source of failure is not acceptable, and the use of aliased parameters makes it a compile-time check.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 30 Mar 2011 14:44:33 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

> There is are a couple of small holes that occur by using Unchecked_Deallocation, but no one is going to do that by accident […]

I should mention that there is a relatively easy way to defeat these accessibility checks: simply use .all'Unchecked_Access. More generally, any unchecked programming can of course default the protection. Nothing new about that - that's true of any Ada construct. But of course it is obvious that unchecked programming is being used, so again that is a management problem.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sat, 23 Apr 2011 20:47:16 +0200
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

```
> type Reference_Type (Element : not
  null access Element_Type) is
> private
>       with
>          Implicit_Dereference =>
  Element;
```

Is it necessary that Element is a discriminant? If the aliased business works with fields, you could write something like this:

```
type String_Reference is record
  Data : access String;
end record;

function "+" (S: aliased String)
   return String_Reference is
begin
  return String_Reference'(
     Data => S'Access);
end "+";
```

String_Reference could be part of an array, so we would get ragged arrays as a side effect.

In any case, it seems to me that the definition of "master" in 7.6.1 needs updating.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 25 Apr 2011 02:19:35 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

> Is it necessary that Element is a discriminant?

Yes, because access discriminants have special accessibility rules which happen to have the right effect.

> If the aliased business works with fields, you could write something like this:

>

> type String_Reference is record

> Data : access String;

> end record;

You can write this, but you lose all accessibility checking if you do. The accessibility of a normal component is that of the type, which is typically library level. OTOH, a discriminant in a returned object has the accessibility of the point of call; combined with the rules for aliased parameters, such a discriminant will always succeed (no runtime checks or overhead needed) and will always be safe (can't copy it into anything with a longer lifetime, which is essentially anything).

I'm no fan of accessibility checks, but in this case at least they don't get in the way beyond preventing operations that we don't want to allow in the first place.

> In any case, it seems to me that the definition of "master" in 7.6.1 needs updating.

It did, but only for bugs. The access discriminant semantics is from Ada 95, although it was never defined properly (probably still isn't, although not for the lack to trying). We've just found a good use for the strange semantics.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Thu, 28 Apr 2011 21:47:37 +0200
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

>> Is it necessary that Element is a discriminant?

> Yes, because access discriminants have special accessibility rules which happen to have the right effect.

This is unfortunate because it means that this cannot be used to make variadic argument list trick safer and less of a hack.

> It did, but only for bugs. The access discriminant semantics is from Ada 95, although it was never defined properly […]

I don't think the difference is observable in Ada 95 because you couldn't return new objects of limited type.

By the way, how tight are the access level checks? Is it relatively safe to assume that if an Ada 2005 compiler compiles a program which makes heavy use of anonymous access types and runs it without exceptions, then there are no dangling pointers? (Ignoring unchecked deallocation, of course.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 28 Apr 2011 18:54:52 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

One could argue that variadic arguments are themselves a hack. :-)

This feature is intended for one particular use (and any other uses are a happy accident): providing safe user-defined dereferencing. What is needed for it to be safe is to prevent any copying of the access value while still allowing it to be dereferenced (including assigning into it).

We originally had some syntax to define the accessibility of the returned access type, but it was eventually pointed out that access discriminants already had the appropriate accessibility. (Anonymous access return types also have this same accessibility.) Thus we changed the mechanism to use the discriminants rather than inventing a new feature.

The advantage of the aliased parameters is that they eliminate the runtime checks by forcing the checks to the call site (where they can be statically made 99% of the time).

[…]

> By the way, how tight are the access level checks? Is it relatively safe to assume that if an Ada 2005 compiler compiles a program which makes heavy use of anonymous access types and runs it without exceptions, then there are no dangling pointers? (Ignoring unchecked deallocation, of course.)

The intent is that it is impossible to create a dangling pointer if no unchecked programming is used. (Unchecked_Deallocation, 'Unchecked_Access, Unchecked_Conversion, Address_to_Access_Conversions, abuse of Unchecked_Unions, etc.) That goes for all access types (not just anonymous ones). The problem, of course, is that it is impractical to do much without using one of those things. (I've only succeeded in using 'Access once in one of my programs; in all other cases I had to use 'Unchecked_Access.)

We're constantly fixing holes in the model, and it is easy to use the unchecked things, so I wouldn't consider it impossible to get a dangling pointer. (Personally, I prefer to hide pointers as much as possible, as in the container cursors, so that dangling pointer detection becomes much more possible, and their creation becomes less likely.)

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sat, 30 Apr 2011 20:32:04 +0200
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

> One could argue that variadic arguments are themselves a hack. :-)

It would make it possible to call this little gem, PostgreSQL's main client function for executing SQL statements,

```
PGresult *PQexecParams(
                PGconn *conn,
                const char *command,
                int nParams,
                const Oid *paramTypes,
                const char * const
                              *paramValues,
                const int *paramLengths,
                const int *paramFormats,
                int resultFormat);
```

without any allocations and in a type-safe manner (for a predefined set of types). For such untyped external interfaces, variadic subprograms are often handy.

> The advantage of the aliased parameters is that they eliminate the runtime checks by forcing the checks to the call site (where they can be statically made 99% of the time).

I'm wondering if it is necessary that the returned limited record is controlled, so that a reference counter can be incremented and later decremented to ensure that the access discriminant does not become dangling. That would make the whole thing a bit clumsy to use, and come with quite a bit of run-time overhead.

> The intent is that it is impossible to create a dangling pointer if no unchecked programming is used.[…]

Anonymous access types seem to help quite a bit. I use 'Access for access discriminants, creating proxies, to fake the in-out parameter mode for functions, and on locally defined callback functions.

> (Personally, I prefer to hide pointers as much as possible, as in the container cursors, so that dangling pointer detection becomes much more possible, and their creation becomes less likely.)

And implicit deference could make them even safer to use.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 30 Apr 2011 18:46:57 -0500
Subject: Re: Ada 2012 : aliased parameters?
Newsgroups: comp.lang.ada

[…]

> I'm wondering if it is necessary that the returned limited record is controlled […]

It's not required, but that is the way it will be used in the containers (so that the tampering check can apply only so long as the access exists). One hopes that compilers will work to minimize the overhead in this case (the non-list finalization implementation that GNAT is supposedly getting will be ideal for such cases).

[…]

> Anonymous access types seem to help quite a bit. […] And implicit deference could make them even safer to use.

Exactly. And more convenient, too.

I would like to see containers use to be as easy as using access types; if that is true, then there is little reason to use the less safe access types to create lists and trees (and maps and sets). There always will be cases not covered by containers or where performance needs are ultra-critical -- but those should be the unusual cases. Ada 2012 definitely moves us closer to that goal.

Breaking circularity in data structures

From: Gene Ressler <gene.ressler@gmail.com>
Date: Sun, 27 Mar 2011 21:11:10 -0700 PDT
Subject: Breaking a circularity
Newsgroups: comp.lang.ada

Need some expert help here with a circularity in the data structures for a discrete event simulation.

We need event queues and an arbitrary number of event types. The handlers for events must have access to a simulation state record, so we used a generic:

```
generic
   type State_Type is private;
package Event_Queues is

   type Event_Type is abstract
      tagged limited private;

   procedure Handle(
      State : in out State_Type;
      Event : access Event_Type)
   is abstract;

   type Event_Ptr_Type is
      access all Event_Type'Class;

   procedure Add(
      Event_Queue : in out
          Event_Queue_Type;
      Time : in Time_Type;
      Event : access Event_Type'Class);

   -- other methods not shown.
private

   type Event_Type is abstract record
      Id : Positive;
      Time : Time_Type;
   end record;

   function Sooner_Than(
      A, B : in Event_Ptr_Type)
   return Boolean;

   -- We are just going to put a thin
   -- wrapper around Ada ordered sets to
   -- implement our event queue.
   package Event_Queues is new
      Ada.Containers.Ordered_Sets(
         Event_Ptr_Type, Sooner_Than,
         "=");
   type Event_Queue_Type is new
      Event_Queues.Set with null record;

end Event_Queues;
```

Here's the circularity: Instantiating an event queue requires the simulation state type, but the simulation state must contain an event queue (of the instantiated type).

This is the best I've come up with so far:

```
private
   type Event_Queue_Wrapper_Type;

   type State_Type is
      record
         Future_Events : access
             Event_Queue_Wrapper_Type;
         -- other stuff not shown
      end record;

   package Simulation_Events is
     new Event_Queues(State_Type);

   -- Now we can define the wrapper
   -- to contain the event queue.
   type Event_Queue_Wrapper_Type is
      record
         Queue : Simulation_Events.
                     Event_Queue_Type;
      end record;
```

Is there a cleaner way to do this? The painful part is introducing a pointer to the event queue in State_Type just to break the circularity, when this seems superfluous and means I should make State_Type controlled to release the queue.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 28 Mar 2011 02:59:35 -0700 PDT
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

[…]

How about this:

```
generic
   type State_Type is tagged private;
package States_With_Event_Queues is

   type State_With_Event_Queue_Type
      is new State_Type with private;

   type Event_Type is abstract tagged
      limited private;

   procedure Handle(
      State : in out State_Type;
      Event : access Event_Type)
   is abstract;

   type Event_Ptr_Type is
      access all Event_Type'Class;

   procedure Add(
      Event_Queue : in out
          Event_Queue_Type;
      Time : in Time_Type;
      Event : access Event_Type'Class);

   -- other methods not shown.
private

   type Event_Type is abstract record
     Id : Positive;
     Time : Time_Type;
   end record;

   function Sooner_Than(
      A, B : in Event_Ptr_Type)
   return Boolean;

   -- We are just going to put a thin
   -- wrapper around Ada ordered sets
   -- to implement our event queue.
   package Event_Queues is new
     Ada.Containers.Ordered_Sets(
        Event_Ptr_Type, Sooner_Than,
        "=");
   type Event_Queue_Type is new
      Event_Queues.Set with
     null record;

   type State_With_Event_Queue_Type
```

```
   is new State_Type with record
     Event_Queue : Event_Queue_Type;
   end record;

end States_With_event_Queues;
```

Would that work?

BTW, why does Event_Type have to be limited? If it weren't limited, you would not need Event_Ptr_Type. Also, why does it have to be abstract?

From: Gene Ressler <gene.ressler@gmail.com>
Date: Mon, 28 Mar 2011 12:47:09 -0700 PDT
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

[…]

Interesting. Ought to be fine, but

> BTW, why does Event_Type have to be limited? If it weren't limited, you would not need Event_Ptr_Type.

Events are "serial numbered" at creation time and meant to be unique.

Pointer equality should be equivalent to content equality.

> Also, why does it have to be abstract?

Events only have meaning in their specializations to particular kinds, i.e. requests, mission starts, mission completions, etc. So the root event type is just an (abstract) placeholder, not something you'd want to create. […]

From: Simon Wright <simon@pushface.org>
Date: Mon, 28 Mar 2011 12:37:12 +0100
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

> BTW, why does Event_Type have to be limited? If it weren't limited, you would not need Event_Ptr_Type. Also, why does it have to be abstract?

When I made my events limited, it was because any two events are different. In this case, we have

```
type Event_Type is abstract record
  Id : Positive;
  Time : Time_Type;
end record;
```

so it probably won't make any sense for an Event to be copyable (what would Id be in the copy?). Of course the events could be privately non-limited. […]

You'll certainly need pointers somewhere, because Event_Type instances have to be held in the Queue. Indefinite_Ordered_Sets might do the trick, though. It would save the bother of having Handle deallocate the Event.

From: Gene Ressler <gene.ressler@gmail.com>
Date: Mon, 28 Mar 2011 15:38:25 -0700 PDT
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

[…]

Yes and no. The only reason for making Event_Queue generic is so that a State can be passed through the dispatching Handle call. Since this is an "in out" parameter that's invariably a record type, it's being passed by pointer behind the scenes. Yet there doesn't seem to be a way to take advantage of *this* pointer for breaking the circularity. In (eeewwww) C we could use a void* or other pointer trick here.)

E.g. if there were a generic parameter type for "access to incompletely declared type," we could pull it off by making Event_Queue generic in this way.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Mar 2011 22:01:24 -0500
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

[…] The way we handled this problem in Claw was to make State an abstract tagged type (rather than using a generic). Then, the parameter to the Handle call can be class-wide, which allows it to access any extension of the type.

I suspect that this also breaks the circularity (since the State type doesn't need to be defined until much later), but I haven't tried to figure it out.

From: Gene Ressler <gene.ressler@gmail.com>
Date: Mon, 28 Mar 2011 14:31:32 -0700 PDT
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

> […] How is the Event_Queue going to know which instance of State_Type to pass to a call of Handle (State, Event)?

There is currently only one state per simulation. The user of an Event_Queue instantiation is in a tight loop removing events from its event queue (which is embedded in its state), then calling Handle(State, Event) using its current state. This is dispatched to the correct handler based on (derived) event type. Handlers often schedule new events in the same event queue. It's essentially a callback. The state is the callback's execution context.

> Could the same State instance be passed to more than one Event_Queue?

It's easy to imagine simulations needing more than one event queue.

These would all need access through the state. But the current application has only a single queue.

> Will you really be using different State_Types? […]

Not in this version, but it's foreseeable.

[…]

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Mon, 28 Mar 2011 04:06:26 -0700 PDT
Subject: Re: Breaking a circularity
Newsgroups: comp.lang.ada

[…] Not sure you can do better in the sense of removing the access but perhaps you can bundle all that up into a generic wrapper? e.g.

```
package States is
  type State is record
    I : Integer;
  end record;
end States;

with Ada.Containers.Ordered_Sets;
generic
  type State_Type is private;
package Event_Queues is
  subtype Time_Type is Duration;
  type Event_Type is abstract tagged
    limited private;
  procedure Handle (
    State : in out State_Type;
    Event : access Event_Type)
  is abstract;
  type Event_Ptr_Type is
    access all Event_Type'Class;
  type Event_Queue_Type is private;
  procedure Add (
    Event_Queue : in out
      Event_Queue_Type;
    Time  : in  Time_Type;
    Event : access Event_Type'Class);
private
  type Event_Type is abstract tagged
limited record
    Id   : Positive;
    Time : Time_Type;
  end record;
  function Sooner_Than (
    A, B : in Event_Ptr_Type)
  return Boolean;
  package Event_Queues is new
Ada.Containers.Ordered_Sets
    (Event_Ptr_Type, Sooner_Than,
     "=");
  type Event_Queue_Type is new
    Event_Queues.Set
  with null record;
end Event_Queues;

with Event_Queues;
generic
  type State_Type is private;
package Event_Queue_Wrappers is
  type Event_Queue_Wrapper_Type is
    private;
private
  type EQW_State_Type is
```

```
    record
      State        : State_Type;
      Future_Events : access
        Event_Queue_Wrapper_Type;
    end record;
  package Simulation_Events is new
    Event_Queues (EQW_State_Type);
  type Event_Queue_Wrapper_Type
    is record
      Queue : Simulation_Events.
                Event_Queue_Type;
    end record;
end Event_Queue_Wrappers;

with Event_Queue_Wrappers;
with States;

procedure Temp is
  package EQW is new
    Event_Queue_Wrappers (
      State_Type => States.State);
  EQ :
    EQW.Event_Queue_Wrapper_Type;
begin
  null;
end Temp;
```

At least that decouples the state your interested in from the queue mechanism. […]

Forcing exception handling

From: iloAda <egzgheib@gmail.com>
Date: Mon, 28 Feb 2011 09:27:51 -0800 PST
Subject: Forcing Exception Handling
Newsgroups: comp.lang.ada

Hello everybody,

I was wondering if there is a way in Ada to force exception handling.

For instance, if there is a call to a function that may raise an exception, force the caller to handle that exception.

I was wondering if there is a compile time pragma that will instruct the compiler to force the handling of exceptions!!

[…]

From: Vinzent Hoefler
Date: Mon, 28 Feb 2011 19:34:46 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

> What's wrong with the raise statement ?

He wants the "exception" statement forced, I suppose.

And no, there is currently no such thing in Ada, if only because there's no way of telling the compiler which subroutine may raise an (user-defined) exception. Or, IOW, there is no "throws" statement as there is in Java.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 28 Feb 2011 21:35:03 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

I think the Ada Rapporteur Group considered the idea and rejected it. I seem to remember they had good reasons but I forgot which they were.

Maybe the first was that pretty much any statement can raise Storage_Error, and that Program_Error can be raised during elaboration, when there is no exception handler in place yet.

From: Georg Bauhaus <rm-host.bauhaus@maps.futureapps.de>
Date: Mon, 28 Feb 2011 21:38:26 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

IIRC, the new aspect specifications of Ada 2012 can be used to specify that some exception might be raised.

Not sure, though.

(Given that the feature is controversial in Java, not everyone might want them for this…)

From: Shark8 <onewingedshark@gmail.com>
Date: Mon, 28 Feb 2011 16:19:22 -0800 PST
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] IIRC, the new aspect specifications of Ada 2012 can be used to specify that some exception might be raised. […]

I'm honestly not sure this would be a good idea…

In fact, given that Ada doesn't have an "exception hierarchy" like Delphi does it may be a horrendous idea: There is no way to catch all exceptions because there is no way to catch all exceptions of a certain type (i.e. Storage_Error'Class) which is caused by the lack of some root-type "Exception_Root."

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 28 Feb 2011 21:54:08 -0600
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] I think the Ada Rapporteur Group considered the idea and rejected it. I seem to remember they had good reasons but I forgot which they were. […]

I think it is more of a case of being unable to come to consensus than "rejecting" it. (Same effect, ultimately.) Some people feel that Java's version doesn't work very well in practice, leaving to too many "when Some_Exception => null" handlers rather than consideration of why the exceptions are being propagated. That seems like bad software management to me. In any case, any such "contracts" in Ada would have to be optional (for compatibility reasons), and that might actually help reduce the problems.

We had such a discussion just a few weeks ago (not yet filed, so I can't point you at it), but in any case there won't be anything in Ada 2012 (we're very close to freezing that version, nothing significant can be added at this point).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 1 Mar 2011 09:34:44 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] Some people feel that Java's version doesn't work very well in practice, leaving to too many "when Some_Exception => null" handlers rather than consideration of why the exceptions are being propagated.

I never understood this. Why Some_Exception is not added to the contract instead? The rule is simple, if you don't know how to handle it, then don't.

> That seems like bad software management to me. In any case, any such "contracts" in Ada would have to be optional (for compatibility reasons), and that might actually help reduce the problems.

Unless Storage_Error and Program_Error made manageable contracted exceptions would have only limited use.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 1 Mar 2011 09:56:48 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

> In fact, given that Ada doesn't have an "exception hierarchy" like Delphi does it may be a horrendous idea: There is no way to catch all exceptions because there is no way to catch a all exceptions of a certain type (i.e. Storage_Error'Class) which is caused by the lack of some root-type "Exception_Root."

It is a question. Since exceptions are used in case-like clauses, more natural where a model of a discrete type. In order to have constructs like:

```
when Numeric_Exception'First..
     Numeric_Exception'Last =>
```

BTW, it shows the importance of extensible enumeration types, because exceptions should be that kind of type.

When exceptions are bunch of types like in C++ you would loose the above.

Also consider this:

```
declare
  Foo : exception:
```

```
begin
   ...
   raise Foo;
end;  -- Foo propagates out its scope!!
```

Unless Ada would have upward closures, that cannot work when Foo is a type.

I hope everybody agrees that upward closures should not be introduced.

A possibility to handle this is exception promotion, e.g. Foo to Program_Error, when Foo gets finalized.

BTW, contracted exception would solve this problem by making such programs illegal: each block would have an implicit contract not to propagate local exceptions.

From: iloAda <egzgheib@gmail.com>
Date: Tue, 1 Mar 2011 01:34:02 -0800 PST
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

Mmmmmm...That's what I was fearing!!

Actually the idea behind my question was that I'm working on a real time system, and as u know, we can't afford to a let a real time system crash because of an unhandled exception. Since I use some other libraries in my system (that weren't written by myself or anybody else I know) that will raise exceptions, I wanted to be forced to handle them. I was imagining that it could be done with some kind of a compiler pragma that will force the current package to handle all exception that may be raised, but as you guys pointed out, the compiler might not be able to know which procedure raises an exceptions (by contrast Java requires the addition of the "throws" instruction which tells the compiler what's gonna happen)

One solution might be to just put the following code everywhere (even though I don't like this):

```
exception
 when others =>
    -- Do something that will allow the
       system to keep on running
```

Have you guys done something like that before?

From: Vinzent Hoefler
Date: Tue, 01 Mar 2011 10:47:01 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] Since I use some other libraries in my system (that weren't written by myself or anybody else I know) that will raise exceptions, I wanted to be forced to handle them.

Better not to let them raised at all. I know, this may not be easy, but it's the most reliable solution.

> One solution might be to just put the following code everywhere (even though I don't like this):
>

```
> exception
>  when others =>
>     -- Do something that will allow the
   system to keep on running
```

This is dangerous. Especially when doing it "everywhere". Ignoring errors has never been a good idea, especially for the unexpected ones.

> Have you guys done something like that before?

Occasionally, yes. At the outermost task level to log the exception that caused the system to crash. If you do that inside a loop, you might be able to restart the affected task from there.

But if you have more than one task and they interact with each other, fault recovery like that gets quite complex, because then things like data consistency within the whole system may be at risk. And you don't want that to happen.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Tue, 1 Mar 2011 10:51:14 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

> Since I use some other libraries in my system […] that will raise exceptions, I wanted to be forced to handle them.

You should use some static analysis tool for that. E.g. SPARK.

```
> exception
>  When others =>
>     -- Do something that will allow the
   system to keep on running
```

That is of course meaningless, because you don't know WHAT happened in order to determine the SOMETHING to be done.

> Have you guys done something like that before?

No, because see above. The most close thing is:

```
when Error : others =>
   Trace ("Fatal:" &
        Ada.Exceptions.
        Exception_Information (Error));

+/-  raise;
```

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 01 Mar 2011 17:11:58 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] Have you guys done something like that before?

Or try AdaControl, rule Exception_Propagation

From: Mark Lorenzen
<mark.lorenzen@gmail.com>
Date: Tue, 1 Mar 2011 05:27:26 -0800 PST
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

If you cannot afford a crash, then you should make sure that exceptions do not occur in the first place, i.e. use SPARK as others have mentioned.

```
> exception
>  when others =>
>     -- Do something that will allow the
   system to keep on running
```

I'm currently working on a system where we can afford that the program crashes, but we will use the above "when others" construct to catch all possible exceptions (pre-defined and user-defined) on the outer-most level and then simply dump the exception trace to memory (a bit like Dmitry's example) for later (off-line) analysis after a reboot that doesn't whipe that specific part of memory.

We will use the addr2line utility to translate the addresses of the exception trace into file names and line numbers. You must compile your executable *with* debug information in order to perform the (offline) address translation. If you don't/can't run the executable with debug information (e.g. due to size constraints), then you can strip the debug information from the file that you are executing but keep it in the executable used for off-line analysis, and the address translation will still work correct.

[…]

From: Shark8
<onewingedshark@gmail.com>
Date: Tue, 1 Mar 2011 07:23:56 -0800 PST
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

What's wrong with handling it with OTHERS?

> A possibility to handle this is exception promotion, e.g. Foo to Program_Error, when Foo gets finalized.

A third is to convert Exceptions to a hierarchy as stated in my last post.

I'm not sure that's a Good Thing, but I think it's actually better than type extension; which I thought would be nice earlier… but the more I think on the problem the more it seems it would be utterly burdensome for implementors as well as possibly impossible-to-get-right.

From: Dmitry A. Kazakov
<mailbox@dmitry-kazakov.de>
Date: Tue, 1 Mar 2011 16:44:01 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

[…]

> What's wrong with handling it with OTHERS?

You have an object of non-existent type:

when Error : **others** =>

Assuming that Error is of Root_Exception_Type'Class, what should Error'Tag return?

[…]

> A third is to convert Exceptions to a hierarchy as stated in my last post.

No, that does not work, because the exception type would be dead in the handler. C++ model cannot work in Ada 2005+, because in Ada tagged types can be local.

Further problems are overlapping choices in the exception clause. If the semantics of

```
exception
  when X =>
  when Y =>
```

is dispatching on the tag of the current exception, which people suggesting class-wide exceptions have in mind. Then that would be incompatible with the choice

```
  when others =>
```

which has the semantics of a class-wide call. If you choose that semantics instead, then you would have to write

```
exception
  when X'Class =>
  when Y'Class =>
```

and lose the property that alternatives do not overlap. That is way unacceptable to me.

These are the reasons why I don't think that exception -> type mapping is such a good idea. I prefer exception -> value mapping + contracts.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 1 Mar 2011 18:00:32 -0600
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

I agree; I think that if we go this way all exceptions would have to appear in the contract. To make that manageable, we'd also need a way to define a "set" of exceptions so they could be referred to together. I personally think that is sufficient, and it would let the compiler tell you what it knows about exceptions being raised.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 1 Mar 2011 18:02:52 -0600
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] I hope everybody agrees that upward closures should not be introduced.

Right; this was one of the reasons that the intended plan (in Ada 2005) to make exceptions extensible tagged types eventually got dropped. It just doesn't work very well for Ada (it could have if done initially, but it is too late now).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 1 Mar 2011 18:11:28 -0600
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] Have you guys done something like that before?

[…] For the Janus/Ada compiler and tools, we used to have such handlers, but they covered up errors so well that we got rid of them. It's better for the compiler to just crash outright because that makes customers call us with a report right away rather than trying to figure out what went wrong. (And often we can figure out the bug just from the default error walkback and a bit of poking in the source code -- saves lots of debugging time.)

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 2 Mar 2011 09:28:55 +0100
Subject: Re: Forcing Exception Handling
Newsgroups: comp.lang.ada

> […] I agree; I think that if we go this way all exceptions would have to appear in the contract. […]

I thought about sort of conditional exceptions. E.g. storage error possibly raised when stack has less than n storage units. The compiler should estimate n in the cases mandated by the ARM. In other cases (recursion etc) the program is illegal, but the programmer may specify n explicitly using a pragma (he must know what he's doing, of course).

Another example of conditional exception would be "I don't raise, if you don't."

And we need a mechanism of exception propagation from slave tasks to their masters (when the task's exception contract is not null).

On local storage pools

From: Brian Drummond <brian_drummond@btconnect.com>
Date: Wed, 23 Feb 2011 19:01:22 +0000
Subject: Using local storage pools...
Newsgroups: comp.lang.ada

I am trying to learn a little about storage pools, with a view to (hopefully) using local pools to improve the Binary_Trees benchmark in the same way as some of the faster C benchmarks.

[…] they do not explicitly free each tree node (the "free" call has been deleted!) but free the entire pool at the end of the loop. But if that's valid, Ada should be able to do the same.

http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gnat_ugn_unw/Some-Useful-Memory-Pools.html

suggests System.Pool_Local offers a way to do likewise - a pool that is automatically reclaimed when it goes out of scope.

This turns out to have its own performance problem, but that is another story…

The question(or four) for now is … should the following really raise Storage_Error, i.e. am I doing something silly, and if so, what?

Or is this a bug in GNAT?

NOTE - using System.Pool_Global.Unbounded_No_Reclaim_Pool (commented out) instead of the pool shown, works as expected.

(Tested on GCC4.5.0 and GNAT GPS 2010)

[…]

```
with System.Pool_Local;
with System.Pool_Global;
with Ada.Unchecked_Deallocation;

procedure pooltest is

  type Node;
  type Treenode is access Node;
  type Node is record
    Left  : Treenode := null;
    Right : Treenode := null;
    Item  : Integer  := 0;
  end record;

  P : System.Pool_Local.
      Unbounded_Reclaim_Pool;
  --P : System.Pool_Global.
  --    Unbounded_No_Reclaim_Pool;
  for Treenode'Storage_Pool use P;

  procedure free is new
     Ada.Unchecked_Deallocation(
       Node, Treenode);

  TestNode : Treenode;

begin
  Testnode := new Node'(null, null, 1);
  free(Testnode);
end pooltest;
```

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 23 Feb 2011 21:51:20 +0100
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

This looks like a genuine bug at s-pooloc.adb:114. To trigger the bug, two conditions must hold simultaneously:

- the pool contains exactly one allocated object.
- the user calls Unchecked_Deallocation on this object.

[…]

This procedure is *not* called by the finalization of the pool, which simply walks the linked list of nodes and deallocates each one, but does not modify any nodes.

Because this pool is intended for use without any explicit Unchecked_Deallocation, I would qualify this bug as minor.

The workaround, in your case, is to simply not do any Unchecked_Deallocation and let the finalization of the storage pool do the deallocation.

From: Simon Wright <simon@pushface.org>
Date: Wed, 23 Feb 2011 21:01:51 +0000
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

Looks to me like a bug in GNAT. I expect it's not been found before because the point of this pool type is that you *don't* deallocate, but leave it up to finalization!

I'm not quite clear how we can use this pool in the binary-trees benchmark, since the rules say you create a class to represent the tree, which must include the access types, and the storage pool needs to exist before the access types are used. Perhaps a generic?

Even then, why not just declare an access type in the appropriate scope and let the compiler figure out how to deallocate storage on scope exit? (in other words, I don't see why GNAT includes System.Pool_Local in the first place).

Oh, perhaps it's so that you can actually create such a generic?

From: Brian Drummond <brian_drummond@btconnect.com>
Date: Thu, 24 Feb 2011 00:00:00 +0000
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

> Looks to me like a bug in GNAT. I expect it's not been found before because the point of this pool type is that you *don't* deallocate, but leave it up to finalization!

Probably, but the doc doesn't say that freeing is illegal, and the source file (s_pooloc.ads/b) does have a deallocate routine, so I believe it ought to work…

I can imagine cases where you free to improve memory footprint, but for whatever reason you can't (or don't) entirely eliminate memory leaks (so cleanup at a well-defined point)- in that case, the pool should allow both strategies.

When I created a "local pool" per task, it only left scope on termination, therefore the memory footprint blew up without freeing. You could argue this was my programming error.

> I'm not quite clear how we can use this pool in the binary-trees benchmark, since the rules say you create a class to represent the tree, which must include the access types, and the storage pool needs to exist before the access types are used. Perhaps a generic?

Exactly. In my binary_trees experiments I have made the Treenode package a generic. That part seems to work…

> Even then, why not just declare an access type in the appropriate scope and let the compiler figure out how to deallocate storage on scope exit? (in other words, I don't see why GNAT includes System.Pool_Local in the first place).

I think it's so you can also declare a storage pool along with the locally-declared access type, (i.e. locally instantiated generic) such that the entire pool goes out of scope and can be deleted on leaving the scope.

Using the main pool, you would have to deallocate the locally declared access types but leave the longer-lived objects…

Unfortunately, the current implementation of System.Pool_Local derives from System.Pool_Global, so it is implemented on the main heap, and its "Finalize" simply loops over every object freeing them individually. Therefore there is no performance gain this way…

From: Brian Drummond <brian_drummond@btconnect.com>
Date: Thu, 24 Feb 2011 00:27:17 +0000
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

> Because this pool is intended for use without any explicit Unchecked_Deallocation, I would qualify this bug as minor.

I believe it should support both strategies (especially since it has a "deallocate") but I can't argue it's anything other than minor if I'm the first to find it!

> The workaround, in your case, is to simply not do any Unchecked_Deallocation and let the finalization of the storage pool do the deallocation.

Which was the original intent. Thanks for the detective work!

[…]

From: Brian Drummond <brian_drummond@btconnect.com>
Date: Thu, 24 Feb 2011 17:04:04 +0000
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

>> Should I also report it to either Debian, or mainstream GCC?

> To the GCC bugzilla, please. The bug has been confirmed on 4.4.5 and 4.5.2 but my investigation reveals it has been present at least since 2001 (first commit into the public GCC source repository).

Done:- Bug 47880.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 23 Feb 2011 21:42:49 +0100
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

Why don't you implement an arena pool? It is a quite common technique to allocate nodes of a tree in an arena and never deallocate them explicitly.

I am always use this for the nodes of the abstract syntax tree (AST). Note that since arena never deallocates, allocation in arena becomes trivial.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Feb 2011 10:26:26 +0100
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

> […] the shootout rules explicitly disallow implementing your own pool.

Huh:

```
type Nodes_Arena is array
   (1..<huge-number>) of
   aliased Node;
Last_Allocated : Integer := 0;

function Create
  return not null access Node is
begin
  Last_Allocated := Last_Allocated +
                    1;
  return Nodes_Arena
      (Last_Allocated)'Access;
end Create;
```

This is an implementation of arena. How can this be disallowed?

> (But importing one from available libraries seems to be permitted; so is there a suitable candidate?)

You can try this one:

http://www.dmitry-kazakov.de/ada/components.htm#7.1

From: Georg Bauhaus <rm-host.bauhaus@maps.futureapps.de>
Date: Thu, 24 Feb 2011 10:51:51 +0100
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

"Each program should

...

"allocate, walk, and deallocate many bottom-up binary trees

  * allocate a tree

* walk the tree nodes, checksum the node items (and maybe deallocate the node)

* deallocate the tree

...

"Note: these programs are being measured with the default initial heap size - the measurements may be very different with a larger initial heap size or GC tuning.

"Please don't implement your own custom memory pool or free list.</>"

"The binary-trees benchmark is a simplistic adaptation of Hans Boehm's GCBench,

(http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_bench/applet/GCBench.java)

which in turn was adapted from a benchmark by John Ellis and Pete Kovac."

http://shootout.alioth.debian.org/u32/performance.php?test=binarytrees

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 24 Feb 2011 11:09:36 +0100
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

I see no contradiction. Tree is allocated. It is deallocated too (when array is destroyed). […]

> "Please don't implement your own custom memory pool or free list.</>"

They seem trying to exclude pool access overhead from the measure, but they have no working idea how. All these "benchmarks" are measuring things having nothing or little to do with the manifested objective.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Thu, 24 Feb 2011 07:34:35 -0500
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

> […] they do not explicitly free each tree node (the "free" call has been deleted!) but free the entire pool at the end of the loop. […]

[…] This technique is commonly used.

I have used it in various compilers and static analysis tools and other programs. Whenever you have large numbers of small heap-allocated objects that all have roughly the same lifetime, this technique simplifies the program, makes it more efficient, and helps avoid dangling pointers.

> http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gnat_ugn_unw/Some-Useful-Memory-Pools.html

suggests System.Pool_Local offers a way to do likewise - a pool that is automatically reclaimed when it goes out of scope.

Pool_Local is not what you want (for efficiency) in this case.

What you want is a pool that allocates large chunks of memory, and allocates individual small objects within that, and deallocates the whole thing (all the chunks) at the end (Finalize). For efficiency, you never call Unchecked_Deallocation.

I'm not sure if such a pool is part of gnatcoll, but if not, I'll probably add it to gnatcoll someday.

Pool_Local would be more appropriate if the allocated objects are large (and usually have similar lifetimes).

[…]

I don't see anything wrong with your code. If you report it to AdaCore (report@adacore.com) it will get fixed (at low priority, if you're not a supported customer). I'm interested in storage pools -- maybe I'll be assigned to fix the bug (if it is a bug).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 25 Feb 2011 21:02:34 -0600
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

> […] they do not explicitly free each tree node[…] but free the entire pool at the end of the loop. […]

This sounds like a job for Ada 2012 subpools. Of course, to use them, you have to have an Ada 2012 compiler (and I don't think there is an implementation at the moment - we [the ARG] just approved the AI last week).

If the pools are completely disjoint and you can live with local access types, you can use individual local pool objects to do the job, but that typically isn't the case in practice.

You can write a pool to do mass deallocations, but such a pool is always erroneous according to the language (both Ada 95 and Ada 2005), so there can be no guarantee that it will work on another compiler. [Bob Duff will tell you that the intent was such things would work, and they often will - as compilers don't try to do bad things unless there is a good reason - but the language doesn't provide any help.]

From: Pascal Obry <pascal@obry.net>
Date: Sat, 26 Feb 2011 19:59:47 +0100
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

>> This sounds like a job for Ada 2012 subpools. […]

> Hum, can you tell me more about this AI? What's the number?

I think I found it:

http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05s/ai05-0111-3.txt?rev=1.8

[…]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 25 Feb 2011 21:07:01 -0600
Subject: Re: Using local storage pools...
Newsgroups: comp.lang.ada

[…]

Another thought: the Multiway_Tree container may very well have this behavior. The bounded version *has* to have this behavior. Not sure what effect using the container would have on the benchmark, but it would be an interesting option to try. (Of course, this is another Ada 2012 feature, but it is easy to implement in existing compilers and I believe it is available in recent versions of GNAT.)

On alternative design for Ada 2005/2012

From: Tom Moran <tmoran@acm.org>
Date: Thu, 21 Apr 2011 23:05:26 +0000 UTC
Subject: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

Randy,

A lot of Claw used/tested new Ada 95 features. What would you *design* differently with Ada 2005 or 2012? (Not syntactic sugar).

Similar question to other folks.

From: Lucretia <Lucretia9000@yahoo.co.uk>
Date: Thu, 21 Apr 2011 18:10:09 -0700 PDT
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…] Interfaces if the FSF GNAT doesn't produce an ICE.

[…]

P.S: The containers should've used interfaces!

From: Shark8 <onewingedshark@gmail.com>
Date: Thu, 21 Apr 2011 18:23:25 -0700 PDT
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…]

Indeed they should have.

It really is too bad that they chose to emulate Java-style interfaces rather than Delphi-style interfaces; with Delphi's properties you can have a 'field' of an object which can be calculated (a procedure or function) or a renaming of an internal field, for both reading and writing to the property. Read-only and write-only properties are also do-able.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Apr 2011 19:36:04 -0500
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…]

Well, obviously we would have used overriding indicators from the beginning.

(No surprise there, the indicators were one of the first Ada 2005 features -- they got added in part because of a hard push I made because of my Claw experience.) We would have used "private with" a lot, too. Both of these are time-consuming to retrofit, but they wouldn't have changed the design a lot.

I would have liked to have been able to make some or all of the types limited; one reason we didn't do that was the loss of the function syntax. Had we done that, we would have spent less time trying to get finalization to work right (the "clone" semantics is complicated to implement, and not that easy to work with, either).

For Ada 2012, I would have definitely used the iterators to walk lists.

Probably also would have used some of the contract features (preconditions, etc.) although they don't work well in the Claw model (where objects can change state asynchronously to the program).

I'm sure there is other stuff that would have helped in specific cases (null exclusions come to mind), but I don't think the design would have been wildly different. I don't think we would have used any interfaces, for instance.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 22 Apr 2011 19:41:59 -0500
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…]

>P.S: The containers should've used interfaces!

We tried that with the new Queue containers, and IMHO it is a disaster. You have to use an extra instantiation and extra formal parameters for a feature that you are not going to use on 90% of the programs. (Typically, you only use one kind of queue in a program.)

The introduction of the interfaces also forced a horrible structure which requires the implementation to declare all of its internal data structures in a *visible* nested package. (This is partially the fault of protected types, which don't allow nested types for good reasons -- but if we didn't have the interface we could hide the protected type in the private part.)

Some of this could be fixed with substantial violence to the Ada visibility and implementation models, but it is not clear that what you would end up with be Ada. Not likely to happen, and surely not before Ada 2020.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 23 Apr 2011 08:44:27 +0200
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…]

Yes, I have same experience. Interfaces do not work good with generics, which should not surprise anybody.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sat, 23 Apr 2011 14:27:24 -0700 PDT
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…]

> Yes, I have same experience. Interfaces do not work good with generics, which should not surprise anybody.

Agree here. This approach seems to be bearable in Java, where standard collections (java.util.*) are all both generics and interface-based, but I think this is because all this Java stuff is more dynamically than statically typed. On the user perspective, the benefits of these interfaces are close to zero and the very few potential advantages (like the ability to change the concrete collection type without modifying all user code) are more results of Java not having type aliasing (typedefs or renamings) features.

C++ uses non-interface approach and I have never seen a situation where this was an issue. Ada got it right, too.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 26 Apr 2011 18:12:02 -0500
Subject: Re: design changes per Ada 2005/12?
Newsgroups: comp.lang.ada

[…]

> Things like iterators could be implemented ok with interfaces.

And indeed they are (see AI05-0139-2). But it is interesting that iterators were the only example where interfaces worked. We tried and discarded models using interfaces for both implicit dereferencing and indexing, because they either ended up as nothing more than a marker (which can be accomplished in a straightforward way without using any interfaces), or because they required violating visibility/overloading rules.

Ada bindings to C++ class templates

From: David Sauvage <sauvage.david@gmail.com>
Date: Sat, 5 Mar 2011 01:36:41 -0800 PST
Subject: Ada bindings to C++ class templates
Newsgroups: comp.lang.ada

Hi, I'm trying to know more about how to write Ada bindings to C++ class templates.

Unfortunately, I have found no documentation concerning this subject.

May be we could use the C++ STL vector class template [1] as a basis for the discussion & hints (if any).

I would think that the Ada binding would start from a generic package.

I would prefer to implement the necessary intermediate C layers to make the binding compiler portable if possible. (instead of using g++ -fdump-ada-spec for example)

Any help & hints would be appreciated.

[1] http://en.wikipedia.org/wiki/Vector_(C++)

From: Maciej Sobczak <maciej@msobczak.com>
Date: Sat, 5 Mar 2011 02:56:56 -0800 PST
Subject: Re: Ada bindings to C++ class templates
Newsgroups: comp.lang.ada

> I'm trying to know more about how to write Ada bindings to C++ class templates.

There is no direct way to couple Ada with C++ at the level of class templates, so…

> I would prefer to implement the necessary intermediate C layers to make the binding compiler portable if possible.

and this is the recommended way to go - create a layer of thin C interfaces to whatever C++ stuff you want to use and bind to that interface. This seems to be time consuming, but in fact is not - the actual application code on both sides tends to be much bigger and this allows to quickly amortize the additional effort to create the C interface layer.

The problem is that the C interface layer will not be generic and will force you to "flatten" the type space to what is available in C. On one hand this seems to be a severe limitation (it is from the design point of view), but on the other hand (the deployment point of view) it isolates the two sides well enough to allow complete replacement of either component, and it also open some new opportunities, like using the low-level component in interpreters (Python, Tcl, etc.). So, in short - yes, creating a C layer is a preferred way to go and even though requires some up-front effort, it brings long-term benefits.

Adding proof function definitions to SPARK

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Sun, 27 Feb 2011 04:51:42 -0800 PST

Subject: SPARK: Defining the meaning of a proof function
Newsgroups: comp.lang.ada

One significant gap in the SPARK language is the absence of any way to define the meaning of a proof function (i.e. a function declared in annotations). The only way to do this is in a user proof rule that is read by the Simplifier.

If the user were able to define the meaning of a proof function in the SPARK text then this could be incorporated into the VCs generated by the Examiner. This would remove the need for a number of user rules and make completion of the proofs simpler, quicker and less error-prone.

At present, a function operation can be given a return annotation, and this is used to add a hypothesis about the value returned by a call of that function - but only where that operation appears in code. Hypotheses are not added when the equivalent proof function appears in a proof context (pre/post, assert, check).

As an experiment I extended my SPARKRules utility to mimic the effect of adding proof function definitions to SPARK. The extended program edits the Examiner generated VCs to add the hypotheses that (I think) the Examiner would add if it had definitions for proof functions.

The results were very positive and suggest that the combination of adding proof function definitions to SPARK and using more capable solvers via Victor has the potential to make proof a great deal simpler and quicker.

I used the current version of the Tokeneer software as the trial application. Full details of the work and the results are in:

http://www.sparksure.com/resources/Proof_Function_Definitions.pdf

Summarising the results:

- The current version of Tokeneer has 84 distinct user rules (some very complex) that create 97 rules in 26 user rule files. These complete the proofs of 101 VCs in 43 operations. (There are a further 24 VCs that have review proofs.)
- If the proof function definitions are added and Victor used to submit the VCs to alt-ergo, then there are just 10 VCs (out of 101) left unproven in 8 operations (out of 43). Completion of these remaining VCs requires 17 user rules (all of which are quite simple).

It is the combination of proof function definitions and Victor that is effective:

Adding the proof function definitions leaves 58 VCs in 30 operations.

Using Victor leaves 76 VCs in 36 operations.

I hope that this experiment will make the enhancement more likely - supported customers are encouraged to use their influence to promote this change.

Although I will not make this extended version of SPARKRules (called SPARKRulesPF) generally available (since it is not a properly tested program) I am prepared to make it available on request provided that the reason for the request is given and an undertaking made to share any results. Replies to this message will reach me, or you can use a sparksure.com address. The program compiles and runs on both Windows 7 and openSUSE 11.3 using the latest versions of GNAT and SPARK (but Victor is available only on Linux at present).

From: Alexander Senier <mail@senier.net>
Date: Mon, 28 Feb 2011 09:21:33 +0100
Subject: Re: SPARK: Defining the meaning of a proof function
Newsgroups: comp.lang.ada

[…]

> One significant gap in the SPARK language is the absence of any way to define the meaning of a proof function (i.e. a function declared in annotations). The only way to do this is in a user proof rule that is read by the Simplifier.

There is another way to define the meaning of proof functions. However, it is not integrated into the SPARK source text either. The current release of the Isabelle theorem prover [1] adds the HOL/SPARK proof environment. This environment can read in the VCs produced by the Examiner and the Simplifier (and associated FDL rules etc.) and present them as proof contexts in Isabelle.

Abstract SPARK proof functions can be associated with concrete Isabelle functions (of suitable type, checked by Isabelle). Using HOL/SPARK can be advantageous, as e.g. existing powerful proof strategies, automated external provers and existing proofs may be used to prove open VCs.

Furthermore, it is much harder to introduce logical inconsistencies in Isabelle/HOL than in hand-written rule files.

Examples for using HOL/SPARK can be found in "src/HOL/SPARK/Examples" in the Isabelle2011 distribution. Unfortunately, there is no documentation available currently (being improved at the moment).

[1] http://isabelle.in.tum.de

On constraints in extended return

From: Simon Wright <simon@pushface.org>
Date: Sat, 19 Mar 2011 10:42:45 +0000
Subject: Constraints in extended return
Newsgroups: comp.lang.ada

This code transposes a matrix:

```
function Transpose (
    M : Complex_Matrix)
  return Complex_Matrix
is
begin
  return
    Result : Complex_Matrix (
                M'Range (2),
                M'Range (1)) do
      for J in M'Range (1) loop
        for K in M'Range (2) loop
          Result (K, J) := M (J, K);
        end loop;
      end loop;
  end return;
end Transpose;
```

This is all very well for providing a value where no constraint is otherwise imposed, for example as an actual in a subprogram call, but what about the case where there is a prior constraint?

```
Input : Complex_Matrix (1 .. 2, 11 .. 12);
Output : Complex_Matrix (1 .. 2, 1 .. 2);
begin
  Input := (...);
  Output := Transpose (Input);
```

Is there any way for the extended return to determine the constraints of the 'target'? I suspect not, but the language in RM6.5 is deep.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Sat, 19 Mar 2011 09:30:18 -0400
Subject: Re: Constraints in extended return
Newsgroups: comp.lang.ada

[…]

> […] what about the case where there is a prior constraint?

You get Constraint_Error if the constraint is wrong.

But this isn't directly related to extended return statements.

You get the same thing for old-fashioned returns:

```
function F(…) return String is
begin
  …
  return "Hello";
end F;


X : String (1..100);


X := F(…); -- Constraint_Error
```

> Is there any way for the extended return to determine the constraints of the 'target'?

No. You can do that for 'out' parameters, but unfortunately not for function results. So normally, you would avoid

constraining at the call site, and do things like:

```
Output : constant
   Complex_Matrix := Transpose (Input);
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 19 Mar 2011 14:51:23 +0100
Subject: Re: Constraints in extended return
Newsgroups: comp.lang.ada

[…]

> No. You can do that for 'out' parameters, but unfortunately not for function results.

Fortunately you mean, because how could we otherwise create unconstrained objects?

From: Simon Wright <simon@pushface.org>
Date: Sat, 19 Mar 2011 15:55:00 +0000
Subject: Re: Constraints in extended return
Newsgroups: comp.lang.ada

> Output : constant Complex_Matrix := Transpose (Input);

Thanks, that's what I thought would be the case. This will constrain my solution :-) but we all need boundaries!

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 21 Mar 2011 21:02:32 -0500
Subject: Re: Constraints in extended return
Newsgroups: comp.lang.ada

[…]

> Is there any way for the extended return to determine the constraints of the 'target'? I suspect not, but the language in RM6.5 is deep.

Why do we care what the bounds are? Assignments "slide" bounds; all that is required is that the lengths of the array dimensions match. This is true whether this is an extended return or a regular return (at least it had better be; it would be awful to have the semantics change depending on the form of the return statement).

For instance,

```
Output := Input;
```

is both a legal assignment and does not raise Constraint_Error (both dimensions have length 2).

Now, if you get the lengths wrong, you have a problem, but not the actual bounds.

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

July 03-06 — 4th **International Conference on Software Language Engineering** (SLE'2011), Braga, Portugal. Topics include: Formalisms used in designing and specifying languages and tools that analyze such language descriptions; Language implementation techniques; Program and model transformation tools; Language evolution; Approaches to elicitation, specification, or verification of requirements for software languages; Design challenges in SLE; Applications of languages including innovative domain-specific languages or "little" languages; etc.

July 05-07 — 15th **International Conference on System Design Languages** of the SDL Forum Society (SDL'2011), Toulouse, France. Topics include: Industrial application reports (industrial usage and experience reports, tool engineering and frameworks, domain-specific applicability, such as aerospace, automotive, control, ...); Evolution of development tools and languages (domain-specific profiles and extensions, modular language design, semantics and evaluation, methodology for application, standardization activities); Modeling in multi-core and parallel applications; Education and Promotion of System Design Languages; etc.

July 07-09 — 23rd **International Conference on Software Engineering and Knowledge Engineering** (SEKE'2011), Miami Beach, USA. Topics include: Integrity, Security, and Fault Tolerance; Reliability; Industry System Experience and Report; Component-Based Software Engineering; Embedded Software Engineering; Reverse Engineering; Programming Languages and Software Engineering; Program Understanding; Software Assurance; Software dependability; Software Engineering Tools and Environments; Software Maintenance and Evolution; Software product lines; Software Quality; Software Reuse; Software Safety; Software Security; Software Engineering Case Study and Experience Reports; etc.

July 13-14 — 11th **International Conference on Quality Software** (QSIC'2011), Madrid, Spain. Topics include: Software quality (review, inspection and walkthrough, reliability, safety and security, ...); Evaluation of software products and components (static and dynamic analysis, validation and verification); Economics of software quality; Formal methods (program analysis, model construction, ...); Applications (component-based systems, distributed systems, embedded systems, enterprise applications, information systems, safety critical systems, ...); etc.

July 14-20 — 23rd **International Conference on Computer Aided Verification** (CAV'2011), Snowbird, Utah, USA. Topics include: Algorithms and tools for verifying models and implementations, Program analysis and software verification, Verification methods for parallel and concurrent hardware/software systems, Applications and case studies, Verification in industrial practice, etc.

July 18-21 — 6th **International Conference on Software and Data Technologies** (ICSOFT'2011), Seville, Spain. Topics include: Software Engineering, Software Maintenance, Model-Driven Engineering, Software Economics, Reverse Engineering, Programming Languages, Distributed Systems, Security and Privacy, etc.

☺ July 25-29 — 25th **European Conference on Object-Oriented Programming** (ECOOP'2011), Lancaster, UK. Topics include: all areas of object technology and related software development technologies, such as Analysis and design methods and patterns; Distributed, concurrent, real-time systems; Language design and

implementation; Modularity, components, services; Software development environments and tools; Type systems, formal methods; Compatibility, software evolution; etc.

☺ July 26 6th **Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems** (ICOOOLPS'2011). Topics include: efficient implementation and compilation of OO languages in various application domains ranging from embedded and real-time systems to desktop systems.

August 15-18 6th **IEEE International Conference on Global Software Engineering** (ICGSE'2011), Helsinki, Finland. Topics include: Strategic issues in distributed development (cost-benefit-risk analysis, ...); Methods and tools for distributed software development (requirements engineering, design, coding, verification, testing and maintenance, development governance); Empirical studies and lessons learnt from distributed development; etc.

August 22-27 6th **International Conference on Software Engineering Advances** (ICSEA'2011), Nice, France. Topics include: Advances in fundamentals for software development; Advanced mechanisms for software development; Advanced design tools for developing software; Software security, privacy, safeness; Open source software; Software deployment and maintenance; Software economics, adoption, and education; etc.

☺ August 29-30 16th **International Workshop on Formal Methods for Industrial Critical Systems** (FMICS'2011), Trento, Italy. Topics include: Design, specification, code generation and testing based on formal methods; Verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability; Tools for the development of formal design descriptions; Case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; Impact of the adoption of formal methods on the development process and associated costs; Application of formal methods in standardization and industrial forums; etc.

August 29-31 5th IEEE **International Conference on Theoretical Aspects of Software Engineering** (TASE'2011), Xi'an, China. Topics include: Embedded and Real-Time Systems; Program Analysis; Software Architectures and Design; Component-Based Software Engineering; Reverse Engineering and Software Maintenance; Aspect and Object Orientation; Dependable Concurrency; Specification and Verification; Model-Driven Engineering; Software Safety, Security and Reliability; Static Analysis; etc.

Aug 29 – Sep 02 15th IEEE **International Enterprise Computing Conference** (EDOC'2011), Helsinki, Finland. Topics include: the full range of engineering technologies and methods contributing to intra- and inter-enterprise distributed application systems; industry specific solutions, e.g. for aerospace, automotive, finance, logistics, medicine and telecommunications; etc.

☺ Aug 30 – Sep 02 **International Conference on Parallel Computing** 2011 (ParCo'2011), Gent, Belgium. Topics include: all aspects of parallel computing, including applications, hardware and software technologies as well as languages and development environments, in particular Applications of multicores, GPU-based applications, Parallel programming languages, compilers and environments, Best practices of parallel computing, etc.

Au 30–Se 02 ParCo2011 - **Parallel Computing with FPGAs** (ParaFPGA'2011). Topics include: programming environments for FPGAs, parallel languages and design tools for FPGA application development, FPGA-based parallel applications, etc.

Aug 30 – Sep 02 37th EUROMICRO **Conference on Software Engineering and Advanced Applications** (SEAA'2011), Oulu, Finland. Topics include: information technology for software-intensive systems; tracks on Lean Software Product Development (LSPD), Resilient Systems, Robotics and Critical Infrastructures (RSRCI), Embedded Software Engineering (ESE), etc.

Au 30–Se 02 **Track on Software Process and Product Improvement** (SPPI'2011). Topics include: Value-based software engineering; Open source software and software quality; Agile and lean development; Software reuse, product lines, and software ecosystems; Dependability, safety, security, or usability; Quality assurance, inspections, testing; Software evolution; Innovative approaches to software development; Empirical studies and experimental approaches; etc. etc.

☺Au 30–Se 02 **Track on Embedded Software Engineering** (ESE'2011). Topics include: Design and implementation of embedded software; Programming methodologies and languages for embedded software; Testing and certification of embedded software; Embedded software verification and validation; Software-intensive systems applications, e.g., in automotive, avionics, energy, industrial automation, health care, and telecommunication; etc.

September 05-09 8th **Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering** (ESEC/FSE'2011), Szeged, Hungary. Topics include: Case studies and experience reports; Engineering of distributed/parallel software systems; Engineering of embedded and real-time software; Engineering secure software; Reverse engineering and maintenance; Software architecture and design; Software components and reuse; Software dependability, safety and reliability; Software tools and development environments; Theory and formal methods; etc.

September 12-14 16th **European Symposium on Research in Computer Security** (ESORICS'2011), Leuven, Belgium. Topics include: Accountability, Information Hiding, Information Flow Control, Integrity, Formal Security Methods, Language-Based Security, Risk Analysis and Management, Security Verification, Software Security, etc.

September 13-15 **Forum on specification & Design Languages** (FDL'2011), Oldenburg, Germany. Topics include: the application of languages, their associated design methods and tools for the design of electronic systems. Deadline for submissions: August 30, 2011 (onsite meetings).

September 13-16 5th **European Conference on Software Architecture** (ECSA'2011), Essen, Germany. Topics include: software tools and environments for architecture-centric software engineering; component-based models, middleware, component-based deployment; technology of components and component-based frameworks; industrial applications, case studies, best practices and experience reports; architecture description languages and metamodels; etc.

☺ September 13-16 40th **International Conference on Parallel Processing** (ICPP'2011), Taipei, Taiwan. Topics include: all aspects of parallel and distributed computing, such as Compilers, Programming Models and Languages, Multi-core and Parallel Systems, etc.

Sep 14-16 15th **International Real-Time Ada Workshop** (IRTAW'2011), Liébana, Cantabria, Spain. In cooperation with Ada-Europe.

☺ September 19-21 FedCSIS2011 - 3rd **Workshop on Advances in Programming Languages** (WAPL'2011), Szczecin, Poland. Topics include: Compiling techniques; Domain-specific languages; Formal semantics and syntax; Generative and generic programming; Languages and tools for trustworthy computing; Language concepts, design and implementation; Model-driven engineering languages and systems; Practical experiences with programming languages; Program analysis, optimization and verification; Program generation and transformation; Programming tools and environments; Proof theory for programs; Specification languages; Type systems; etc.

☺ September 19-23 11th **International Conference on Parallel Computing Technologies** (PaCT'2011), Kazan, Russia. Topics include: all aspects of the applications of parallel computer systems; methods and tools for parallel solution of large-scale problems; languages, environment and software tools supporting parallel processing; etc.

September 21-23 9th **International Conference on Formal Modeling and Analysis of Timed Systems** (FORMATS'2011), Aalborg, Denmark. Topics include: Methods and Tools (techniques, algorithms, data structures, and software tools for analyzing timed systems and resolving temporal constraints, such as scheduling, worst-case execution time analysis, optimization, model checking, testing, constraint solving); Applications (adaptation and specialization of timing technology in application domains in which timing plays an important role, such as real-time software, problems of scheduling in manufacturing and telecommunication, ...); etc.

September 22-23 5th **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2011), Banff, Alberta, Canada. Topics include: Generative programming, metaprogramming; Product-line architectures; Analysis of language support for generative programming; Semantics, type-systems of generative programs; Case Studies and Demonstration Cases; etc.

Sep 25 – Oct 01 27th IEEE **International Conference on Software Maintenance** (ICSM'2011), Williamsburg, VA, USA. Topics include: reverse engineering and re-engineering; static and dynamic analysis; software migration and renovation; maintenance and evolution process; mining software repositories; empirical studies in software maintenance and evolution; testing, only in relation to maintenance (e.g., regression testing); etc.

☺ September 26-30 CBSoft2011 - 15th **Brazilian Symposium on Programming Languages** (SBLP'2011), Sao Paulo, Brazil. Topics include: the fundamental principles and innovations in the design and implementation of programming languages and systems; such as: Programming paradigms and styles, including object-oriented, real-time, multithreaded, parallel, and distributed programming; Program analysis and verification, including type systems, static analysis and abstract interpretation; Programming language design and implementation, including new programming models, programming language environments, compilation and interpretation techniques; etc.

☺ September 28-30 **Facing the Multicore-Challenge II Conference**, Karlsruhe, Germany. Topics include: Parallel programming models, environments and languages; Library and tool support; Scalability issues and portability of software solutions; Compiler techniques and code optimization strategies; Practice and experience of multicore programming; Parallel applications and benchmarks; etc. Deadline for submissions: July 17, 2011 (full papers), September 4, 2011 (short talks, posters).

September 29-30 10th **International Conference on Intelligent Software Methodologies, Tools and Techniques** (SoMeT'2011), Saint Petersburg, Russia. Topics include: Software methodologies, and tools for robust, reliable, non-fragile software design; Software developments techniques and legacy systems; Automatic software generation versus reuse, and legacy systems; Software evolution techniques; Agile Software and Lean Methods; Formal methods for software design; Software maintenance; Software security tools and techniques; Formal techniques for software representation, software testing and validation; Software reliability, and software diagnosis systems; Model Driven Development (DVD), code centric to model centric software engineering; etc.

☺ 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. Deadline for submissions: July 8, 2011 (student research competition).

☺ 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.

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).

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.

November 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 submissions: September 16, 2011 (student symposium). 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. Deadline for submissions: July 8, 2011 (abstracts), July 12, 2011 (papers).

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. Deadline for submissions: September 5, 2011 (abstracts), September 13, 2011 (papers).

☺ 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 25-30 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: July 14, 2011 (abstracts round 2), July 18, 2011 (papers round 2), October 6, 2011 (abstracts round 2), October 10, 2011 (papers round 2).

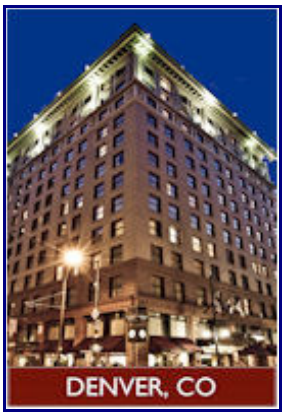
♦ June 11-15 17th International Conference on Reliable Software Technologies - Ada-Europe'2012. Stockholm, Sweden.

SIGAda 2011
DENVER, COLORADO

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

Sponsored by SIGAda, ACM's Special Interest Group on the Ada Programming Language,
in cooperation with SIGAPP, SIGBED, SIGCAS, SIGCSE, SIGPLAN,
Ada-Europe, and the Ada Resource Association
(Cooperation approvals pending)

November 6-10, 2011 — Denver, Colorado



Magnolia Hotel
818 17th Street
Denver, Colorado 80202 (USA)

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

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
jorgen.bundgaard@tdcadsl.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.

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.

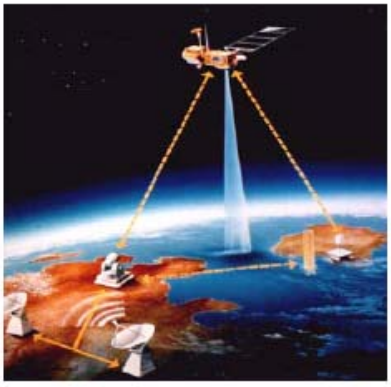
Grant for Reduced Student Fees

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.

Ellidiss Software
TNI Europe Limited

- Ellidiss Software, the trading name of TNI Europe Limited, offers STOOD 5 for full state of the art software design modeling using AADL , HRT-HOOD and UML.
- STOOD 5 is the leading commercial software toolset supporting AADL and is being used to develop software systems for mission critical projects including Airbus, US101, Rafale, Tiger and M346.
- Ellidiss's CP HOOD software toolset is deployed widely on Eurofighter Typhoon, Tornado, Hawk and Nimrod.
- HRT-UML, the latest toolset developed by INTECS of Italy, used in the Space and Automotive industries, is also available from Ellidiss.
- For further information contact :

pam.flood@ellidiss.com
+44 (0)1260 291449



Implementing a Software Product Line for a complex Avionics System in Ada 83

Richard Bridges

Eurocopter Deutschland GmbH, D-81663 München, Germany; Tel:+49 89 607 28891; email:Richard.Bridges@eurocopter.com

Frank Dordowsky

ESG Elektroniksystem- und Logistik-GmbH, Livry-Gargan-Str. 6, D-82256 Fürstenfeldbruck, Germany; Tel+49 8 9216 2871; email: Frank.Dordowsky@esg.de

Dr. Holger Tschöpe

Eurocopter Deutschland GmbH, D-81663 München, Germany Tel:+49 89 607 26713; email: Holger.Tschoepe@eurocopter.com

Abstract

The NH90 is a medium weight multi-role helicopter that has been successfully sold to 14 nations and their armed forces. The growing number of customers and their specific application domains for the NH90 has led to an increasing number of functionally different helicopter variants. In order to cope with the high number of software variants, the NH90 software team has developed a software architecture that is based on Software Product Line (SPL) principles.

Software product lines rely on variability to manage the differences between products. Many variability realisation techniques discussed in the literature rely on mechanisms that are discouraged in the development of safety critical avionics software.

The majority of the avionics software of the NH90 is written in the programming language Ada 83. Its successor Ada 95 offers additional features to implement variation points. However, a transition to Ada 95 was not possible due to a number of project specific restrictions.

The construction of the NH90 SPL relies on three pillars: an embedded real-time framework, extensive code generation and a set of design and coding patterns.

This article discusses the main properties of the software architecture that enabled the introduction of the software product line, the techniques used for code generation, and the use of design and coding patterns. It provides an example of an implementation of a software product line in a real project within the avionics domain under the limitations of a dated technology.

Keywords: Avionics Systems, Software Product Lines, Ada

1 Introduction

1.1 Project background

The NH90 is a medium weight multi-role military helicopter that comes in two basic versions: the Tactical Transport Helicopter (TTH) and the NATO Frigate Helicopter (NFH). It is being produced in more than 20 variants for 14 nations and their armed forces.

The software division at Eurocopter Germany develops the on-board software for three computers of the NH90 avionics system. The growing number of customers and their specific set of mission requirements for the NH90 has led to an increasing number of functionally different helicopter variants. Moreover, during the long development time that is typical for complex military avionics projects, the computing technology has changed considerably over time so that the current operational software has to fit to several processor architectures. In order to cope with the high number of software variants and technology variations, the NH90 software team has developed concepts and strategies for SW architecture and tool modifications based on Software Product Line (SPL) principles [8].

1.2 Nh90 avionics system architecture

The NH90 Avionics System is decomposed into two major subsystems: the CORE System and the MISSION System ([4],[5]). Each major subsystem is controlled and monitored by a redundant set of main computers, the *Core Management Computer* (CMC) and the *Mission Tactical Computer* (MTC) respectively. Major end items of the CORE system are the *Display and Keyboard Units* (DKUs) and the *Multifunctional Displays* (MFDs) that together form the main human-machine interface (HMI) of the NH90 helicopter. The equipment of each subsystem is connected to its main computers either via a dual redundant MIL-STD 1553 bus, via ARINC 429 lines, serial RS-485 lines, or via Ethernet lines. The main computers act as bus controllers for their respective data buses. For both pairs of redundant main computers, one assumes the *system role* of

primary bus controller where the other acts as *backup* bus controller.

This article describes the software architecture of the CMC and the MTC. Both computers share the same hardware and software architecture as described in Sect. 3 below and are known generically as *CMTC*.

1.3 Article outline

Section 2 summarizes the limitations imposed by regulations of the domain and the dated implementation technology on the selection of variability mechanisms available for the NH90 SPL. It also lists the major product line design principles that address these limitations.

The NH90 software team has developed a solution that follows these principles and that relies on three pillars: a software architecture based on an embedded real-time framework, code generation and a set of design and coding patterns. The main subject of this paper is the software architecture that is described in section 3, followed by a short account of the code generation technology (Sect. 4). Section 5 discusses the use and implementation of design patterns under the limitations of Ada 83.

Section 6 concludes the article with a summary.

2 The NH90 software product line

2.1 NH90 SPL goals

The NH90 was originally planned for 4 nations (France, Germany, Italy, Netherlands) for all armed forces (sea, land, and air) so there were already 8 variants from the beginning [18]. Every new customer then increased the total number of variants. Moreover, for every contractual variant there may be more than one technical variant.

The first dramatic increase occurred in 2002 when the number of variants rose from under 10 to nearly 25. The second large increase was in 2005 (from 25 to over 40).

These variants are not only functionally different but also employ different hardware and software technologies as shown in Table 1.

Table 1 NH90 Software Platforms

	Customer	Compiler / Runtime	CPU Type and Speed
I	Germany, Italy, France (Navy), Netherlands, Finland, Norway, Sweden (CMC), Greece	Alsys Ada 83	Dual Motorola 68040
		ARTK	25 MHz / 20 MHz
II	Sweden (MTC), Australia, Oman, New Zealand	GHS Ada 95	Dual Power MPC8245
		GSTART	250 MHz
III	Spain, France (Army), Belgium, Portugal, Germany, Italy, …	Aonix Ada 95	Single PowerPC750Fx
		RAVEN	580 MHz

The initial solution attempt to copy all software and documentation of one variant and to maintain the different variants in separate branches soon became unmanageable and much too costly in terms of man power and development resources. The NH90 software team then decided to apply product line principles to solve the variant problem. The major concept of a software product line is to produce and maintain all current and future helicopter variants out of a single asset base as described in [7]. With this approach the integration of new features or equipment will be simplified and the reduced effort for qualification and maintenance will lead to an overall cost reduction

2.2 NH90 SPL constraints

One advantage of software product lines is that a large body of knowledge has been developed in the past 15 years with many solutions to common problems that arise during implementation of a product line. Unfortunately, a number of these solutions cannot be used in the domain of military avionics due to the specialities of this domain. Some of these are listed in the following:

The RTCA DO-178B ([1]) guides certification authorities on the approval of safety critical airborne software. Three topics of these recommendations are especially important for software product lines: *dead code elimination, option-selectable software* and *deactivated code*. Dead code is code that will never be executed and the DO-178B requires dead code to be completely removed. Deactivated code is only executed in certain configurations. As an example, the NH90 software team produces software versions that will be loaded to several helicopter variants where not all functions are executed in all these variants. The code that implements these functions is option-selectable and therefore represents deactivated code. For approval, it must be verified that deactivated code cannot be executed inadvertently. In order to facilitate this verification, variant decision code must not be distributed over the code base.

Some approaches use tool supported source code modification based on feature selection. Modified code must be re-verified at high cost, or the tool – which would be rated as development tool in DO-178B terms – must be qualified to the same design assurance level and the same standards as the software that it modifies. The authors are not aware of a commercially available SPL tool that is qualifiable in the sense of DO-178B.

Safety critical real-time systems must be *fully deterministic* and *predictable*. They are not allowed to crash due to memory exhaustion, and hard real-time systems must always meet their deadlines. The system implementation must be verifiable in that respect, and analysis during design must assure that resource consumption and timing constraints will always be met.

Object-oriented features such as inheritance and polymorphism are often used to implement variation points ([19],[13]). Certification authorities have raised a number of concerns on the application of object-oriented principles in the past ([2],[11],[3],[12]). The new issue of the DO-178, DO-178C, will include guidance for the application of object-oriented techniques in one of its technical supplements. However, airborne software developers still need to consider if the cost and restrictions for meeting the

objectives of DO-178 as well as the risks of introducing new technology do not outweigh the benefits of object orientation

The development of military systems for several customers must consider secrecy and the fact that system maintenance is often performed by the customers own organisation or industry. A suitable reuse strategy must therefore consider the following principles:

- *Non-disclosure Principle*: A customer requires that deliverables including documentation and source code do not reveal any customer specific information to other customers.
- *Customer Relevance Principle*: A customer requires that deliverables only contain design and implementation details which are relevant to this customer.

There are also some constraints specific to the NH90 project:

- A single software version may be loaded in different helicopter variants. The software must identify the helicopter variant during run-time and then activate the functions of this variant and de-activate all others.
- In the NH90 project, different hardware and software technologies are used: the main computers of some variants are equipped with multiprocessor boards on Motorola MC-68040 basis, others use single boards with PowerPC architectures (see Tab. 1). Some variants use Ada 95, others still Ada 83. Application component design must not be affected by the underlying technology.
- *A large code base written in Ada 83* was already available and a migration to later versions of Ada was not possible due to schedule and budget constraints. Any architectural solution to the variability problem must therefore provide downward compatibility to the large Ada 83 code base.

2.3 NH90 SPL constraints

The development of the NH90 SPL was guided by the following major principles:

1 Feature based selection of pre-developed components with subsequent integration into the final variant specific product instantiation. This process should only need as little manual adaptation as possible – the ultimate goal is to have no manual adaptation at all.

2 Components must be reusable across variants without modification — this applies not only to code but also to all other artefacts such as documentation, models, test scripts and data etc.

3 The assets must not contain any variant specific information and they shall not make any assumptions about the helicopter variant for which they may have been selected. There must be no variant decision logic within the components, and component code must not be aware of the actual variant it is running in.

4 Complex and distributed adaptations shall be delegated to code generation.

Some practitioners recommend to use language pre-processors (e.g. `#ifdef` in C) to implement variability. However, pre-processors are not available in standard Ada, and the NH90 SPL had to rely on a truly Ada conformant solution.

3 Software architecture

Both CMC and MTC share the same hardware and the same software architecture. The architecture of both computers is shaped by the *embedded real-time framework NSS* that is described in the following subsection. The actual version of the operational software on either computer is instantiated with a mechanism called *adaptation* that is described in 3.4 below.

3.1 Embedded real-time framework NSS

The software framework NSS relieves the application programmer of intricate real-time programming tasks such as real-time scheduling, device and I/O handling, data conversion between Ada data structures and raw I/O data, error and exception handling, redundancy management etc. The real-time framework is organised as a layered system ([6],[10]) as indicated in Figure 1.

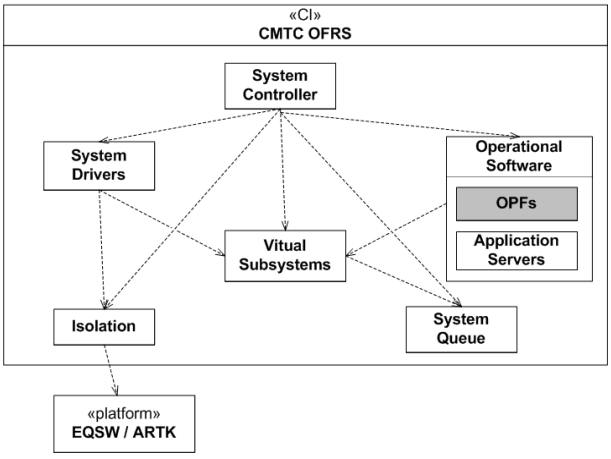


Figure 1 Software Architecture Overview

The main components of the architecture are the following:

The System Controller forms the uppermost layer and controls the complete operational software. It is responsible for the control, schedule and dispatch of Ada tasks as well as general management functions such as start-up, shutdown and redundancy management for example. A very important subcomponent in this context is the *button database* which is described below. The system controller determines the helicopter variant by reading a designated non-volatile memory location that contains the variant id string. It is also responsible for identifying the actual system role and mode and initiates the transition into that role and mode respectively.

The System Drivers drive the physical exchange of raw IO data. There are drivers for Milbus, ARINC, Ethernet, serial and discrete lines. They store data received from external equipment in the *Virtual Subsystems* (see below), or retrieve data from the Virtual Subsystems and send it to the external equipment.

The Application Servers make use of the system drivers and provide a higher level of abstraction on data exchange. Important examples of Application Servers are the *DKU Server* which handles the communication to the DKUs, or the *NVM Server* which manages access to the non-volatile memory (NVM) of the main computers.

The On-board Processing Functions (OPFs) implement the proper application specific processing requirements. An OPF is a set of operations which accept data or control input, perform computational or control functions and produce data or control output. The OPFs form the application components. They provide an interface to the system controller that is called the *button layer*. This naming shall reflect that the realtime framework does not know the internals of the operations and simply regards them as buttons to be pushed. The button layer consists of standardised Ada procedures that are called by the system controller. The information as to which buttons must be called in which system mode at what frequency is stored in the *button database* that is part of the system controller, as mentioned above. The button database plays an important role in the implementation of the NH90 SPL and is described in more detail below.

The Virtual Subsystems are a common real-time data store which provides a uniform data abstraction of the avionic subsystems to the operational processing functions, and isolates these from I/O interface specific data representations. They also organise the data exchange between application components and between application components and the NSS. With the virtual subsystems, the architecture adopts a style known as repository style ([10]) or blackboard style ([6]). The virtual subsystems are divided into *virtual devices* - a virtual device is an Ada package representation of an equipment or equipment capability. As such it forms one of the basic building blocks of the NH90 SPL.

The Isolation layer provides an abstract interface to the equipment software (EQSW) and the Ada Runtime Kernel (ARTK). It isolates the operational software from system dependent features and enables the operational software to run on target and host platforms as well as the different hardware architectures with only minor modifications.

The EQSW/ARTK layer contains the software provided by equipment manufacturers of the computer hardware and the compiler vendors.

The framework and the application components exhibit the sandwich structure that is typical for frameworks (e.g. [14]): application components call NSS services but are themselves called by the NSS. Therefore, application components must be visible to the NSS. However, different software variants require differing sets of application components and therefore lead to changing dependencies and dispatch routines within the NSS. In object-oriented systems, or at least if function pointers would be available, this could be implemented by a registration pattern (e.g. the publisher-subscriber pattern in [6]). Since these mechanisms are not available for Ada 83, the interface between NSS and the application servers, drivers and the OPFs is generated.

It is important to understand that all concurrency and therefore the complete real-time tasking is provided by the framework alone; the application components as well as server and driver code is strictly sequential. A major distinction of the operations that the system controller activates is between cyclic or acyclic activation. The variant specifics of the dispatch of cyclic and acyclic operations is slightly different and described in the following subsections.

3.2 Cyclic processing

The periods of the cyclic operations are completely determined by the MILBUS message frequencies that range from 50 Hz down to 1.5625 Hz. There is one uniquely identified periodic Ada task per frequency and per processor (the standard main computer has two processing boards), and every cyclic operation is allocated to a unique periodic task depending on its frequency and the processing board it has to run on.

The set of application functions that must be executed depends on the *system mode* (operational, maintenance) and the *system role* (master, slave). Moreover, since a single software supports several helicopter variants, different sets of application operations are executed for the different helicopter variants. The system controller must therefore determine which set of operations to execute in the actual combination of helicopter variant, mode and role. To do this, the system controller must evaluate a runtime data structure that is called *periodic button database*. The periodic button database is part of the button database that was briefly introduced in section 3. It stores for each operation the associated periodic task as well as the roles and modes and the helicopter variants in which the operation is required to be executed. The code section below shows the generated Ada construct for the periodic button database

```
-- <<Generated>>
Pdb : constant Periodic_Button_Db := (
. . .
16 => (On_Helicopter =>
       (A => True, others => False),
     In_Role =>
      (Master => True, Slave => True),
     In_Mode =>
      (Operational => True,
       Maintenance => True),
     Task_Name => P1,
     Button_Name => Capability_A_Process_P1),
. . . ) ;
```

Since it is not possible with Ada 83 to store a reference to a subroutine in a data structure, every operation must be uniquely identified by an operation identifier (*Button Name*) that is an enumeration literal of a generated enumeration type. The Button Name is derived automatically from the fully qualified operations' name. It is obvious that such a design would not be maintainable without code generation.

A lookup of every operation in every processing cycle would produce too much overhead so the team has developed a prefetch solution, called *configuration*: during start-up and during role or mode changes, the system controller scans the periodic button database and selects all operations that are permitted in the actual combination of helicopter variant, role and mode, and establishes a list of the permitted operations (*activation list*) for every periodic task (Figure 2). The function Is_In_Set accesses the periodic database to check if a button (identified by an index) is permitted for a given role, mode and helicopter variant:

```
function Is_In_Set (
  Index : in Natural_16;
  On_Helicopter : in Helicopter_Variant ;
  In_Role : in Computer_Role;
  In_Mode : in Computer_Mode) return Boolean is
begin
  return Pdb (Index).On_Helicopter (On_Helicopter)
       and then
     Pdb (Index).In_Role (In_Role) and then
     Pdb (Index).In_Mode (In_Mode);
end Is_In_Set;
```

During cyclic execution for each periodic task, the task simply dispatches every operation that is on its activation

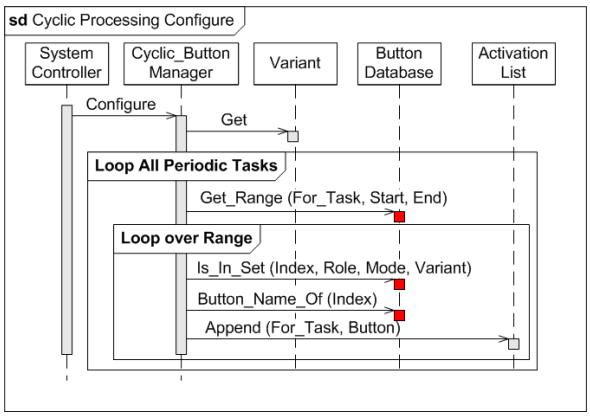


Figure 2 Cyclic Processing Configuration

list (Figure 3). This list contains the operation IDs instead of the operations, and the actual dispatcher is a large generated case statement on the Button Name. The button database is not consulted during cyclic execution:

```
procedure Execute (T : in Task_Name) is
begin
  if Activation_List (T).Size /= Null_Length then
    for I in 1 .. Activation_List (T).Size loop
      Dispatch (Activation_List (T).Item ( I ));
    end loop;
  end if;
end Execute;
-- <<Generated>>
procedure Dispatch (Name : in Button_Name) is
begin
  case Name is
. . .
    when Capability_All_Process_P1 =>
      Capability_All .Process_P1;
    when Capability_A_Process_P1 =>
      Capability_A.Process_P1;
    when Capability_B_Process_P1 =>
      Capability_B.Process_P1;
. . .
    end case;
end Dispatch;
```

Within every periodic task, the operations are sequential and it is therefore possible to specify an execution order of the operations. This execution order is maintained during the runtime as long as there is no other role or mode switch.

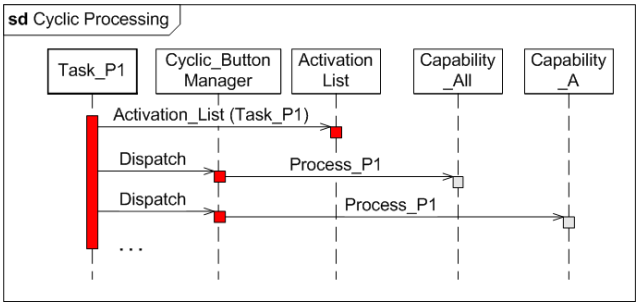


Figure 3 Cyclic Processing

The result is a set of periodic tasks with harmonic periods that are scheduled non-deterministically which is in line with the Ada tasking model. We apply rate monotonic scheduling theory ([15]) to ensure and verify the schedulability of the tasks.

3.3 Event handling

The NSS implements an event based implicit invocation architectural style ([10]) to handle acyclic processing requests. Events are the only means to establish asynchronous communication between components, including OPF and NSS components. Events are used to process asynchronous MILBUS messages for example, but the main use of events is to process crew commands from the DKU.

A component or one of its operations can raise an event by putting data into a data structure called *Traffic Light Data (TLD)* that is located in the virtual subsystems. The TLD enqueues an event associated with the data to the *system queue* (Figure 4). The system queue works across processor boundaries. Operations or components that raise an event are called *producers* of the event.

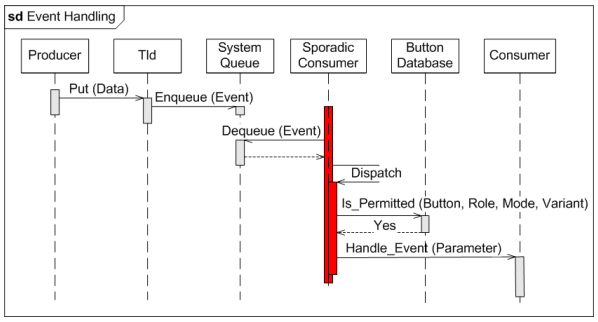


Figure 4 - NSS Event Handling

Operations or components that react on an event are called *consumers* of the event. There may be more than one consumer per event.

On every CPU there is a special task in the system controller, the *Sporadic Consumer Task SPC*, that dequeues the events from the system queue. The Sporadic Consumer Task dispatches the consumer operations for the event with a large, generated case statement on the event id. However, before the consumer operation is executed, the dispatcher checks if the operation is permitted for the current combination of helicopter variant, role and mode by interrogating the *event button database*:

```
procedure Execute is
begin
  System_Queue.Get (New_Message);
  if New_Message.Event /= Null_Event then
   Dispatch (New_Message);
  end if ;
end Execute;
-- <<Generated>>
procedure Dispatch (E : in Event_Message) is
begin
  case E.Event is
. . .
   when Capability_A_Switch_On =>
    if Is_Permitted (Capability_A_Perform, Variant,
      Role, Mode) then
     Capability_A.Perform(Switch_On);
    end if ;
   when Capability_A_Switch_Off =>
    if Is_Permitted (Capability_A_Perform, Variant,
      Role, Mode) then
     Capability_A.Perform(Switch_Off);
    end if ;
. . .
end case;
```

The event button database is similar in structure as the periodic button database and is also part of the button database that has been introduced in section 3.1. The consumer operations are also identified by generated operation identifiers.

The name of the Sporadic Consumer has been derived from its implementation as a sporadic server algorithm ([17],[16]). This algorithm optimises response times for events with irregular arrival times but fits well in the rate monotonic scheduling theory so that the deadlines for the periodic tasks can still be maintained.

3.4 Adaptation

The real-time framework is composed of two basic segments: an NSS kernel segment, comprising a set of components which are common to all NSS, CMC and MTC variants and software versions, and an adaptable computer specific segment, comprising a set of components all of which need some kind of adaptation to the specific needs of the computer. The framework has been designed so that all components requiring adaptation are separate Ada units. This concentration of adaptable code into a few separate units increases reliability, maintainability, testability of the adaptation. Moreover, most modules requiring adaptation are generated.

The non-generated units that must be manually adapted are kept in separate variant specific file branches. There is an ongoing effort to reduce the number of these non-generated adapted files by either generating them or by removing them with a more adaptable design. The mapping registry pattern (section 5 below) is an example of such a design improvement

4 Software generators

More than 50 percent of the adaptation source code of the main computers is generated from data maintained in an ORACLE database called ODIN. ODIN is an acronym for *OFRS Data and Interfaces in NH90*. ODIN imports interface definition data from other engineering databases (Figure 5). Software engineers then add software related information (see [5] and [8] for a more complete overview over the NH90 tool chain). The content of ODIN is extracted and fed into a set of Perl scripts that produce the generated code. The tool chain has been qualified so that the generated source code does not need the verification required for the manually written code.

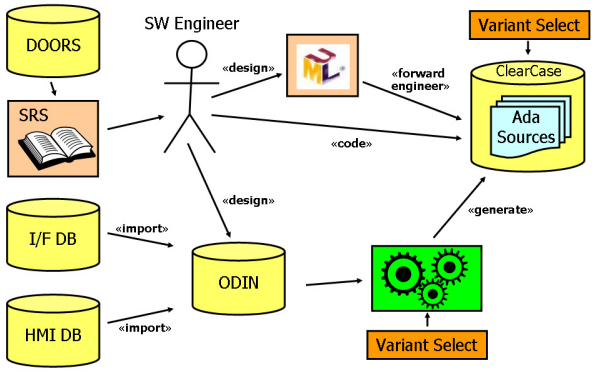


Figure 5 - NH90 Tool Chain

There are three main areas in the ODIN data model:

- *Real-time Model*: All definitions necessary to generate the Ada tasking model and the allocation of Ada tasks to processors.

- *Virtual Subsystems*: All data required for the generation of the virtual subsystems. This is also the area that stores the imported interface data.
- *Buttons (Operations):* This area contains all information associated with the operations in order to generate the button databases, including the fully qualified Ada name and the frequency of the operation as well as its allocation to a processor board.

The most important requirement imposed by NH90 SPL principles is to maintain a single data repository for all variants in order to avoid duplicated effort for maintaining several copies of the same data. We call such a comprehensive repository a *superset*. The data definitions in this superset also form assets in the sense of [7].

In order to fulfil the specific constraints of military avionics as listed in section 2, the extraction and code generation must only produce software items that are applicable to the software version under construction. This requires that the data elements must be filtered on extraction. In ODIN, the extraction filter uses the *feature* as filter criterion. Buttons and devices are linked to features as their implementations. A helicopter variant is defined, in ODIN, as a set of features. A set of helicopter variants can be combined into a *variant group*.

The *extraction script* takes a variant group as input and selects all elements from the ODIN superset that are associated with at least one feature of at least one variant in the variant group. The extraction script produces an intermediate output in the form of formatted ASCII files. The *generator scripts* pick up these intermediate files and produce the Ada sources. They use a set of templates that form the skeletons of the generated Ada packages. The generator scripts are implemented in Perl.

The generator scripts do not contain any variant decision logic – the filter rules are concentrated in the extract scripts in order to reduce the complexity of the generator scripts and thus to facilitate the qualification of the tool set.

5 Design patterns

The application components must be designed in a certain way to meet the modularisation principles of the NH90 SPL. In fact, a product line is much more sensitive to the quality of the design than a system not intended for re-use. Developers must follow clean design practices because sloppiness in that respect can break the re-usability of a component which in turn reduces the benefits of the product line approach. In order to assist the developers in achieving the design goals as well as enforcing compliance to the software architecture, the NH90 team has developed a set of design patterns that provide standard solutions to recurring design problems, especially for variant handling.

The pattern names have been selected to resemble well-known patterns established in the literature (e.g. [9]), based on the intent of the pattern. The implementation of course differs due to the unavailability of object-oriented features – instead, it relies on sophisticated use of the mechanisms of the software architecture and on code generation.

One example of an NH90 specific design pattern is the *mapping registry* pattern. This pattern addresses the problem that sometimes cross-sectional type mappings between different discrete types are needed that span multiple components but also depend on the helicopter variant.

The mapping tables cannot be located within a single component, and they cannot be generated because it is not possible to extend ODIN and the generator suite whenever a new mapping is introduced. Manually maintaining a mapping table and its access methods is also not a viable solution because the mapping table would need separate source code branches for every software version and would therefore create a major maintenance problem.

The mapping registry pattern resolves these problems by providing a central variable mapping table where components can register the type mappings they need. The implementation of the pattern relies on another architectural pattern: every component must provide an initialization procedure that is executed by the system controller before entering the operational mode.

The central variable mapping table is encapsulated by a *registry object*. The registry provides a *registration operation* to set up the actual mappings in the registry. These registry operations can be called by the initialization procedure of the component in order to provide the type mappings that are relevant to it (Figure 6).

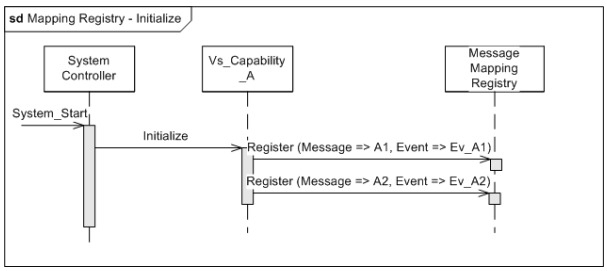


Figure 6 - Registry Mapping Initialization

The registry also provides a map function that can be used by any client component to look up the mapped element.

With the mapping registry pattern it is possible to dynamically adjust the type mapping tables to the types actually used in the helicopter variant without manual recoding and without creating numerous variant specific branches of the source files that contains the mapping. The implementation of the pattern only uses Ada 83 capabilities and the features of the NH90 software architecture.

6 Summary

The NH90 SPL is an example of a software product line in a real large project within the complex domain of military avionics. Product lines in this domain are subject to a number of restrictions that limit the choices of implementation concepts. The most important restrictions for the NH90 SPL are the strict regulations in the domain

and the limited support by a dated technology implicated by the mandated use of the existing design and code base.

The NH90 software team has developed a solution that relies on three pillars: a software architecture well suited for product lines, intensive use of code generation, and the provision of design and coding patterns to help developers to meet the design constraints imposed by the SPL. Any of these three building blocks is indispensable.

With the software product line approach the NH90 software team is now in the position to efficiently manage the helicopter variants: the effort for updating a product instance is negligent by now, and the effort for the creation of a new product instance has been reduced to a quarter of the original effort without the SPL. As a positive side effect we witness an increase of the overall quality, transparency and maintainability of code and design.

References

[1] RTCA DO-178B Software Considerations in Airborne Systems and Equipment Certification, December 1992.

[2] Position paper CAST-4: Object-oriented technology (OOT) in civil aviation projects: Certification concerns, January 2000.

[3] EASA certification memo: Use of object oriented techniques at design or source code level, May 2008. Issue 1 Rev. 2.

[4] Richard Bridges. NH90 helicopter avionics systems from the 1990s to 2010 and beyond. In *Workshop Software-Architekturen für Onboardsysteme in der Luft- und Raumfahrt*. Fachausschuss T6.4 Software Engineering, Deutsche Gesellschaft fuer Luft- und Raumfahrt, Oct 2007.

[5] Gerd Budich. Generation of Ada code from specifications for NH90 computers. In *Proceedings of the 26th European Rotorcraft Forum*, 26–29 September 2000.

[6] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[7] Paul Clements and Linda Northrop. *Software Product Lines - Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2002.

[8] Frank Dordowsky and Walter Hipp. Adopting software product line principles to manage software variants in a complex avionics system. In John D. McGregor and Dirk Muthig, editors, *Proceedings of the 13th International Software Product Line Conference, San Francisco, California, USA 2009*, volume 1. Software Engineering Institute, August 2009.

[9] E. Gamma, R. Helms, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] David Garlan and Mary Shaw. An introduction to software architecture. Technical Report CMU-CS-94-166, Carnegie Mellon University, 1994.

[11] K.J. Hayhurst and C.M. Holloway. Considering object oriented technology in aviation applications. In *Digital Avionics Systems Conference, 2003. DASC '03. The 22nd*, volume 1, pages 3.B.1–3.1–8 vol.1, Oct. 2003.

[12] Javier Miranda. Towards certification of object-oriented code with the gnat compiler. *Ada User Journal*, 28(3):178–183, September 2007.

[13] Thomas Patzke and Dirk Muthig. Product line implementation technologies. Programming language view. Technical Report IESE Report No. 057.02/E, IESE Institut Experimentelles Software Engineering, October 2002.

[14] Wolfgang Pree. Meta patterns — A means for capturing the essentials of reusable object-oriented design. *Lecture Notes in Computer Science*, 821:150ff, 1994.

[15] Lui Sha and John B. Goodenough. Real-time scheduling theory and Ada. Technical Report CMU/SEI-89-TR-014, ESD-TR-89-022, Software Engineering Institute, April 1989.

[16] Brinkley Sprunt and Lui Sha. Implementing sporadic servers in Ada. Technical Report CMU/SEI-90-TR-6, ESD-90-TR-207, SEI Software Engineering Institute, May 1990.

[17] Brinkley Sprunt, Lui Sha, and John Lehoczky. Scheduling sporadic and aperiodic events in a hard real-time system. Technical Report CMU/SEI-89-TR-11, ESD-TR-89-19, SEI Software Engineering Institute, April 1989.

[18] Stefan Steyer. NH90 Beschaffungsdrama. Die Entwicklung des NH90. *Rotorblatt*, 16(3):52–55, 2009.

[19] Mikael Svahnberg, Jilles Van Gurp, Jilles Van, Gurp, and Jan Bosch. A taxonomy of variability realization techniques. *Software—Practice and Experience*, 35:705–754, 2001.

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/category/developers-center/gems/.

Gem #89: Code Archetypes for Real-Time Programming – Part 1

Marco Panunzio, University of Padua

Date: 22 June 2010

Abstract: In the programming of real-time systems, code that deals with concurrency and real time often draws from explicit or implicit recurring patterns. Therefore it is best factored out. In this series of Ada Gems, we illustrate a set of code pattern archetypes that are intended to ease the development of real-time systems.

Introdution

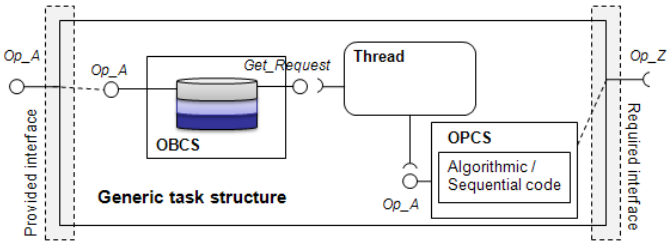
In this series of Ada Gems we propose a set of code archetypes for the development of real-time systems. The code archetypes comply with the restrictions of the Ravenscar Profile, a subset of the Ada language specifically suited for the development of high-integrity real-time systems. The Ravenscar Profile was devised to guarantee that programs written in accordance with it are amenable to static analysis in the time dimension. In fact, the profile excludes all Ada constructs that are exposed to nondeterministic or unbounded execution time. In the space dimension, the profile prohibits the use of constructs that implicitly perform dynamic memory allocation.

As an additional benefit, Ravenscar systems can be implemented on top of real-time kernels that can be very small in size and fast in time, which are attractive characteristics for applications that can afford little overhead and must undergo extensive qualification/certification.

The essential elements of the Ravenscar restrictions are: (i) *static existence model*. The system is composed of a fixed set of tasks and protected objects, defined at library level and with a statically assigned priority or ceiling priority. No task terminates, and abort statements are disallowed. (ii) *communication model*. Tasks are only allowed to communicate asynchronously, via protected objects. Task rendezvous is therefore disallowed. (iii) *deterministic execution model*. The profile excludes all constructs that introduce nondeterminism: relative delays, as they introduce nondeterminism in the suspension time of tasks; requeue statements; the use of the Ada.Calendar package, as all time-related operations rely on the high-precision Ada.Real_Time package; protected objects are restricted to having at most one entry on which only a single task can enqueue; guards of entries are composed of a simple Boolean condition to avoid side effects and nondeterminism in the evaluation time.

In this series of Ada Gems we illustrate a set of code archetypes that realize common programming patterns suited for the development of Ravenscar-compliant real-time systems. The use of the archetypes permits factorization and thus helps reduce the size of the code that implements the concurrent elements of the system.

An important additional goal of these archetypes is to achieve complete separation between the algorithmic/sequential code of the system and the code that manages concurrency and real-time aspects. This separation of concerns permits developing the algorithmic contents of the system (that is, the behavior of the system) independently of the management of tasking and real-time issues.



This goal is achieved by encapsulating the sequential code in a suitable task structure. The figure above depicts the generic structure of our task archetypes.

The sequential code is enclosed in a structure that we term the **Operational Control Structure** (OPCS). The code is executed by a **Thread**, which represents a distinct flow of control of the system. The task structure may be optionally equipped with an **Object Control Structure** (OBCS). The OBCS represents a synchronization agent for the task: as we shall see, we use it mainly for *sporadic tasks*. The OBCS consists of a protected object that stores incoming requests for services to be executed by the Thread. As multiple clients may independently require services to be executed by that Thread, the operations that post execution requests in the OBCS are protected. Upon each release, the Thread fetches one of those requests (FIFO ordering is the default) and then executes the sequential code, stored in the OPCS, which corresponds with the request.

As we will illustrate in the third Gem of this series, the operations provided by the OBCS, which form the *provided interface* of the overall entity, match the signature of the sequential operations of the OPCS (Op_A in the figure above). Thanks to that, the callers need not be aware that they are in fact only posting execution requests in the OBCS, while the actual execution will be performed by the Thread.

The sequential code embedded in the OPCS may need to invoke services from other software entities (the operation Op_Z in the figure above). In the fifth Gem of this series we will describe how those functional needs can be fulfilled.

As a conclusion, our entities encapsulate their internal structure and expose to the external world just an interface that matches the signature of the operations embedded in the OPCS. The different concerns dealt with by each such entity are separately allocated to its internal constituents: the sequential behaviour is handled by the OPCS; tasking and execution concerns by the *Thread*; interaction with concurrent clients and handling of execution requests are handled by the OBCS.

Structure of this Ada Gems Miniseries

1. Introduction and Cyclic Task
2. Simple Sporadic Task
3. Sporadic Task – System Types and Task Types
4. Sporadic Task – Sequential Code and OBCS
5. Intertask Communication

Acknowledgments The task structure we adopt is an evolution of the HRT-HOOD design methodology [1], from which we also inherit the terms OBCS and OPCS.

Early work on code generation from HRT-HOOD to Ada was described in [2] and [3].

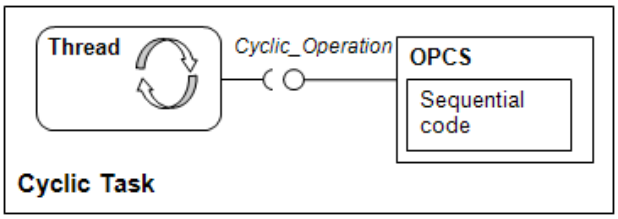
The code archetypes that we describe in this Ada Gems miniseries were used for the code generation in the HRT-UML track of the EU-funded ASSERT project [4]. In that project, Matteo Bordin, then at the University of Padua, was the main designer of the code generation strategy and code archetypes.

Finally we would like to thank Tullio Vardanega for his preliminary review of the contents of this miniseries, and Matteo Bordin, now with AdaCore, for his extensive review of the contents and his useful suggestions.

Let's get started…

Cyclic Task

In this section we illustrate our code archetype for cyclic tasks. It allows the developer to create a cyclic task by instantiating a generic package, passing the operation that needs to be executed periodically.



The archetype is quite simple. In fact, we only need to create a task type that cyclically executes a given operation with a fixed period. The specification element of the archetype is:

```
with System; use System;
generic
  with procedure Cyclic_Operation;
package Cyclic_Task is
  task type Thread_T
   (Thread_Priority : Priority;
    Period : Positive) is
   pragma Priority (Thread_Priority);
  end Thread_T;
end Cyclic_Task;
```

The specification above defines the *task type* for the cyclic thread. Each thread is instantiated with a statically assigned priority and a period which stays fixed throughout the whole lifetime of the thread. The *task type* is created inside a generic package, which is used to factorize the code archetype and make it generic on the cyclic operation. The Ada body is specified as follows:

```
with Ada.Real_Time;
with System_Time;

package body Cyclic_Task is
 task body Thread_T is
  use Ada.Real_Time;
  Next_Time : Time := System_Time.System_Start_Time;
 begin
  loop
    delay until Next_Time;
    Cyclic_Operation;
    Next_Time := Next_Time + Milliseconds (Period);
  end loop;
 end Thread_T;
end Cyclic_Task;
```

The body of the task consists of an infinite loop. Just after activation, the task enters the loop and is immediately suspended until a system-wide start time (*System_Start_Time*). This initial suspension is used to synchronize all the tasks that are to execute in phase and let them have their first release at the same absolute time. When resuming from the suspension (which notionally coincides with the release of the task), the task contends for the processor and executes the *Cyclic_Operation* specified in the instantiation of its generic package. Then it calculates the next time it needs to be released (*Next_Time*) and as first instruction of the subsequent loop, it issues a request for absolute suspension until the next multiple of its period.

The code archetype is simple to understand, yet a few comments are in order.

Firstly, we must stress that the use of *absolute time* and thus of the construct **delay until** (as opposed to *relative time* and the construct **delay**) is essential to prevent the actual time of the periodic release from drifting.

Secondly, the reader should note that the *Cyclic_Operation* is parameterless. That is not much of a surprise, as it is consistent with the very nature of cyclic operations which are not requested explicitly by any software client.

Finally, this version of the cyclic task assumes that all tasks are initially released at the same time (*System_Start_Time*). Support for a task-specific offset (phase) is easy to implement: we just need to specify an additional *Offset* parameter on task instantiation, which is then added to *System_Start_Time* to determine the time of the first release of the task. The periodic release of the task will then assume the desired phase with respect to the synchronized release of the tasks with no offset. In the next Ada Gem we will illustrate a simple code archetype to realize sporadic tasks.

References

[1] Alan Burns and Andy J. Wellings: "HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems". Elsevier Science, 1995. ISBN 978-0444821645.

[2] Juan Antonio de la Puente, Alejandro Alonso, and Angel Alvarez: "Mapping HRT-HOOD Designs to Ada 95 Hierarchical Libraries." Reliable Software Technologies – Ada-Europe, Springer Volume LNCS 1088, 1996.

[3] Matteo Bordin and Tullio Vardanega: "Automated Model-Based Generation of Ravenscar-Compliant Source Code". Proceedings of the 17th Euromicro Conference on Real-Time Systems, IEEE Computer Society, 2005. ISBN 0-7695-2400-1.

[4] ASSERT project (Automated proof-based System and Software Engineering for Real-Time Systems) http://www.assert-project.net

Gem #92: Code Archetypes for Real-Time Programming – Part 2

Marco Panunzio, University of Padua

Date: 11 October 2010

Abstract: In the previous Gem in this series we introduced the key concepts underlying our code archetypes and described the simplest of our archetypes, which realizes a cyclic task. In this Gem we show how to realize an equally simple sporadic task. In the next Gem in this series, we will depart from this level of simplicity to realize a complete archetype that overcomes some of the limitations intrinsic in these initial solutions.

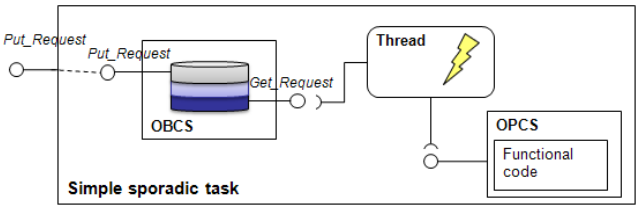
Let's get started…

Simple sporadic task

A sporadic task is a task such that any two subsequent activations of it are always separated by no less than a minimum guaranteed time span. This minimum separation is typically called *minimum inter-arrival time* (MIAT). In this initial archetype, the task executes a single operation at each activation, and it does so in response to a request issued by an external client.

Much like the cyclic task, our sporadic task is composed of: (i) a protected object, shared with the outside world, that external clients invoke to post their requests for execution. In the previous Gem we termed this resource an OBCS; and (ii) a thread of control that waits for incoming requests, fetches the first of them from the protected object and executes the sporadic operation that corresponds to a specific operation as provided by the OPCS.

The task structure whose code we are about to show is sketched in the figure below.



Fortunately, the Ada language is well equipped to realize that structure, and we implement that in our archetype using a protected object under the *Ceiling_Locking* policy for (i) and a task type for (ii).

For the moment, we illustrate a base version of the archetype for a sporadic task. In the explanation we also illustrate some of its limitations that will not be present in a more complex version of the archetype.

The specification for the simple sporadic task follows:

```
with System;
generic
  with procedure Sporadic_Operation;
  Ceiling : System.Priority;
  OBCS_Size : Integer;
package Simple_Sporadic_Task is
  procedure Put_Request;
  task type Thread_T
   (Thread_Priority : System.Priority; Interval : Integer)
   is
    pragma Priority (Thread_Priority);
  end Thread_T;
end Simple_Sporadic_Task;
```

The specification defines (as for the Cyclic task that we presented in the previous Gem) a task type inside a generic package. When instantiating the package we specify the sporadic operation for the task, the Ceiling Priority for the OBCS protected object, and the size of the queue of requests of the OBCS.

Additionally, we create the procedure *Put_Request*, that is used by clients to post a request to the sporadic task.

The body for the package is instead:

```
with System_Time;
with System_Types;
with Ada.Real_Time;

package body Simple_Sporadic_Task is
  Protocol : System_Types.Simple_Sporadic_OBCS
        (Ceiling, OBCS_Size);

  procedure Put_Request is
  begin
   Protocol.Put_Request;
  end Put_Request;

  task body Thread_T is

   use Ada.Real_Time;
   Next_Time: Time:= System_Time.System_Start_Time +
              System_Time.Task_Activation_Delay;
   MIAT : Time_Span := Milliseconds (Interval);
   Release : Time;

  begin
   loop
    delay until Next_Time;
    Protocol.Get_Request (Release);
    Next_Time := Release + MIAT;
    Sporadic_Operation;
   end loop;
  end Thread_T;

end Simple_Sporadic_Task;
```

Comparing this body to the body of the cyclic task, two major differences appear: (i) the presence of an OBCS; and (ii) a slightly modified loop structure.

As in the cyclic task, the sporadic task enters its infinite loop and suspends itself until the system-wide start time. After that: (i) it calls the *entry* Get_Request(Time) of the OBCS; (ii) after the execution of the entry (from which, as we show later on, it obtains a timestamp of when release actually occurred), the task executes the *Sporadic_Operation* (single, for now) specified at the instantiation of its generic package; (iii) it calculates the next earliest time of release (*Next_Time*) so as to respect the minimum separation between subsequent activations. Therefore, on the next iteration of the loop the task issues a request for absolute suspension until that time, and thus it won't probe the OBCS for execution requests until the required minimum separation has elapsed.

As a final note, when the procedure *Put_Request* is called, it just performs a simple indirection to an OBCS procedure with the same name. To appreciate that, we must take a look at the OBCS, which acts as the synchronization agent for the task.

The specification of the OBCS is as follows:

```
with System;
with Ada.Real_Time; use Ada.Real_Time;

package System_Types is
  protected type Simple_Sporadic_OBCS (
      C : System.Priority; Size : Integer) is
    pragma Priority(C);
    procedure Put_Request;
    entry Get_Request (Release_Time : out Time);
  private
    Max_Pending : Integer := Size;
    START_Pending : Integer := 0;
    Barrier : Boolean := False;
  end Simple_Sporadic_OBCS;
end System_Types;
```

The OBCS declares a procedure *Put_Request* that is used to post requests in its queue, and a guarded entry *Get_Request(Time)* that is used by the thread to fetch the requests. In the private part of the declaration, the *Max_Pending* attribute is used to set the maximum number of pending requests that the OBCS can hold (obviously no greater than its size); the *START_Pending* attribute indicates the actual number of pending requests; finally the Boolean *Barrier* is used to control the guard of *Get_Request(Time)*.

```
package body System_Types is
  protected body Simple_Sporadic_OBCS is

    procedure Update_Barrier is
    begin
      Barrier := Start_Pending > 0;
    end Update_Barrier;

    procedure Put_Request is
    begin
      if Start_Pending < Max_Pending then
       Start_Pending := Start_Pending + 1;
      end if;
      Update_Barrier;
    end Put_Request;

    entry Get_Request (Release_Time : out Time)
        when Barrier is
    begin
      Release_Time := Ada.Real_Time.Clock;
      Start_Pending := Start_Pending - 1;
      Update_Barrier;
    end Get_Request;

  end Simple_Sporadic_OBCS;
end System_Types;
```

The body of the OBCS is quite easy to understand. When the procedure *Put_Request* is called, the number of pending requests (*START_Pending*) is increased unless the maximum number has already been reached. In that case the new request is just silently ignored.

The entry *Get_Request(Time)* is used by the task to probe the OBCS for pending requests. In the case where there are requests, the *Barrier* guard is open and the task: (i) saves the time stamp of the execution of the entry (which notionally coincides with the release of the task), that is later used to calculate the next release time; and (ii) decreases the number of pending requests.

At the end of *Put_Request* and *Get_Request*, the value of the *Barrier* guard is refreshed using *Update_Barrier*. In the event that there are no more pending requests, *Barrier* is set to false. For this reason, if the guard is closed when the task calls the entry, the call is blocked until a new request is posted.

The check for the request queue to be not empty is not directly used as the guard expression for the entry, so as to comply with the restriction of the Ravenscar Profile that requires guards to be simple Boolean conditions, and thus have deterministic evaluation. The OBCS has a single entry, as the profile requires, and the only task that can be enqueued on it is the task to which the OBCS belongs, thus ensuring full compliance with the Ravenscar Profile.

While the proposed structure achieves our goal of creating a sporadic task, we immediately notice two potential drawbacks: the *Sporadic_Operation* is parameterless, and the synchronization protocol is very, perhaps too, simple to capture real-life system needs.

For what concerns the first issue, clients of the sporadic task simply trigger new releases of the task, but cannot, for example, pass data to the task as parameters of the release request. Creating a nontrivial producer-consumer collaboration pattern with this task structure is impossible because the consumer task (our sporadic task) cannot receive any data to process.

For what concerns instead the OBCS, in this version it is a simple counter of pending requests.

In the next Gems in this series, we will illustrate how to support sporadic operations with parameters and start to realize more complex queuing policies for execution requests.

Gem #94: Code Archetypes for Real-Time Programming – Part 3

Marco Panunzio, University of Padua

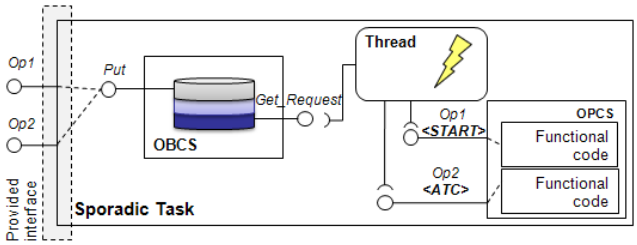
Date: 8 November 2010

Abstract: In the previous Ada Gem we described a code archetype for a simple sporadic task. Nevertheless, we recognized that the archetype is not completely satisfactory for at least two reasons: (i) it is not possible to pass parameters to the sporadic operation; (ii) the synchronization agent (OBCS) is a simple counter of pending requests. In this Ada Gem we illustrate a more complex archetype that supports the invocation of a sporadic operation with parameters. Additionally, we want to explore how it is possible to enrich the OBCS to support complex queueing policies for the incoming requests.

Let's get started…

Sporadic task

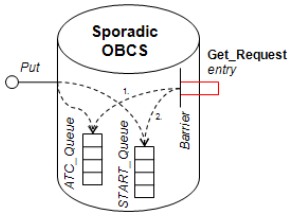
The archetype of a sporadic task that we wish to illustrate is depicted in the figure below.



Suppose that we want to create a sporadic task that at each release can execute either operation `Op1` or operation `Op2`, according to incoming requests by clients. Executing different operations with the same task is not an unusual need in real-time systems, especially when the execution platform has scarce computational power and memory resources, and the excessive proliferation of tasks may tax the system too much.

Furthermore, in this archetype we may also want to establish some relative ordering of importance between `Op1` and `Op2`. We consider `Op1` as the nominal operation of the sporadic task (the operation normally called by clients), and we call it the START operation; in contrast, we consider `Op2` to be a modifier operation, called the ATC. Then, we stipulate that pending requests for execution of `Op1` are served by the sporadic task in FIFO ordering, but requests for `Op2` take precedence over pending requests of `Op1`. This choice implies that modifier operations are allowed to cause a one-time (as opposed to permanent) modification of the nominal execution behavior of the task.

OBCS for a sporadic task



We want to encapsulate the implementation of this policy and simply expose to clients of this sporadic task a set of procedures with the signatures of `Op1` and `Op2`; the role of these procedures is to reify the corresponding execution requests. The invocation (type and actual parameters) is recorded in a language-level structure and stored in the OBCS. When the sporadic task fetches a request, it decodes the original request and calls the appropriate operation with the correct parameters.

Sporadic Task — System Types and Task Type

Let us now have a look at the set of types we need to implement this archetype. They are declared in a modified version of the package `System_Types` that we also used in the preceding Ada Gem.

```
with System;
with Ada.Real_Time; use Ada.Real_Time;
with System_Time;
with Ada.Finalization; use Ada.Finalization;

package System_Types is

 -- Abstract parameter type --
 type Param_Type is abstract tagged record
  In_Use : Boolean := False;
 end record;

 -- Abstract functional procedure --
 procedure My_OPCS (Self : in out Param_Type)
  is abstract;

 type Param_Type_Ref is access all Param_Type'Class;
 type Param_Arr is array(Integer range <>)
  of Param_Type_Ref;
 type Param_Arr_Ref is access all Param_Arr;

 -- Request type --
 type Request_T is (NO_REQ, START_REQ, ATC_REQ);

 -- Request descriptor to reify an execution request
 type Request_Descriptor_T is
  record
    Request : Request_T;
    Params : Param_Type_Ref;
  end record;

 -- Parameter buffer
 type Param_Buffer_T(Size : Integer) is
  record
    Buffer : aliased Param_Arr(1..Size);
    Index : Integer := 1;
  end record;

 type Param_Buffer_Ref is access all Param_Buffer_T;
 procedure Increase_Index(Self : in out Param_Buffer_T);
```

We have declared a set of types to represent parameters, a type describing the kinds of requests (START_REQ, ATC_REQ, and an additional kind NO_REQ just for the sake of the explanation), and a request descriptor type to encapsulate the information about invocations of `Op1` and `Op2`. We also declare procedure `My_OPCS(..)`, which is an abstract procedure that represents all possible operations that can be invoked by the sporadic task.

We now continue on with the remainder of the specification of package `System_Types`:

```
 -- Abstract OBCS --
 type OBCS_T is abstract new Controlled
  with null record;
 type OBCS_T_Ref is access all OBCS_T'Class;

 procedure Put(Self : in out OBCS_T;
               Req : Request_T; P : Param_Type_Ref)
  is abstract;

 procedure Get(Self : in out OBCS_T;
               R : out Request_Descriptor_T)
  is abstract;

 -- Sporadic OBCS --
 type Sporadic_OBCS(Size : Integer) is
    new OBCS_T with
  record
    START_Param_Buffer : Param_Arr(1..Size);
    START_Insert_Index : Integer;
    START_Extract_Index : Integer;
    START_Pending : Integer;
    ATC_Param_Buffer : Param_Arr(1..Size);
    ATC_Insert_Index : Integer;
```

```
   ATC_Extract_Index : Integer;
   ATC_Pending : Integer;
   Pending : Integer;
 end record;

 overriding
 procedure Initialize(Self : in out Sporadic_OBCS);

 overriding
 procedure Put(Self : in out Sporadic_OBCS;
         Req : Request_T; P : Param_Type_Ref);

 overriding
 procedure Get(Self : in out Sporadic_OBCS;
          R : out Request_Descriptor_T);

end System_Types;
```

Above, we declare a root type to represent an abstract OBCS (`OBCS_T`) and a `Sporadic_OBCS` type that implements the queueing policy we previously described. `START_Param_Buffer` and `ATC_Param_Buffer` are two distinct circular buffers that are used to store the invocations of the respective types of operation. In addition, we create a buffer for parameters.

The package body follows:

```
package body System_Types is

 -- Sporadic OBCS --
 procedure Initialize (Self : in out Sporadic_OBCS) is
 begin
  Self.START_Pending    := 0;
  Self.START_Insert_Index :=
    Self.START_Param_Buffer'First;
  Self.START_Extract_Index :=
    Self.START_Param_Buffer'First;
  Self.ATC_Pending     := 0;
  Self.ATC_Insert_Index  :=
    Self.ATC_Param_Buffer'First;
  Self.ATC_Extract_Index  :=
    Self.ATC_Param_Buffer'First;
 end Initialize;

 procedure Put(Self : in out Sporadic_OBCS;
         Req : Request_T; P : Param_Type_Ref) is
 begin
  case Req is
   when START_REQ =>
    Self.START_Param_Buffer
      (Self.START_Insert_Index) := P;
    Self.START_Insert_Index :=
      Self.START_Insert_Index + 1;
    if Self.START_Insert_Index >
      Self.START_Param_Buffer'Last then
     Self.START_Insert_Index :=
      Self.START_Param_Buffer'First;
    end if;
    -- Increase the number of pending requests,
    -- but do not overcome
    -- the number of buffered ones
    if Self.START_Pending <
      Self.START_Param_Buffer'Last then
     Self.START_Pending := Self.START_Pending + 1;
    end if;
   when ATC_REQ =>
    Self.ATC_Param_Buffer (Self.ATC_Insert_Index)
      := P;
    Self.ATC_Insert_Index :=
      Self.ATC_Insert_Index + 1;
    if Self.ATC_Insert_Index >
      Self.ATC_Param_Buffer'Last then
     Self.ATC_Insert_Index :=
      Self.ATC_Param_Buffer'First;
    end if;

    if Self.ATC_Pending <
     Self.ATC_Param_Buffer'Last then
     -- Increase the number of pending requests,
     -- but do not overcome
     -- the number of buffered ones
     Self.ATC_Pending := Self.ATC_Pending + 1;
    end if;

   when others => null;
  end case;
  Self.Pending := Self.START_Pending +
           Self.ATC_Pending;
 end Put;

 procedure Get(Self : in out Sporadic_OBCS;
          R : out Request_Descriptor_T) is
 begin
  if Self.ATC_Pending > 0 then
   R := (ATC_REQ,
     Self.ATC_Param_Buffer(Self.ATC_Extract_Index));
   Self.ATC_Extract_Index :=
     Self.ATC_Extract_Index + 1;
   if Self.ATC_Extract_Index >
     Self.ATC_Param_Buffer'Last then
    Self.ATC_Extract_Index :=
     Self.ATC_Param_Buffer'First;
   end if;
   Self.ATC_Pending := Self.ATC_Pending - 1;
   else
    if Self.START_Pending > 0 then
     R := (START_REQ,
        Self.START_Param_Buffer
           (Self.START_Extract_Index));
     Self.START_Extract_Index :=
       Self.START_Extract_Index + 1;
     if Self.START_Extract_Index >
        Self.START_Param_Buffer'Last then
      Self.START_Extract_Index :=
        Self.START_Param_Buffer'First;
     end if;
     Self.START_Pending := Self.START_Pending - 1;
    end if;
  end if;
  R.Params.In_Use := True;
  Self.Pending := Self.START_Pending +
           Self.ATC_Pending;
 end Get;
```

```
  procedure Increase_Index
        (Self : in out Param_Buffer_T) is
  begin
   Self.Index := Self.Index + 1;
   if Self.Index > Self.Buffer'Last then
     Self.Index := Self.Buffer'First;
   end if;
  end Increase_Index;
end System_Types;
```

In the package body we implement the desired queuing policy. Procedure `Put(..)` simply inserts the representation of the incoming request in the queue of the requested operation kind (START_REQ or ATC_REQ). The ordering among requests of the same operation kind is FIFO.

Procedure `Get(..)` is used to extract a request descriptor. We can see that as long as there are pending ATC requests, they are selected based on their arrival order. When the ATC queue is empty, requests for START operations are fetched.

The task that uses this sporadic OBCS has a specification almost identical to the "simple sporadic task" we presented in the preceding Ada Gem. The only difference is that `Get_Request` now also fetches a request descriptor.

```
with System_Types; use System_Types;
with System; use System;
with Ada.Real_Time; use Ada.Real_Time;

generic
  with procedure Get_Request(
           Req : out Request_Descriptor_T;
           Release : out Time);
package Sporadic_Task is

  task type Thread_T (Thread_Priority : Any_Priority;
        MIAT : Integer) is
   pragma Priority (Thread_Priority);
  end Thread_T;

end Sporadic_Task;
```

The body for the task type follows:

```
with System_Time; use System_Time;
package body Sporadic_Task is

  task body Thread_T is
   Req_Desc : Request_Descriptor_T;
   Release : Time;
   Next_Time : Time := System_Start_Time;
  begin
   loop
     delay until Next_Time;
     Get_Request (Req_Desc, Release);
     Next_Time := Release + Milliseconds (MIAT);
     case Req_Desc.Request is
      when NO_REQ =>
         null;
      when START_REQ | ATC_REQ =>
         My_OPCS (Req_Desc.Params.all);
      when others =>
         null;
     end case;
   end loop;
  end Thread_T;

end Sporadic_Task;
```

Notice that the descriptor of the fetched request can be used to discriminate the action to perform according to the type of operation (this is done with the case statement). In our case, if we fetch a request of kind START_REQ or ATC_REQ, we simply execute My_OPCS, that will dynamically dispatch to the requested operation. This mechanism will be clear when, in a later Gem, we complete the picture with the declaration of `Op1` and `Op2` as seen by their clients.

Gem #96: Code Archetypes for Real-Time Programming – Part 4

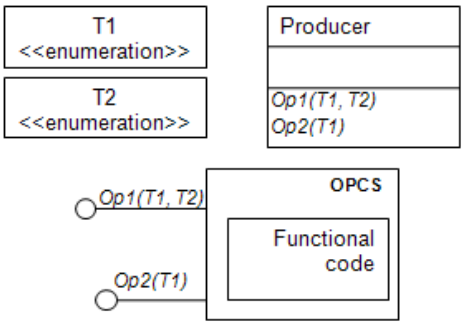
Marco Panunzio, University of Padua

Date: 6 December2010

Abstract: In the previous Ada Gem we started to describe a complete archetype for a sporadic task. We illustrated the structure of the task and the realization of a complex queuing policy for its synchronization agent (OBCS). In this Ada Gem, we complete the picture with the description of the OPCS, which contains the functional code executed by the sporadic task, and show how we complete the declaration of the OBCS and of the provided interface exposed to clients of the sporadic task.

Let's get started…

It is now time to create the functional code for the procedures executed by our sporadic task. Suppose we want a Consumer that provides operations *Op1* and *Op2* as depicted in the figure below.



The operations *Op1* and *Op2* would be included in an OPCS structure which encapsulates their respective functional code, decoupling it from other nonfunctional concerns. The OPCS is then embedded in the sporadic task structure.

First, we declare two simple enumeration types, *T1* and *T2*, in a separate package, to be used by our functional code:

```
package Types is
  type T1 is (F1, F2);
  T1_Default_Value : constant T1 := F1;
  type T2 is (X1, X2);
  T2_Default_Value : constant T2 := X1;
end Types;
```

Then in another package we declare a new type, say *Consumer_FC*, that extends Controlled (thus it is a tagged

type) and has two primitive procedures *Op1(T1, T2)* and *Op2(T1)*:

```
with Types;
with Ada.Finalization; use Ada.Finalization;

package Consumer is

  type Consumer_FC is new Controlled with private;
  type Consumer_FC_Ref is access all
                  Consumer_FC'Class;
  type Consumer_FC_Static_Ref is access all
                  Consumer_FC;
  type Consumer_FC_Arr is array(Standard.Integer
     range <>) of Consumer_FC_Ref;
  type Consumer_FC_Arr_Ref is access
                  Consumer_FC_Arr;

  overriding
  procedure Initialize(This : in out Consumer_FC);

  procedure Op1 (This : in out Consumer_FC;
            a : in Types.T1; b : in Types.T2);
  procedure Op2 (This : in out Consumer_FC;
            a : in Types.T1);

private

  type Consumer_FC is new Controlled
    with null record ...;

end Consumer;
```

Consumer_FC is the type that represents what we called the OPCS. The sequential code is given in the bodies of the two procedures *Op1* and *Op2*:

```
package body Consumer is
  -- procedure Initialize omitted
  procedure Op1(This : in out Consumer_FC;
            a : in Types.T1; b : in Types.T2) is
  begin
  -- User-defined sequential code here --
  end Op1;

  procedure Op2 (This : in out Consumer_FC;
            a : in Types.T1) is
  begin
  -- User-defined sequential code here --
  end Op2;

end Consumer;
```

Now let's complete the definition of the OBCS and discuss the instantiation of the sporadic task.

```
with System;
with Types;
with System_Types;
with Consumer;
with Ada.Real_Time; use Ada.Real_Time;

package Op1_Op2_Sporadic_Consumer is

  use System; use Types;

  use System_Types;

  -- Nested generic package for instantiating a sporadic task:

  generic
   Thread_Priority : Priority;
   Ceiling : Priority;
   MIAT : Integer;
   -- The OPCS instance
   OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
  package My_Sporadic_Factory is

   procedure Op1(a : in T1; b : in T2);
   procedure Op2(a : in T1);

  private
   -- ...
  end My_Sporadic_Factory;

private

 Param_Queue_Size : constant Integer := 3;
 OBCS_Queue_Size : constant
    Integer := Param_Queue_Size * 2;

 -- Create data structures to reify invocations of Op1

 type Op1_Param_T is new Param_Type with record
  OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
  a : T1;
  b : T2;
 end record;

 type Op1_Param_T_Ref is access all Op1_Param_T;

 type Op1_Param_Arr is array(Integer range <>)
   of aliased Op1_Param_T;

 overriding
 procedure My_OPCS(Self : in out Op1_Param_T);

 -- Create data structures to reify invocations of Op2

  type Op2_Param_T is new Param_Type with record
   OPCS_Instance : Consumer.Consumer_FC_Static_Ref;
   a : T1;
  end record;

 type Op2_Param_T_Ref is access all Op2_Param_T;

 type Op2_Param_Arr is array(Integer range <>)
   of aliased Op2_Param_T;

 overriding
 procedure My_OPCS(Self : in out Op2_Param_T);

 -- Create an OBCS that matches the interface of the
 -- OPCS (FC)
 protected type OBCS
  (Ceiling : Priority;
  Op1_Params_Arr_Ref_P : Param_Arr_Ref;
  Op2_Params_Arr_Ref_P : Param_Arr_Ref)
 is
  pragma Priority(Ceiling);

  entry Get_Request(Req : out Request_Descriptor_T;
                    Release : out Time);
  procedure Op2(a : in T1);
  procedure Op1(a : in T1; b : in T2);

 private

   -- The queue system for the OBCS
```

```
   OBCS_Queue : Sporadic_OBCS(OBCS_Queue_Size);
   -- Arrays to store a set of reified invocations
   -- for Op1 and Op2
   Op1_Params : Param_Buffer_T(Param_Queue_Size) :=
    (Size => Param_Queue_Size, Index => 1,
     Buffer => Op1_Params_Arr_Ref_P.all);
   Op2_Params : Param_Buffer_T(Param_Queue_Size) :=
    (Size => Param_Queue_Size, Index => 1,
     Buffer => Op2_Params_Arr_Ref_P.all);
   Pending : Standard.Boolean := False;

 end OBCS;

end Op1_Op2_Sporadic_Consumer;
```

In essence, in the specification above: (i) we declare a nested generic package (*My_Sporadic_Factory*) that we use to instantiate a sporadic task. In this manner we can instantiate several sporadic tasks which only differ in their timing attributes and properties (MIAT, priority and ceiling priority for the OBCS); the generic package provides an interface to the rest of the system that matches its OPCS (it provides *Op1* and *Op2* in our case); (ii) in the private part of the parent package (*Op1_Op2_Sporadic_Consumer*), we create the data structures to store reified invocations of *Op1* and *Op2*. This is done by extending *Param_Type* (defined in System_Types, see the previous Ada Gem in this series) by new types *Op1_Param_T* and *Op2_Param_T* that are records containing the parameters of the call and a reference to the OPCS (an access to *FC* in our case). Additionally, we override procedure *My_OPCS*. Therefore, when *My_OPCS* is called on *Op1_Param_T* or *Op2_Param_T*, it will dispatch to the appropriate procedure that we later define in the body of this package. The reader can check again that this is what really happens when the sporadic task type (defined in the previous Gem in this series) calls the procedure *My_OPCS* after fetching the request descriptor from the OBCS.

```
with Ada.Real_Time; use Ada.Real_Time;

with Sporadic_Task;
with Types; use Types;

package body Op1_Op2_Sporadic_Consumer is

 use System_Types;

 -- Redefinition of My_OPCS. Call Consumer_FC.Op1
 -- and set In_Use to False.

 procedure My_OPCS(Self : in out Op1_Param_T) is
 begin
  Self.OPCS_Instance.Op1(Self.a, Self.b);
  Self.In_Use := False;
 end My_OPCS;

 -- Redefinition of My_OPCS. Call Consumer_FC.Op2
 -- and set In_Use to False.
 procedure My_OPCS(Self : in out Op2_Param_T) is
 begin
  Self.OPCS_Instance.Op2(Self.a);
  Self.In_Use := False;
 end My_OPCS;

 protected body OBCS is

  procedure Update_Barrier is
  begin
   Pending := (OBCS_Queue.Pending) > 0;
  end Update_Barrier;

  -- Get_Request stores the time of the release of the task,
  -- gets the next request (according to the OBCS
  -- queuing policy), and updates the guard.

  entry Get_Request (Req: out Request_Descriptor_T;
                     Release: out Time) when Pending is
  begin
   Release := Clock;
   Get(OBCS_Queue, Req);
   Update_Barrier;
  end Get_Request;

  -- When a client calls Op1, the request is reified
  -- and put in the OBCS queue.

  procedure Op1(a : in T1; b : in T2) is
  begin
   if Op1_Params.Buffer(Op1_Params.Index).In_Use then
    Increase_Index(Op1_Params);
   end if;

   Op1_Param_T_Ref(Op1_Params.Buffer(
                     Op1_Params.Index)).a := a;
   Op1_Param_T_Ref(Op1_Params.Buffer(
                     Op1_Params.Index)).b := b;
   Put(OBCS_Queue, START_REQ,
       Op1_Params.Buffer(Op1_Params.Index));
   Increase_Index(Op1_Params);
   Update_Barrier;
  end Op1;

  -- When a client calls Op2, the request is reified
  -- and put in the OBCS queue.

  procedure Op2(a : in T1) is
  begin
   if Op2_Params.Buffer(Op2_Params.Index).In_Use then
    Increase_Index(Op2_Params);
   end if;

   Op2_Param_T_Ref(Op2_Params.Buffer(
                     Op2_Params.Index)).a := a;
   Put(OBCS_Queue, ATC_REQ,
       Op2_Params.Buffer(Op2_Params.Index));
   Increase_Index(Op2_Params);
   Update_Barrier;
  end Op2;

 end OBCS;

 package body My_Sporadic_Factory is

  Op1_Par_Arr :
     Op1_Param_Arr(1..Param_Queue_Size) :=
       (others =>
        (False,
        OPCS_Instance,
        T1_Default_Value,
        T2_Default_Value));

  Op1_Ref_Par_Arr : aliased Param_Arr :=
      (Op1_Par_Arr(1)'access,
```

```
        Op1_Par_Arr(2)'access,
        Op1_Par_Arr(3)'access);

   Op2_Par_Arr :
     Op2_Param_Arr(1..Param_Queue_Size) :=
      (others =>
       (false,
        OPCS_Instance,
        T1_Default_Value));

   Op2_Ref_Par_Arr : aliased Param_Arr :=
      (Op2_Par_Arr(1)'access,
       Op2_Par_Arr(2)'access,
       Op2_Par_Arr(3)'access);

   -- Creation of the OBCS
   Protocol : aliased OBCS(Ceiling,
                           Op1_Ref_Par_Arr'access,
                           Op2_Ref_Par_Arr'access);

   -- Indirection to Get_Request of the OBCS

   procedure Getter(Req : out Request_Descriptor_T;
                    Release : out Time) is
   begin
    Protocol.Get_Request(Req, Release);
   end Getter;

   -- Instantiate the generic package using
   -- the procedure above

   package My_Sporadic_Task is new
      Sporadic_Task(Getter);

   Thread : My_Sporadic_Task.Thread_T(Thread_Priority,
                                      MIAT);

   -- When a client calls Op1, redirect the call to the OBCS
   procedure Op1(a : in T1; b : in T2) is
   begin
    Protocol.Op1(a, b);
   end Op1;

   -- When a client calls Op2, redirect the call to the OBCS
   procedure Op2(a : in T1) is
   begin
    Protocol.Op2(a);
   end Op2;

  end My_Sporadic_Factory;

end Op1_Op2_Sporadic_Consumer;
```

The package body above overrides *My_OPCS* for each operation provided to external clients (*Op1* and *Op2*). The overriding simply ensures that *My_OPCS* calls the correct operation with the stored parameter of the original request and then signals that the parameters are no longer in use (which ensures correct management of the circular buffers in the OBCS).

The body of the OBCS follows the same logic as the simpler OBCS described in Gem #92. Procedure *Op1* and *Op2* are simply extended to reify call requests and correctly store the parameters of the calls in the request descriptor.

Finally, in the body of the generic package *My_Sporadic_Factory*, we create an OBCS with a defined ceiling priority and the queues to store the parameters of reified calls to *Op1* and *Op2*. The sporadic thread is instantiated in the same package, and we complete the picture by redirecting the calls from *Op1* and *Op2*, in the provided interface of the task structure, to the operations with the same names in the OBCS.

Gem #103: Code Archetypes for Real-Time Programming – Part 5

Marco Panunzio, University of Padua

Date: 11 April 2010

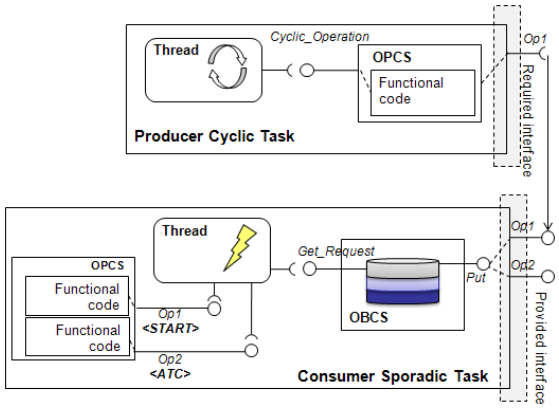
Abstract: In the previous Ada Gem we completed the creation of a complete sporadic task. In this Ada Gem that ends our miniseries, we want to complete the example by adding the realization of the communication between different tasks. In particular, we investigate how we can correctly manage the calls of operations outside a task. Those calls are performed from an OPCS and they have to be correctly routed to the endpoint of the communication.

Let's get started…

Intertask communication —

A producer-consumer example

Suppose we want to realize the simple producer-consumer collaboration pattern depicted in the figure below.



The *Producer* is a cyclic task which, after some processing, produces some data that is sent to a *Consumer* sporadic task. Data is passed as a parameter of operation *Op1*.

This implies that we have to equip the *Producer* cyclic task with the means to communicate with the *Consumer* sporadic task. However, the task structure that we created encapsulates the functional code inside a structure called the OPCS. Therefore, inside the functional code of the Producer we cannot directly call *Op1* of the Consumer (i.e., the functional/sequential code), but we have to call the appropriate provided interface of the whole Consumer task.

Let us see how we can achieve this goal when creating the package Producer (analogous to the package Consumer that we presented in the previous Ada Gem of the series).

```
package Producer is

   type Producer_FC is new Controlled with private;
   type Producer_FC_Ref is access all Producer_FC'Class;
   type Producer_FC_Static_Ref is access all Producer_FC;
```

```
  -- [code omitted]

  overriding
  procedure Initialize(This : in out Producer_FC);

  procedure Op0 (This : in out Producer_FC);
  procedure Set_x(This : in out Producer_FC;
        v : in Consumer.Consumer_FC_Ref);

private

  type Producer_FC is new Controlled with record
   x : Consumer.Consumer_FC_Ref;
  end record;

end Producer;
```

We are adding a member *x* in the record of *Producer_FC* that represents a reference to the Consumer that provides *Op1*, which consumes the data produced by the Producer. The reader should note that the static type of this reference is the OPCS of the Consumer (which was called *Consumer_FC*).

```
package body Producer is

  -- [procedure Initialize omitted]

  procedure Op0(This : in out Producer_FC) is
  begin
   This.x.Op1([T1_VALUE]);
  end Op1;

  procedure Set_x(This : in out Producer_FC;
        v : in Consumer.Consumer_FC_Ref)
  is
  begin
   This.x := v;
  end Set_x;
end Producer;
```

In the body of procedure *Op0* (where the sequential code executed by the Producer task is specified), we insert the call to *Op1* that is performed using reference *x*.

```
-- Package spec
type s1_T is new Consumer.Consumer_FC with record
  Op1_Ref : access procedure (a : in Types.T1;
                              b : in Types.T2);
  Op2_Ref : access procedure (a : in Types.T1);
end record;

overriding
procedure Op1(This : in out s1_T; a : in Types.T1;
              b : in Types.T2);

overriding
procedure Op2(This : in out s1_T; a : in Types.T1);

-- Package body
procedure Op1(This : in out s1_T; a : in Types.T1;
              b : in Types.T2) is
begin
  This.Op1_Ref.all(a,b);
end Op1;
```

Above, in another package, we create a new type *s1_T*. This type extends *Consumer_FC* and adds a pointer to a procedure with the signature of *Op1*, the operation that we call at the Producer side, and *Op2*. We also override *Op1* for *s1_T*, so that a call to *Op1* is reissued to the pointer just defined.

Analogously, suppose we create a new type *s0_T* that extends *Producer_FC*.

In the few remaining code excerpts below, we complete the example with the instantiation of the cyclic task for the Producer and the sporadic task for the Consumer.

```
s0_Instance : aliased s0_T;
package My_Cyclic_Producer_Task is new
  Op0_Cyclic_Producer.My_Sporadic_Factory(
    Thread_Priority => 1,
    Period => 2000,
    OPCS_Instance =>
     Producer.Producer_FC(s0_Instance)'access);

s1_Instance : aliased s1_T;

package My_Sporadic_Consumer_Task is new
  Op1_Op2_Sporadic_Consumer.My_Sporadic_Factory(
    Thread_Priority => 2,
    Ceiling => 2,
    MIAT => 500,
    OPCS_Instance =>
     Consumer.Consumer_FC(s1_Instance)'access);
```

Note that we pass a pointer to *s0_Instance* (respectively *s1_Instance*) as the OPCS during the instantiation of the Producer (respectively Consumer) cyclic (respectively sporadic) task.

```
s1_Instance.Op1_Ref :=
  My_Sporadic_Consumer_Task.Op1'access;
s1_Instance.Op2_Ref :=
  My_Sporadic_Consumer_Task.Op2'access;
```

Finally, with the assignment above, we are able to impose that whenever *Op1* is called on *s1_Instance*, then the call is directed to the operation Op1 on the provided interface of the Consumer Sporadic Task, and from there it follows the correct delegation chain (redirection to the OBCS and reification of the request in the OBCS queue).

```
s0_Instance.Set_x(
  Consumer.Consumer_FC_Ref(s1_Instance'access));
```

Finally, we establish the binding between the Producer and the Consumer with the call above. In fact, it ensures that the call of *Op1* inside the OPCS of the Producer (*Producer_FC*) is a call to the provided interface of the Consumer sporadic task.

Conclusion

In this Ada Gems miniseries we described a set of Ravenscar-compliant code archetypes for the realization of recurrent patterns in real-time systems. We presented two basic patterns for the realization of cyclic and sporadic tasks, commented on their drawbacks, and showed how to improve them to realize sporadic operations with parameters and an example of complex queuing policy. Finally, we showed how to perform intertask communication.

The code archetypes we described were used for the code generation in the HRT-UML/RCM track of the EU-funded ASSERT project.

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. José Javier Gutiérrez
Ada-Spain
P.O.Box 50.403
28080-Madrid
Spain
Phone: +34-942-201-394
Fax: +34-942-201-402
Email: gutierjj@unican.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