ADA USER JOURNAL

Volume 27
Number 1
March 2006

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication else-where.

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

In my tenure of the editorship I have come to learn that the March issue of the Ada User Journal always lies in a kind of in between old and new business. This particular issue adds evidence to this observation. By publishing the epilogue of John Barnes' Rationale for Ada 2005 we do close an important business indeed, which accompanied us for 6 issues, ever since December 2004. That was a jolly good time, I have to say, owing to both the enthusiasm for seeing Ada 2005 materialise and the writing art of the author, which was particularly enjoyable. As for new business, well, the yearly Ada-Europe conference looms on the horizon, which will be the source of inspiration for the Ada community and of contents for a numerous future issues of the journal. We all look forward to it. In addition to John Barnes' closing epilogue of the Rationale and the usual wealth of information off the News and Calendar sections, we host an article by Muthu Ramachandran (an author we hosted already in issue 26-2) from the University of Leeds, where we are happy to have some friends of Ada. Happy reading!

Tullio Vardanega
Padova
March 2006
Email: tullio.vardanega@math.unipd.it

News

Santiago Urueña
Technical University of Madrid (UPM). Email: suruena@datsi.fi.upm.es

Contents

Ada-related Organizations

ARA – Ada 2005 Rationale Available

From: Randy Brukardt <randy@rrsoftware.com>
Date: Sat, 25 Feb 2006 18:57:37 -0600
Subject: Complete Rationale for Ada 2005 available on AdaIC
Newsgroups: comp.lang.ada

The ARA is proud to announce that the complete Ada 2005 Rationale is now available on its website at http://www.adaic.com/standards/rationale05.html.

The Rationale for Ada 2005 provides an overview of Ada 2005 features, examples of their use, compatibility with Ada 95, and more. It was written by John Barnes, and was sponsored in part by the Ada Resource Association
[and by Ada-Europe – su].

It is available in HTML and PDF formats. It is currently available only on-line (not for download), as there is an overall revision planned to convert it into a single book format.

Randy Brukardt
Technical Webmaster, Adaic.org/.com

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. -- su]

February 2 – Ada GPS/UML Webinar

From: Marc A. Criley <mc@mckae.com>
Date: Tue, 24 Jan 2006 13:12:20 -0600
Subject: Ada GPS/UML Webinar
Newsgroups: comp.lang.ada

I am just passing this info on as an FYI, since I am on Artisan's mailing list:

Presented by ARTiSAN and AdaCore, the leading developer of Ada technology, this webinar will provide an overview of Ada 2005 and how AdaCore's development environment, "GNAT Programming Studio" (GPS) and ARTiSAN Studio can be used as a workable tool-chain.

Areas of demonstration will include:

Introduction to the new features contained within Ada 2005 (AdaCore)
The GPS Development Environment (AdaCore)
Reverse Engineering Ada 95 (ARTiSAN)
Forward Generating Ada 2005 (ARTiSAN)

The Webinar lasts for about an hour and will run at the following times:

Thursday 2nd February -- 10am EST / 3pm GMT / 4pm CET
Thursday 2nd February -- 1pm EST / 6pm GMT / 7pm CET

To register for this Webinar, click here: http://www.artisansw.com/seminars/UMLwebinar_USreg2.asp

Joining instructions will be emailed through to you separately. There is no charge for people dialing in from the US, UK and France.

February 26 – FOSDEM 2006 Presentations Available

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Date: 20 Feb 2006 21:30:10 +0100
Organization: Ada-Belgium, c/o Dept. of Computer Science, K.U.Leuven
Subject: FOSDEM 2006 - Ada "Developers Room", Sun 26 Feb 2006, Brussels
Newsgroups: comp.lang.ada,fr.comp.lang.ada

Next Sunday Ada-Belgium organizes a full-day Ada "Developers Room" at FOSDEM 2006, the Free and Open Source Developers' European Meeting in Brussels.

An updated version of the program is attached; HTML and PDF versions are available on the Ada-Belgium web site for further distribution.

We invite you to attend some or all of the presentations; they will be given in English. Attendance to FOSDEM is free and no registration is necessary.

Dirk Craeynest, President Ada-Belgium, Dirk.Craeynest@cs.kuleuven.be

Ada-Belgium is pleased to announce its

Ada "Developers Room"
at
FOSDEM 2006
(Free and Open-Source Software Developers' European Meeting)
Sunday, February 26, 2006, 10:00-17:00
Universitè Libre de Bruxelles (U.L.B.), Solbosch Campus
Avenue Franklin D. Roosevelt Laan 50, B-1050 Brussels
http://www.cs.kuleuven.be/~dirk/ada-belgium/events/06/060225-fosdem.html

The Free and Open-Source Developers' Meeting (FOSDEM) is an annual event held in Brussels, Belgium, in February. The 2006 edition will take place on Saturday the 25th and Sunday the 26th of February, 2006. Ada-Belgium has organized a series of presentations related to Ada, to be held in a dedicated developers' room, all day Sunday. Here is the program:

10:00 - 11:00 Jean-Pierre Rosen: Introduction to Ada

Jean-Pierre will put his well-known talent to good use, introducing Ada to beginning or experienced programmers alike.

11:00 - 12:00 Jean-Pierre Rosen: AdaControl

AdaControl is a tool that analyses Ada source text and verifies compliance with coding rules and guidelines. AdaControl is Free Software written under contract with Eurocontrol, and takes advantage of ASIS, the standard interface that allows Ada programs to analyze Ada source text. Jean-Pierre will introduce AdaControl, ASIS, and the business model that allows one to make a living writing Free Software.

12:00 - 13:00 Philippe Waroquiers: Use of Free Software in European Air Traffic Flow Management

Philippe Waroquiers leads software development of the ETFMS system at Eurocontrol, the European air traffic control agency with 34 member states. Software on which millions of travelers' lives each year depend is written in Ada using AdaCore's Free Software Ada compiler, GNAT Pro.

13:00 - 14:00 lunch break

14:00 - 15:00 Ludovic Brenta: Ada in Debian

Ludovic Brenta will explain his work as the main maintainer of Ada in Debian, and the policy that unites all Ada packages, thereby making Debian the best free Ada development platform in the world :) This will be an excellent opportunity for a tour of existing Free Software projects developed in Ada.

15:00 - 16:00 Robert Dewar, AdaCore: Ada Academic Initiative

AdaCore is the company that offers technical support and consulting services around GNAT Pro, the professional version of the GNU project's Free Software Ada compiler. AdaCore is also the main developer of GNAT. The Ada Academic Initiative aims to encourage universities and other education institutions worldwide to use and teach Ada, by offering a broad range of services at no cost to professors and students. If possible, AdaCore will demonstrate the latest GNAT Programming Studio available with the GNAT GPL 2005 Edition.

16:00 - 17:00 Thomas Quinot, AdaCore:

The PolyORB schizophrenic middleware An example of fruitful collaboration between academia and industry, PolyORB allows heterogeneous software components to communicate with one another by bridging various middleware technologies such as CORBA, MOM and the Ada Distributed Systems Annex (annex E).

All presentations will be in English, but most speakers also speak French. You may ask questions on comp.lang.ada, fr.comp.lang.ada, or join the AdaFOSDEM mailing list (in English). Attendance to FOSDEM is free, and no registration is necessary.

More information:

* FOSDEM: http://www.fosdem.org
* AdaCore: http://www.adacore.com
* Free Software from AdaCore: http://libre.adacore.com (includes, among others, GNAT, GPS and PolyORB which will be the focus of some talks)
* Free Software from Adalog: http://www.adalog.fr/compo1.htm (includes, among others, AdaControl)

* Debian: http://www.debian.org
* Eurocontrol: http://www.eurocontrol.int
* Ada-Belgium: http://www.cs.kuleuven.be/~dirk/ada-belgium/
* AdaFOSDEM mailing list, operated by Ada-Belgium: http://listserv.cc.kuleuven.be/archives/adafosdem.html

[See also "FOSDEM 2006" in AUJ 26-4 (Dec 2005), p.231. -- su]

All presentations at the Ada Developers Room, held at FOSDEM 2006 in Brussels on Sunday February 26, 2006, are available online on the Ada-Belgium web-site, both in the original format (ODP or PPT) and in PDF:

http://www.cs.kuleuven.be/~dirk/ada-belgium/events/06/060226-fosdem.html

June 5-9 – Ada-Europe 2006

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Date: 31 Dec 2005 17:49:03 +0100
Organization: Ada-Europe, c/o Dept. of Computer Science, K.U.Leuven
Subject: 2nd CfIP, Conference Reliable Software Technologies, Ada-Europe 2006
Newsgroups: comp.lang.ada,fr.comp.lang.ada

This call for industrial presentations is specifically targeted to those of you who either work in industrial (Ada-related) projects where reliable software technologies are important, or know people working in such projects.

Please think for a moment what others might learn from the experience gained in those projects, and consider (or convince them) to submit a one-page presentation overview by January 12th, 1.5 weeks from now.

Many projects could report a lot of valuable experience: sharing it with others benefits the whole community and might provide useful feedback as well.

We're looking forward to receive many interesting presentations.

Best wishes for the new year,

Dirk Craeynest, Ada-Europe'2006 Publicity Chair

2nd Call for Industrial Presentations
11th International Conference on Reliable Software Technologies
Ada-Europe 2006
5-9 June 2006, Porto, Portugal
http://www.ada-europe.org/conference2006.html

Organized, on behalf of Ada-Europe, by Instituto Superior de Engenharia do Porto in cooperation with ACM SIGAda

*** CfIP in HTML/PDF on web site ***
DEADLINE Thursday 12 Jan. 2006

General Information

The 11th International Conference on Reliable Software Technologies (Ada-Europe 2006) will take place in Porto, Portugal. Following the usual style, the conference will span a full week, including a three-day technical program and vendor exhibitions from Tuesday to Thursday, along with parallel workshops and tutorials on Monday and Friday.

Call for Presentations

In addition to the usual call for papers, and considering the success achieved in the previous conference, we are having a call for presentations primarily aimed at industrialists who have valuable experience to report but who do not wish to write a complete paper.

This separate call for presentations is made for Experience Reports from Industrial Projects and/or Experiments, Case Studies and

See below for further details.

Schedule

12 January 2006: Submission of presentation proposals

20 January 2006: Notification to authors

28 April 2006: Presentation material required

5-9 June 2006: Conference

Submission of Presentations

Presenters are invited to submit a one-page overview of the proposed presentation to Peter Dencker (peter.dencker@aonix.de) by January 12th 2006. The Industrial Committee will review the proposals.

The authors of selected presentations shall prepare their final presentation, together with a short abstract (max 10 lines), by 28th April 2006; they should aim at a 20 minutes talk. The authors of accepted presentations will also be invited to derive articles from them, for publication in the Ada User Journal.

Exhibitions

Commercial exhibitions will span the three days of the main conference. Vendors and providers of software products and services should contact the Exhibition Chair José Ruiz as soon as possible for further information and for allowing suitable planning of the exhibition space and time.

Conference Topics

In the last decade the conference has established itself as an international forum for providers and practitioners of, 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 from industry, academia and government organizations interested in furthering the development of reliable software technologies. To mark the completion of the technical work for the Ada language standard revision process, contributions that present and discuss the potential of the revised language are particularly sought after.

For papers, tutorials, and workshop proposals, the topics of interest include, but are not limited to:

- Methods and Techniques for Software Development and Maintenance: Requirements Engineering, Object-Oriented Technologies, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues

- Software Architectures: Patterns for Software Design and Composition, Frameworks, Architecture-Centered Development, Component and Class Libraries, Component-Based Design

- Enabling Technology: CASE Tools, Software Development Environments and Project Browsers, Compilers, Debuggers and Run-time Systems

- Software Quality: Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems

- Critical Systems: Real-Time, Distribution, Fault Tolerance, Information Technology, Safety, Security

- Mainstream and Emerging Applications: Multimedia and Communications, Manufacturing, Robotics, Avionics, Space, Health Care, Transportation

- Ada Language and Technology: Programming Techniques, Object-Oriented Programming, Concurrent Programming, Distributed Programming, Bindings and Libraries, Evaluation & Comparative Assessments, Critical Review of Language Enhancements, Novel Support Technology, HW/SW platforms

- Experience Reports: Experience Reports, Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics, Experience Reports on Education and Training Activities with bearing on any of the conference topics

Tutorials

Tutorials should address subjects that fall within the thrust 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 Jorge Real. The providers of 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 accordingly be halved. The Ada User Journal will offer space for the publication of summaries of the accepted tutorial in issues preceding and/or following the conference.

Organizing Committee

Conference Chair

Luís Miguel Pinho, Polytechnic Institute of Porto, Portugal, lpinho@dei.isep.ipp.pt

Industrial Committee Co-Chairs

Peter Dencker, Aonix GmbH, Germany, peter.dencker@aonix.de
Michael González Harbour, Universidad de Cantabria, Spain, mgh@unican.es

Industrial Committee

Rod Chapman, Praxis High Integrity Systems
Christopher Smith, Green Hills
Franco Gasperoni, AdaCore
Jacques Brygier, Aonix
Ian Gilchrist, IPL
Pascal Leroy, IBM Rational
Rei Strahle, Saab Systems
Francis Thom, Artisan Software
Tony Elliston, TNI Europe
Amar Bouali, Esterel Technologies
Luís Miguel Pinho, Conference Chair
Dirk Craeynest, Ada-Europe (Vice President)
Erhard Ploedereder, Ada-Europe (President)

[See also same topic in AUJ 26-4 (Jun 2004), pp.231-232. -- su]

March 28 – Ada Conference UK 2006

Ada Conference UK 2006
Building better, safer software
28 March 2006 -- Lowry Hotel
Manchester, UK

Ada answers.

As the need for robust and reliable software systems increases, Ada continues to prove itself as the answer for many of today's most complex programming challenges -- especially in the areas of real time, embedded and safety-critical applications.

Event focus: to promote awareness of the Ada 2005 language revision, and to highlight the increased relevance of Ada in safety-critical programming.

Event outline: Plenary sessions by Robert Dewar and John Barnes, plus a series of technical talks by leading industrial experts, plus a stream of vendor talks running in parallel; also a broad range of leading Ada product vendors will be present in the exhibition atrium. Full details of the event and programme can be found at:
www.ada-uk-conference.co.uk

Event opportunities:

Meet members and colleagues from all sectors of the Ada community; this high-calibre event is expected to attract many professional Ada users and reinforce links between all sections of the Ada community in the UK and beyond.

Examine an extensive range of technologies from the leading Ada toolset and service vendors.

Learn about the latest revision of the Ada programming language and the improvements it offers, notably:

- Comprehensive support for real-time and high-reliability applications

- Enhanced Object-Orientated Programming features and an abstraction mechanism that combines OOP and concurrency

- Generalised program structure and visibility control

- Better access type facilities

Discuss innovative and challenging experiences with the Ada language.

Event location: the conference venue is in the heart of Manchester, at the award-winning Lowry Hotel, with excellent access by road, rail and air.

Registration and enquiries:
Joan.Atkinson@ncl.ac.uk
+44 191 221 2222

The not-to-be missed Ada event of 2006!!

Event lead sponsor and advocate: AdaCore, www.adacore.com

Event sponsor: Green Hills Software, Inc., www.ghs.com

Event sponsor: Wind River, www.windriver.com

Event operated by CSR, in cooperation with the Safety-Critical Systems Club: www.safety-club.org.uk

Ada-related Tools

PragmARC – PragmAda Reusable Components

From: PragmAda Software Engineering <pragmada@earthlink.net>
Date: Tue, 28 Feb 2006 20:48:34 GMT
Subject: New release of the PragmAda Reusable Components
Newsgroups: comp.lang.ada

PragmAda Software Engineering is proud to announce a new release of the PragmAda Reusable Components. This release includes a new component to perform least-squares line fitting to a set of data, and some improvements and added functionality to existing components.

You may download the PragmARCs from:
http://home.earthlink.net/~jrcarter010/pragmarc.htm

Comments and error reports are welcome from all users.

If you'd like to receive such announcements directly, send an email to pragmada@earthlink.net. PragmAda Software Engineering will not use your contact information except to send you the requested announcements, and will not transfer your contact information to another person or entity except as required by law.

[See also same topic in AUJ 26-3 (Sep 2005), pp.152-153. -- su]

The GNU Ada Compiler Project

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Fri, 16 Dec 2005 20:03:45 +0100
Subject: The GNU Ada compiler
Newsgroups: comp.lang.ada

I have finally managed to take over the "The GNU Ada compiler" project. The project will supply community compiled binary packages of the GNAT compiler.

Currently there is only a very old GNAT for DOS available. But as I type this the first packages for Linux are uploaded.

A new homepage will follow soon and until then you can just look at the project page: https://sourceforge.net/projects/gnuada

Of course I need help if this project is really to take off. If you have an rpm based Linux system you can just download the RPM kit (as rpm or via CVS) and you should have a rpm package in no time.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Sun, 18 Dec 2005 19:55:31 +0100
Subject: The GNU Ada: Homepage online.
Newsgroups: comp.lang.ada

I also got the homepage online today. The first page added is about creating RPM packages. As I said: this project can only be a success if as many maintainers join in to provide as many platforms as possible.

There where lots of discussion about the GNAT/GPL edition and that we need GMGPL editions as well. And that a community effort might help here. Now here we are.

And while the compile takes several hours typing "rpmbuild -ba" won't take more then a few seconds. So we should at least get a good collection of rpm based packages out of the door.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 21 Dec 2005 16:07:57 +0100
Subject: Re: The GNU Ada compiler
Newsgroups: comp.lang.ada

Steve Whalen wrote:

> In particular for proselytizing Ada, stable GMGPL Ada compilers need to be available for Windows (Ming and Cygwin), Redhat, SUSE, Mandrake, and Solaris.

I agree wholeheartedly. For Windows, there are already two binary distributions of GNAT under GMGPL: AIDE[1], and MinGW[2]. I have tried neither of them (since I don't have Windows) but it seems to me that MinGW has the larger mind share, and AIDE the better quality thanks to the dedication of its maintainer, Stéphane Riviére.

Solaris also has a binary distribution[3,4] containing both GCC 3.4.4 and 4.0.1 with Ada support; but I have never tried it and I cannot assess its quality.

FreeBSD[5] seems not to be very active WRT Ada, but there are ports for GNAT 3.15p, ASIS and GLADE. The ports for GCC 3.4.4, 4.0.3, 4.1.0 and 4.2.0 (the latter three being works in progress) lack Ada support. I am quite confident that FreeBSD's GCC maintainer would gratefully accept patches to enable Ada on that platform.

[1] http://stephane.rochebrune.org/aide/aide.html

[2] http://www.Mingw.org

[3] http://www.blastwave.org

[4] http://www.canoedissent.org.uk/ss/type.jsp?c=prog

[5] http://www.freebsd.org

Now, it would be nice if Red Hat, SuSE and Mandriva would improve their support for Ada. The best way to make this happen is to lobby them, join their GCC maintenance teams, and contribute. Martin's project on SourceForge is a good testing ground for patches. Well tested patches and build scripts stand a good chance of being accepted into these distributions.

From: Steve Whalen <SteveWhalen001@hotmail.com>
Date: 23 Dec 2005 16:28:34 -0800
Subject: Re: The GNU Ada compiler
Newsgroups: comp.lang.ada

Martin Krischik wrote:

> Currently I concentrate much on the GNAT/GPL but I think that may change when GCC 4.1 is out of the door. Apart from that I monitor the download stats to see what is appreciated and what is wasted effort. Interestingly enough currently source-rpm are more in demand then actual binaries.

[...] Actually, since NYU is _not_ mirroring 3.15p any more, it would be helpful if you used SourceForge to house all of the 3.15p versions, so we would have a single place to send anyone interested in Ada. Then as newer versions became stable for each platform / OS combination, you could push 3.15p down into an "older version" status.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Wed, 28 Dec 2005 20:30:50 +0100
Subject: [gnuada]
Newsgroups: comp.lang.ada

There are several new releases available at the gnuada (http://gnuada.sourceforge.net/) project.

Especially the GNU/3 section should be fairly complete. Still if anybody knows some more packages from the 3.15 area which are missing I appreciate any hint.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Sun, 18 Dec 2005 22:04:11 +0100
Subject: Re: The GNU Ada compiler
Newsgroups: comp.lang.ada

Bjorn Persson wrote:

> Do your packages for SuSE replace the GNAT packages that SuSE provides, or can they coexist?

I never ever would replace the SuSE packages – you need them to compile the kernel!

> In the latter case, how does the user choose which compiler to use?

The classic way:

PATH=/opt/gnat/bin:${PATH}

> I've got the impression that when interfacing to other languages all the pieces need to be compiled with the same version of GCC, so if these releases only provide Ada I suppose they shouldn't be used for mixed-language projects. Or am I misinformed?

That is indeed true and I compile all languages which with the packages so there won't be any problems. Of course that makes the packages as large as they are :-(.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: 10 Jan 2006 10:47:54 -0800
Subject: [GNUADA] "R2" - Now you can install GNAT/GPL and GNAT/GCC at once.
Newsgroups: comp.lang.ada

This is the Release 2 of the GNAT distribution. This Distribution will install GNAT/GCC and GNAT/GPL in two different directories so you can install them both.

You can switch between them using a new configuration Script.

The initial package consists "GNAT/GCC 4.0.2" and "GNAT/GPL 2005" for "SuSE 10.0 x86_64" and "SuSE 9.2 i586" with all the libraries and tools we currently distribute.

From: Bjorn Persson <rombo.bjorn.persson@sverige.nu>
Date: Sat, 18 Mar 2006 15:56:46 GMT
Subject: Re: [gnuada] gcc 4.1.0 available
Newsgroups: comp.lang.ada

The first release for Fedora from the GNU Ada Project is out. It is packaged for Fedora Core 4 on i386, and includes GCC 4.1.0 and GNAT/GPL 2005 with ASIS, the Booch components, GDB, GTK/Ada and XML/Ada. For the GCC edition Glade is also included.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Sun, 12 Mar 2006 17:34:51 +0100
Subject: [gnuada] gcc 4.1.0 available
Newsgroups: comp.lang.ada

The GNU Ada Project [http://gnuada.sourceforge.net/] is pleased to announce a new GNAT release based on GCC 4.1.0. The Release is currently available for "SuSE 10.0 x86_64" and "Solaris 10 UltraSparc" – others are to follow.

The SuSE release consist of all GCC core languages (Ada, C, C++, Fortran, Java, Objective-C, Objective-C++) and all currently supported libraries and tools (ASSIS, Boochs, GDB, GtkAda, XML/Ada).

The Solaris release consists of Ada, C and C++.

SNMP for Ada

From: Stephane Riviere <stephane@rochebrune.org>
Date: Sat, 04 Mar 2006 16:37:15 +0100
Subject: Re: Ada et SNMP ?
Newsgroups: fr.comp.lang.ada

[Translated from French. -- su]

> Do you know of any package that supports SNMP ?

www.ijs.co.nz/code/ada95_snmp_2.zip

That's more for NT or FreeBSD if I have well understood ☺

SNMP is not that simple. Version 1, the simplest and the most up to date is not very reliable.

That of SNMP is a subject that attracts me too, but I have the impression that one has got lots of things to do on one's own, short of binding with some C library reputed as reliable, but that's much less attractive.

ADB – Ada 95 Object Database Server

From: Michael Erdmann <merdmann@users.sourceforge.net>
Date: Fri, 23 Dec 2005 17:34:07 +0100
Organization: http://gnade.sourceforge.net
Subject: Announce: Release of Small Ada 95 Object Database Server Version 0.1.0
Newsgroups: comp.lang.ada

The first Alfa release version 0.1.0 of the ADB project is available for download.

The ADB project provides simple object database server including the libraries for client application development.

The current version is developed on a SuSe 9.0 Linux distribution but it is expected to compile with GNAT 3.15p on other Linux versions as well.

Download instructions:

http://sourceforge.net/project/showfiles.php?group_id=23045.

Select: oos-src 0.1.0

After downloading, simply unpack the release and run make.

The documentation is available at: http://gnade.sourceforge.net/adb

Unfortunately the documentation is incomplete but it will be updated constantly.

Comments are welcome!

[See also "GNADE 1.5.3a – GNAT Ada 95 Database Development Environment" in AUJ 25-3 (Sep 2004), p.123. -- su]

PCHIF – Proof Checker Interface for SPARK

From: JP Thornley <jpt@diphi.demon.co.uk>
Date: Tue, 28 Feb 2006 14:27:56 +0000
Subject: ANN: New version of Proof Checker Interface for SPARK
Newsgroups: comp.lang.ada

The latest version of the SPARK toolset (Version 7.3) has removed the problem that limited the capability of the first version of the Proof Checker Interface (PCHIF).

With version 2.6 of the Proof Checker (included in SPARK Version 7.3) it is now possible to handle both input to and output from the Proof Checker in the interface.

The download page for the new version of the interface can be reached from www.sparksure.com.

If you already have the earlier version then note that the new version has been developed with GtkAda 2.4.0 (the earlier version used 2.2.0).

The downloads include the VC_View tool that is unchanged from the earlier version.

[See also "VC_View and PCHIF – SPARK Proof Tools" in AUJ 26-3 (Sep 2005), p.154. -- su]

L4DA/Lovelace – Ada based Operating System

From: askliepios <askliepios@yahoo.com>
Date: 30 Jan 2006 02:53:34 -0800
Subject: Yet another operating system
Newsgroups: alt.os.development,comp.lang.ada

I plan to write with a few friends a new operating system called Lovelace, an Unix Ada based operating system on top of a L4 pistachio micro-kernel.

There is a (little) web page about it, we are looking for people to help us.

http://lovelace.rochebrune.org

By now we manage to boot some Ada code in Bochs with GRUB (and other real computers too :-p), and we can catch Ada exceptions in the user address space. We have a little memory support and we started a file system and the thread support. Our main goal is to provide :

- A complete Ada framework to develop various OS on top of L4 (in fact by rewrite the Ada GNAT run time lib on top of L4)

- Provide various L4 servers written in Ada with our framework to make a real Unix operating system.

From: Eduardo Zambon <zambon@inf.ufes.br>
Date: Wed, 01 Feb 2006 20:09:38 -0200
Organization: UFES - DI
Subject: Re: Yet another operating system
Newsgroups: comp.lang.ada

Really interesting project. Since AdaOS seems to be stalled, this one is a nice replacement.

The funny thing is that we have an OS research group at my University (UFES – Brazil) and one of our projects is very closely related to Lovelace. We call it L4DA, and as one may guess is an implementation of L4 in Ada. Currently, there is no web page for L4DA since it's still in an early stage of development and is the topic of my master thesis. We hope that a first version will be completed until the beginning of second semester and by then the code will be released under a GPL-like license.

Soon we'll be replacing GNU/Linux with Lovelace/L4DA :) Or better yet: L4DA/Lovelace :))

From: askliepios <askliepios@yahoo.com>
Newsgroups: comp.lang.ada
Subject: Re: Yet another operating system
Date: 2 Feb 2006 01:37:20 -0800

Really interested. If you achieve this project, I will be really happy to implement Lovelace on top of L4da. Please keep me informed about this project.

Maybe we could join our effort in order to implement a standard L4 lib with Ada.

From: Nick Roberts <nick.roberts@acm.org>
Date: Tue, 07 Feb 2006 19:43:20 GMT
Subject: Yet another operating system
Newsgroups: comp.lang.ada

I'd like to comment that, although the AdaOS project is stalled, it is not entirely dead, and I hope to bring it back to full vigor at some point in the not-too-distant future.

I'd also like to offer the use of the AdaOS web site: http://www.adaos.net to anyone who is starting a related project, if it would be any help.

There is also currently a discussion forum at: http://adaos.multiply.com

You are welcome to start a new topic there.

[See also "The AdaOS Project" in AUJ 23-4 (Dec 2002), p.199-200. -- su]

Mine Detector 5.0

From: PragmAda Software Engineering <pragmada@earthlink.net>
Date: Tue, 31 Jan 2006 19:11:27 GMT
Subject: ANN: New Source Version of Mine Detector
Newsgroups: comp.lang.ada

Mine Detector 5.0 is available as source only. This version works with GtkAda 2.2 or later and adds user-selectable levels to the game. At the higher levels, guessing may be required to win.

You may download the source for Mine Detector 5.0 from: http://home.earthlink.net/~jrcarter010/mindet.html

If you'd like to receive these notifications directly by e-mail, send an e-mail to: pragmada@earthlink.net

[See also "Mine Detector Game 4.4" in AUJ 25-4 (Dec 2004), p.190. -- su]

Ada-related Products

AdaCore – GPS 3.1

http://www.adacore.com/2006/01/10/adacore-launches-new-version-of-its-market-leading-ada-integrated-development-environment/

Tuesday January 10, 2006

AdaCore Launches New Version of its Ada Integrated Development Environment

AdaCore today introduced the latest and most versatile version of its GNAT Programming Studio (GPS) product, a sophisticated software development environment for the Ada programming language.

Incorporating a significant number of new features, this new version delivers improved usability and more powerful source navigation. It is available on the latest 64 bit GNU/Linux-based platforms, including those from SGI, HP and Intel.

Productivity-increasing improvements include a more user-friendly location view, enhanced tool tips, code completion, and new project editing capabilities. Human interface improvements include better layout of graphical information, and the ability to export using the Scalable Vector Graphics format.

"With this new version, GPS continues to set the pace as the industry's most advanced Ada development environment," said Arnaud Charlet, GPS Project Manager at AdaCore. "Many features are based on suggestions from customers, resulting in a practical tool that can be used to develop, manage and maintain even the largest and most complex systems."

"Our GNAT Programming Studio has been a success since its inception," added Robert Dewar, AdaCore's President and CEO. "Its intuitive interface, tailorability and extensibility make it an essential tool for the professional Ada programmer. With the enhancements offered in the latest release, GPS remains the Integrated Development Environment of choice for Ada."

GPS offers advanced features such as multi-language support (including Ada, C, and C++) and is available on a wide range of host environments for both native and cross-development, including Unix, Windows and GNU/Linux. An intuitive, unified visual interface, identical across all platforms, serves as a control panel to access tools from AdaCore's GNAT Pro Ada development environment as well as from third parties, easing both development and maintenance. As a result, GPS is particularly suited for large, complex systems requiring tool chain integration, ease of use, user customization, and code navigation/analysis.

This latest version of GPS provides many new improvements, including:

* New availability on IA-64 SGI Altix, IA-64 HP Linux, IA-64 HP-UX, x86-64 GNU/Linux platforms
* New cross-reference queries
* Improved plug-in capabilities and python extensions
* Refactoring (rename entity, named parameter associations)
* More efficient and user-friendly locations view
* Improved assembly view
* Persistent bookmarks
* Version Control System activities (group commit)
* Enhanced tooltips and code completion
* Improved graphs (better layout, ability to export in SVG format)
* New call graph tree
* Project Editor enhancements

About GPS

GPS is a powerful Integrated Development Environment (IDE) written in Ada, based on the GtkAda toolkit. GPS' extensive source-code navigation and analysis tools can generate a broad range of useful information, including call graphs, source dependencies, project organization, and complexity metrics. It also provides support for configuration management through an interface to third-party Version Control Systems, and supports a variety of platforms, including Alpha Tru64, Altix Linux, MIPS-IRIX, PA-RISC HP-UX, SPARC Solaris, x86 GNU/Linux, x86 Solaris, and x86 Windows. GPS is highly extensible; a simple scripting approach enables additional tool integration. It is also tailorable, allowing programmers to specialize various aspects of the program's appearance in the editor for a user-specified look and feel.

Pricing and Availability

GPS 3.1 is available to GNAT Pro customers on selected platforms starting December 14. GPS is included with the GNAT Pro Ada Development Environment. Please contact AdaCore for the latest information on pricing and supported configurations. (sales@adacore.com)

http://www.adacore.com/2006/01/23/gps-31/

AdaCore is pleased to announce the immediate release of GPS 3.1.0 for the following platforms:

* alpha-tru64
* ia64-sgi_Linux
* ia64-hp_Linux
* ia64-hpux
* mips-irix
* pa-hpux
* ppc-darwin
* SPARC-solaris
* x86-Linux
* x86-solaris
* x86-windows
* x86_64-Linux

The 3.1.0 version is a major release and provides many new improvements, including:

* New availability on ia64-sgi_Linux, ia64-hp_Linux, ia64-hpux, x86_64-Linux
* New cross-reference queries
* Improved plug-in capabilities and python extensions
* Refactoring (rename entity, name parameters, ...)
* More efficient and user-friendly locations view
* Improved assembly view
* Persistent bookmarks
* VCS activities (group commit)
* Enhanced tooltips and code completion
* Improved graphs (better layout, ability to export in SVG format)
* New call graph tree
* Project Editor enhancements (extending projects, …)

GPS 3.1 is compatible with all versions of GNAT Pro from 3.15 through to 5.04

[See also "AdaCore – GPS 3.0" in AUJ 26-2 (Jun 2005), p.77. -- su]

AdaCore – Thales Group adopts GNAT Pro

http://www.adacore.com/2006/01/23/thales-group-adopts-adacore-as-a-corporate-ada-standard/

Monday January 23, 2006

Thales Group Adopts AdaCore as a corporate Ada Standard

Signs three year global agreement to benefit from flexible software licensing terms

International electronics and systems group Thales, have announced a global software licensing agreement with AdaCore. This will provide flexible, cost-effective, access to AdaCore's GNAT Pro development environment for developers across the 6 businesses in the worldwide Thales Group and its subcontractors.

The three year corporate licence covers a minimum of 250 Thales developers, with additional licences available to be added flexibly in packs of five by both Thales Group companies and their subcontractors. The agreement means that Thales will adopt GNAT Pro as a corporate standard.

The Ada programming language is designed specifically for large, long-lived applications where reliability, efficiency and safety are vital. The latest version of the language, Ada 2005, was ratified earlier this year. AdaCore has been closely involved with the Ada language since its inception and its GNAT Pro development environment combines market leading technology, including Ada 2005, with an expert support system to provide a natural solution where efficient and reliable code is critical.

Close to 300 Thales developers are currently using AdaCore around the globe. Projects using Ada cover all of Thales markets, including naval, air systems, aerospace and land systems. They range from the ARH Tiger Helicopter Simulator (France/Australia), Thales Raytheon Systems Air Command and Control System (USA), avionics systems for the Airbus A400M (France) and the Combat Management System for the French Navy.

"The advent of Ada 2005 further strengthens the already impressive capabilities of the Ada language for mission-critical defence and avionics projects," said Jean-Michel Tanneau, Thales Group. "Given our use of Ada our new agreement with AdaCore is ideal, providing us with cost-effective and flexible access to the leading Ada development environment for our staff across the world."

"Thales Group's global adoption of AdaCore demonstrates our leadership in the market and the growing use of Ada within complex and safety-critical projects," commented Franco Gasperoni, managing director, AdaCore. "Our innovative licensing arrangement provides Thales and its subcontractors with the flexibility to access our tools and support simply and efficiently while benefiting from volume pricing."

At the heart of GNAT Pro is a full featured multi-language (Ada, C, C++) development environment complete with libraries, bindings and a range of supplementary tools. All its technology combines the flexibility and freedom associated with open source development and the assurance that comes from knowing that all tools go through a rigorous quality assurance process. It is based on GCC technology and is backed by rapid and expert support service.

About Thales

Thales is an international electronics and systems group serving defence, aerospace, security and services markets worldwide. The Group employs 60,000 people throughout the world and generated revenues of 10.3 billion euros in 2004.

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial, open-source software solutions for Ada, a modern programming language designed for large, long-lived applications where reliability, efficiency and safety are absolutely critical. AdaCore's flagship product is GNAT Pro, the commercial-grade open-source Ada development environment, which comes with expert online support and is available on more platforms than any other Ada technology. AdaCore has customers worldwide; see http://www.adacore.com/home/company/customers/ for more information.

Use of Ada and GNAT Pro continues to grow in high-integrity and safety-critical applications, including commercial and defence aircraft avionics, air traffic control, railroad systems, financial services and medical devices. AdaCore has North American headquarters in New York and European headquarters in Paris. www.adacore.com

AdaCore – GNAT Pro 5.04

http://www.adacore.com/2006/02/15/adacore-delivers-most-advanced-ada-2005-development-environment/

Wednesday February 15, 2006

AdaCore Delivers Most Advanced Ada 2005 Development Environment

GNAT Pro 5.04 enables more efficient creation of dependable software

AdaCore today launched the latest and most advanced version of its flagship GNAT Pro open-source Ada development environment, GNAT Pro 5.04. Enabling faster creation of robust, dependable software, it supports all the major new features in the Ada 2005 release of the Ada programming language, with over 120 enhancements to the technology.

Created under the auspices of the International Organization for Standardization (ISO), Ada 2005 introduces significant enhancements in many areas, including Object-Oriented Programming, interfacing with other languages (most notably Java), software architectural design, real-time systems, and predefined libraries. It offers improved support for high-integrity applications, including the standardization of the Ravenscar profile for certifiable[BMB1] concurrent programs. Ada 2005 represents the first major upgrade of the Ada language in 10 years.

GNAT Pro 5.04 incorporates improved installation, easier usage, and new features, including options for stack usage analysis and a tool for enforcing project-specific rules. It is implemented on more than 30 configurations, the widest variety in the Ada industry, including new 64-bit platforms, such as SGI's Altix servers, HP's Integrity servers, and the x86-64.

Over half of the new features in GNAT Pro stem from customer requests, demonstrating the effectiveness of AdaCore's unique support model, which ensures customer queries are answered by the product developers themselves, the largest and most experienced group of Ada experts in the world. "With its many enhancements, Ada 2005 is the best choice for reliable and efficient software, across a wide spectrum of applications, including high-integrity systems," commented Cyrille Comar, managing director, AdaCore. "The latest release of GNAT Pro extends these benefits to programmers, enabling faster development of safe and robust code."

"Ada 2005 truly advances the state of the art in programming language design," added Robert Dewar, AdaCore's CEO. "As one example, its unification of concurrency and object oriented technology is a breakthrough that can help programmers develop more maintainable systems. AdaCore's GNAT Pro 5.04 brings these benefits to the industry now, backed by the quality support and high-caliber expertise that we have been providing to our customers since our company was founded."

GNAT Pro 5.04 includes advanced AltiVec support, both direct to PowerPC and simulated to other compatible targets; greater stack size control and analysis; and efficient, linker-level removal of unused subprograms and data. The new ASIS-based GNATCHECK tool provides evidence of the enforcement of project-specific rules.

About GNAT Pro

GNAT Pro is a robust and flexible Ada development environment based on the

GNU GCC compiler technology. It comprises a full Ada compiler, an Integrated Development Environment (GPS, the GNAT Programming Studio), a comprehensive toolset including a visual debugger, and a set of supplemental libraries and bindings. It is distributed with complete source code, and is backed by rapid and expert support service.

http://www.adacore.com/2006/01/30/gnat-pro-504a/

AdaCore is pleased to announce the immediate availability of the GNAT Pro 5.04 release.

GNAT Pro 5.04 sees important enhancements in many areas, including:

* Support for all major 64-bit architectures
* Increased support for OS versions
* Altivec support
* Improved installation and usage of the toolset
* Stack size control and analysis
* Linker-level removal of unused subprograms (on Linux only so far)
* GNATCHECK (The new coding standard verification tool)
* Support for all major features of Ada 2005 including:
    o "Limited with" and "private with"
    o All new forms of anonymous access types
    o Complete interface feature (including task, protected, synchronized, and limited interfaces)
    o "Object.Operation" notation
    o Complete containers library

[See also "AdaCore – GNAT Pro 5.03a" in AUJ 26-1 (Mar 2005), p.13-14. -- su]

AdaCore – PolyORB 2.0

http://www.adacore.com/2006/03/14/polyorb-20/

PolyORB 2.0 - March 14, 2006

The recent release of PolyORB 2.0 brings increased versatility to our generic middleware technology.

A comprehensive architecture review and strategic reorganization resulted in a clear isolation of the essential controlling logic of the core distribution library. This allowed us to build and verify formal models of the internal components of PolyORB, providing increased confidence in the code.

New scheduling policies are also supported, allowing better adaptation to specific application requirements. Profiling was performed on the distribution runtime. This allowed us to identify and remove performance bottlenecks.

In addition to the extensive interoperability features of previous releases, this latest version makes PolyORB suitable for safe and secure software development in distributed applications.

PolyORB 2.0 is available for the following platforms:

* sparc-solaris
* x86-Linux
* pa-hpux

[See also "ACT – PolyORB 1.0p" in AUJ 25-1 (Mar 2004), p.10-11. -- su]

Aivosto – Visustin v3.1 connects with Project Analyzer

http://www.aivosto.com/visustin.html

February 2006

Visustin v3 Flow chart generator

Added support to flowchart from Project Analyzer v8.

Visualize your code with flow charts. Open up your source file and Visustin shows its execution flow -be it in Visual Basic, VB.NET, VBA, ASP, C/C++, C#, Java, JSP, JavaScript, COBOL, Fortran, Pascal/Delphi, Perl, PHP, T-SQL, PL/SQL or Ada.

Understand existing code. Review algorithms. Verify correctness of program logic. Document complex procedures. Restructure incomprehensible code.

No matter what kind of code you need to document, Visustin can reverse engineer its underlying structure. If, goto, for and while statements, even with and try..catch..finally blocks are visualized in an easy-to-understand format. No new languages to learn – your existing code is all you need. If you see a real complex case, print it out as a mosaic and hang it on your wall.

Automated layout. Visustin creates an optimal visual layout automatically. Just hit one key and you're done – no need to adjust the charts.

All code with comments. Visustin flow charts include all of your code, optionally the comments as well. Create large master charts or small charts with just the important logic.

Multi-page print. Preview and print large flow charts on multiple pages, or squeeze to fit on one sheet.

Save graphs. Use flow charts in your project documentation in GIF, BMP, JPG, PNG, WMF, EMF, PS or DOT image format.

Web publication. Save flow charts as web pages or MHT web archives.

Visio export [Pro Edition] Export your flow charts to edit in Microsoft Visio 2002/2003. Save your drawing efforts by converting your code to Visio format. Visio exportPopup link

Bulk charting [Pro Edition] Save all your source files as flow charts in one run. Also exports as Visio .vsd files.

[See also "Aivosto – Visustin v3 flowcharts Ada code" in AUJ 26-1 (Mar 2005), p.14-15. -- su]

Aonix – ObjectAda 8.2 Available for VxWorks/PowerPC Platform

http://www.aonix.com/pr_12.19.05.html

Latest Aonix ObjectAda Release Now Available for VxWorks/PowerPC Platform

Wind River RTOS developers demand new ObjectAda 8.2 capabilities

San Diego, CA, Paris, France, December 19, 2005

Aonix, a provider of solutions for safety- and mission-critical applications, released its latest version of ObjectAda Windows for the PowerPC/VxWorks development environment. The ObjectAda 8.2 port brings significantly enhanced compiler and debug technology to Wind River's Tornado 2.x and VxWorks 5.x environments and boasts an Ada/VxWorks binding that interfaces Ada constructs with VxWorks primitives in the same application.

The Windows-hosted ObjectAda 8.2 improves the underlying Aonix Ada 95 compiler, reducing compile time for the PowerPC/VxWorks platform. As well, ObjectAda 8.2 includes a newly developed capability to attach the symbolic debugger to a running Ada application, which aids VxWorks developers in resolving programming errors discovered after the test and debug phase is completed. Integration with the VxWorks environment is seamless thanks to a specifically developed VxWorks binding that enables Ada tasks and VxWorks tasks to be combined in the same application.

"Impressed with the overall improvements in the ObjectAda 8.2 release, many PowerPC/VxWorks users have requested these capabilities on their platform," noted Jacques Brygier, VP Marketing of Aonix. "Our customers appreciate the quality of the real-time support we provide through our Ada implementation. Having this available on the VxWorks/PowerPC platform is a real benefit to them as they can take advantage of the performance and productivity capabilities offered by combining the two well-proven technologies."

ObjectAda 8.2 Windows cross PowerPC/VxWorks is available under the CorePack packaging that includes an Ada 95 compiler, Ada 95 optimizer, partial annex C support, partial annex D support, syntactic editor, graphical and command line interfaces, library configuration tool, program builder, source browsing engine, source registration tool, source un-registration tool, source code reference tool, symbolic debugger, and graphical installer. Online documents in PDF format and sample programs provide the developer with immediate development assistance.

Eclipse integration and compatibility with VxWorks 6.x is expected in the beginning of 2006.

Shipping and Availability

ObjectAda Windows cross PowerPC/VxWorks is available immediately for Windows 2000 and XP host platforms and supports Tornado 2.x and VxWorks 5.x running on all PowerPC boards supported by VxWorks. For more information about this product, please visit: www.aonix.com/objectada.html.

About Aonix

Aonix offers mission- and safety-critical solutions primarily to the military and aerospace, telecommunications and transportation-related industries. Aonix delivers the leading high-reliability, real-time embedded virtual machine solution for running Java programs deployed today and has the largest number of certified Ada applications at the highest level of criticality. Our unique modeling solution features UML 2.0 profiles and MDA tailored for the mission- and safety-critical space. Aonix products include PERCS, RAVEN, and Ameos. Headquartered in San Diego, CA and Paris, France, Aonix operates sales offices throughout North America and Europe in addition to offering a network of international distributors. For more information, visit www.aonix.com.

Aonix – ObjectAda 8.2 Available for LynxOS/PowerPC Platform

http://www.aonix.com/pr_02.14.06a.html

Latest Aonix ObjectAda Release Now Available for LynxOS/PowerPC Platforms

Ada LynuxWorks developers benefit from new ObjectAda 8.2 capabilities

Embedded World, Nurernberg, Germany, February 14, 2006

Aonix, a provider of solutions for safety- and mission-critical applications, released its latest versions of ObjectAda Linux for PowerPC/LynxOS and ObjectAda Solaris for PowerPC/LynxOS development environments. The ObjectAda 8.2 release brings significantly enhanced compiler and debug technology to LynuxWorks' LynxOS 4.x environments, enabling developers to mix Ada and C within the same application by using the POSIX1.c API.

The Linux- and Solaris-hosted ObjectAda 8.2 environments improve the underlying Aonix Ada 95 compiler, reducing compile time for the PowerPC/LynxOS platform. In ObjectAda 8.2, the symbolic debugger can be attached to a running Ada application. This feature aids LynxOS developers in resolving programming errors discovered after the test and debug phase is completed.

"Our customers have come to rely on the robust efficiency of the combined Aonix and LynuxWorks offering for more than a decade," noted Jacques Brygier, VP Marketing of Aonix. "They can now take full advantage of ObjectAda 8.2 and the rich set of functionality and real-time capabilities offered by LynxOS."

ObjectAda 8.2 Linux and Solaris development platform targeting the PowerPC/LynxOS embedded platforms are available under the CorePack packaging. CorePack includes an Ada 95 compiler, Ada 95 optimizer, partial annex C support, partial annex D support, syntactic editor, and both graphical and command line interfaces. Other parts of the CorePack toolchain consist of a library configuration tool, program builder, source browsing engine, source registration and unregistration tools, source code reference tool, symbolic debugger, and graphical installer. Online documents in PDF format and sample programs provide the developer with immediate development assistance.

In addition to the CorePack packaging, both environments offer additional optional components. One of these components is Ada-ASSURED, an advanced editor that provides additional language-sensitive features and style-guideline conformance checking.

Aonix ADT (Ada plug-in into Eclipse) and compatibility with Luminosity is expected in Q2 2006.

Shipping and Availability

ObjectAda Linux cross PowerPC/LynxOS is available immediately for Linux RedHat Enterprise 4.0 or compliant host platforms and supports LynxOS 4.x running on all PowerPC boards supported by LynxOS. For more information about this product, please visit www.aonix.com/objectada.html.

Aonix – ObjectAda 8.2 for Windows Update

From: Owner-Intel-ObjectAda <owner-intel-objectada@aonix.com>
Date: Fri, 27 Jan 2006 10:56:31 -0800
Subject: Intel-OA: New ObjectAda 8.2 Update
To: intel-objectada@aonix.com

A new update for Aonix ObjectAda for Windows 8.2, 1102V82-U1, is now available at http://www.aonix.com/ada_patches.html.

Please see the Release Notes for further details on the corrections made and installation instructions. The release notes can be viewed at ftp://ftp.aonix.com/pub/adats/outgoing/1102/8.2/U1/1102V82-U1.Release_Notes.

Downloading ObjectAda updates requires a password which can be obtained from your local Aonix Customer Support department. Please note that a current maintenance agreement is required to obtain the password.

For information on obtaining or renewing a maintenance agreement, please contact your nearest Aonix Sales office. For contact information see http://www.aonix.com/contact_us.html.

[See also "Aonix – ObjectAda 8.2 for Windows" in AUJ 26-4 (Dec 2005), p.242. -- su]

DDC-I – Opens New US Sales Office for the Eastern Region

http://www.ddci.com/display_news_item.php?filename=news_ddci_opens_eastern_region_sales_office.php

DDC-I, Inc. Opens New US Sales Office for the Eastern Region

March 15, 2006 – Phoenix, AZ – DDC-I, a global leader in safety and security critical software development tools for embedded applications announced today the opening of a new U.S. Eastern region sales office and the appointment of Rich Ciccotto to the growing sales team.

In order to better support significant growth, DDC-I has been increasing staff in the sales and engineering departments, with engineering expanding by 25% this year. Ciccotto's appointment puts an experienced industry veteran in the territory to continue the customer responsiveness that DDC-I is recognized for.

Ciccotto has over 24 years of new business development, customer service, training, technical sales and account management experience. A motivational leader, Ciccotto has achieved 23 "Presidents Club" awards in recognition of his accomplishments.

"Rich is a true professional with stellar customer references, exactly the kind of person we want working closely with our customers," said Bob Morris, President & CEO of DDC-I, Inc. "He brings a wealth of experience and motivation that fits nicely with the rest of our team."

Ciccotto joins DDC-I from AONIX, where he served for over 7 years as the Southeast and Midwest Sales Manager. Prior to AONIX, he worked for RTM/Integrated Chipware, Magic Software Enterprises & Computer Associates. As DDC-I's Eastern Sales Manager, Ciccotto will work out of a new east coast sales office located Viera, Florida, which will allow for easy access to his east coast accounts.

About DDC-I, Inc.

DDC-I, Inc. is a global supplier of software development tools, custom software development services, and legacy software system modernization. DDC-I's customer base is an impressive "who's who" in the commercial, military, aerospace, and safety-critical industries. Tools include compiler systems and run-time systems for C, Embedded C++, Ada, JOVIAL and Fortran application development. For more information regarding DDC-I products, contact DDC-I at: 400 North Fifth Street, Phoenix, Arizona 85004; phone (602) 275-7172; fax (602) 252-6054; e-mail sales@ddci.com or visit www.ddci.com

Green Hills – INTEGRITY for Radiation-Hardened RAD750

http://www.ghs.com/news/20060117_rad750.html

Green Hills Software Announces INTEGRITY Support for BAE Systems RAD750

Brings Proven Mission-Critical Real-Time Operating System to Premier Radiation-Hardened Computer for Space Applications

Santa Barbara, CA, January 17 , 2006

Green Hills Software, Inc., the leader in real-time operating systems (RTOS) and device software optimization (DSO), today announced the immediate availability of a complete port of its INTEGRITY real-time operating system to the BAE Systems' RAD750 radiation hardened PowerPC Processor and CompactPCI single board computer. The combination of INTEGRITY and RAD750 yields the industry's most advanced hardware/software architecture for high reliability space systems.

"We selected INTEGRITY because of its proven heritage in mission critical systems as well as its integration with Green Hills Software's powerful MULTI IDE tool set," commented Dave Stofko, Flight Systems Software Manager, Space Systems/Loral. "Coupled with the latest radiation-hardened hardware, this solution represents the state-of- the art in space-based device software platforms."

INTEGRITY RTOS

INTEGRITY, the premier real-time operating system for use in mission critical systems, has been selected for use in space-based systems such as satellites as well as a wide variety of aerospace applications. According to leading industry analysts, INTEGRITY has demonstrated the highest growth in real-time operating system market share for the past 4 years. INTEGRITY's success is due to its advanced technological design, incorporating memory protection, guaranteed resource availability, field upgradeability, optimal real-time response, and the world's leading development tools integration with Green Hills Software's MULTI IDE. In addition, INTEGRITY is the only commercially developed real-time operating system to be certified with integrated modular avionics (IMA) systems by the US Federal Aviation Administration (FAA) to the stringent DO-178B Level A, the highest level of safety in which a system failure may be catastrophic. In comparison, legacy real-time operating systems running on radiation-hardened computers have not taken advantage of the processor's memory management unit (MMU) at all, making the system susceptible to unforeseen interactions between software components, including memory corruptions, faults and denial of service problems.

RAD750 CompactPCI Board Support Package (BSP)

INTEGRITY's comprehensive device driver and debugging support for the

RAD750-based single board computer includes:

* Serial/UART
* Ethernet
* Hardware cache snooping
* High resolution timer
* Watchdog timer
* RAM initialization and booting from EEPROM
* JTAG debugging with the Green Hills probe
* Enhanced power PCI bridge, including PCI enumeration, timers, DMA, EMC access, a hardware semaphore API, and support for power saving modes

A wide range of INTEGRITY middleware available for the BAE space computer includes:

* IPv4 and IPv6 TCP/IP stacks
* Network applications and security (FTP/TFTP, DHCP, DNS, SSL, SSH, Crypto, Firewall)
* DOS/FAT, RAM and Fast File System (FFS) support
* Partition Journaling File System (PJFS)

Reconfigurable Space Systems

Next generation space-based systems will be characterized by complex missions, many of which have field lifetimes measured in years. As such, these systems require an operating system that can meet the highest levels of reliability and security while enabling in-space reconfiguration. INTEGRITY's microkernel design, virtual device drivers, and partitioning architecture enable designers to build truly reliable space-based systems wherein any part of the software, including application programs, RTOS middleware, and even the kernel and interrupt service routines, can be patched, replaced, or upgraded. "This kind of flexibility is a requirement in space due to the harsh environment and likelihood of an SEU (Single Event Upset) that can cause portions of memory to fail. Mission controllers can take advantage of INTEGRITY's field upgrade capabilities to work around hardware problems and extend the life and value of their space-based investments", commented David Kleidermacher, Vice President of Engineering at Green Hills Software.

PJFS

Green Hills Software's Partitioning Journaling File System (PJFS) is a natural partner with INTEGRITY for space-based systems and is also now available for the RAD750 space computer. PJFS employs complete file data and metadata journaling, ensuring that the file system and file data cannot be lost in the event of unexpected power loss. In addition, PJFS employs a patent-pending partitioning architecture that enables applications to have guaranteed media resources and access control, preventing unintended resource exhaustion or other failures that can arise when sharing a traditional file system with complex software. Finally, like INTEGRITY, PJFS sports a very small footprint that is ideal for resource-constrained systems.

About the RAD750

The RAD750 from BAE Systems is the most technologically advanced microprocessor ever offered to the space community. The RAD750 is a licensed radiation hardened version of the IBM PowerPC 750. The RAD750 is a 3rd generation microprocessor, with almost ten times the performance of current space processors, and is the follow-on to BAE System's highly successful and space proven RAD6000 family. The RAD750 is available in a single board computer in the CompactPCI form factor. The RAD750 architecture supports an industry leading performance of 260 MIPS operating at 132 MHz.

Green Hills – Model-Driven Development for Safety-Critical Embedded Software

http://www.ghs.com/news/20060214_esterel.html

Green Hills Software and Esterel Technologies Partner to Create the First Complete Model-Driven Solution for Safety-Critical Embedded Software Development

First Integration of a DO-178B Level A and IEC 61508 SIL3 Compliant Modeling, Code Generation, Verified Compilers and RTOS Solution

SANTA BARBARA, CA and ELANCOURT, France – February 14, 2006

Green Hills Software, Inc., the leader in real-time operating systems (RTOS) and device software optimization (DSO), and Esterel Technologies, a leading worldwide supplier of model-based design, validation and code generation tools for safety-critical embedded software applications, announced today a strategic partnership resulting in the first complete Model-Driven Solution (MDD) for safety-critical embedded software code generation and product development.

"By partnering with Esterel Technologies, Green Hills Software is the first company to provide a highly integrated Model-Driven Solution for safety-critical embedded systems developers, based on the widely established DO-178B, IEC 61508, and SCADE standards," said Dan O'Dowd, founder and chief executive officer of Green Hills Software. "SCADE provides the most comprehensive Model-Driven Development solution available for the safety-critical embedded markets today. The combination of SCADE with our MULTI IDE development solution, royalty-free operating systems and target middleware gives our customers a comprehensive and safe solution for optimizing the time-to-market, time-to-certification and total reliability of safety-critical embedded systems."

Under the agreement, Esterel Technologies' SCADE Qualified Code Generator (KCG) will produce code that will be automatically integrated with Green Hills Software's INTEGRITY-178B Level A and IEC 61508 certified INTEGRITY RTOS's. Furthermore, Green Hills Software's industry leading compilers will be pre-qualified through SCADE's Compiler Verification Kit (CVK) ensuring that any code produced by SCADE and then compiled by Green Hills Software's compilers will go successfully to certification in a cost-effective, timely manner. SCADE KCG and CVK will be integrated with Green Hills Software's technology-leading INTEGRITY RTOS and C/C++ / Ada compilers to enable a seamless workflow between modeling and implementation. The two companies will also collaborate on future integrated features and capabilities, including the integration of the SCADE built-in simulator with MULTI.

This integration will create the first DO-178B Level A and IEC 61508 SIL3-compliant end-to-end solution, spanning software modeling, code generation, compilation, and RTOS integration. Green Hills Software's and Esterel Technologies' products are also integrated with leading UML/SysML modeling tools such as I-Logix' RHAPSODY to support legacy and non-critical code modeling and reverse engineering.

Traditionally, software designers and developers have used separate environments for different development aspects: one for application software modeling, often paper-based, and another for implementation to target. In contrast, the integrated solution developed by Green Hills Software and Esterel Technologies accelerates time-to-market and time-to-certification by generating a DO-178B and IEC 61508-compliant target code in C, directly from the SCADE model.

"Green Hills Software offers the industry's most complete and best technology RTOS and IDE solutions," said Eric Bantegnie, president and chief executive officer of Esterel Technologies. "When these qualities are combined with our market-leading SCADE safety-critical application development environment, our two companies, each the fastest growing and most successful in our respective sectors, provide a truly synergistic solution. This not only benefits our mutual customers, but also reduces the overall cost of development and certifications of DO-178B up to Level A and IEC 61508 SIL3 embedded systems."

Integrated Solution Addresses Critical Development Phases

The combination of SCADE, INTEGRITY and Green Hills Software compilers provides an integrated solution that addresses critical phases of DO-178B and IEC 61508 embedded systems development

* Behavioral design and validation – using SCADE Editor, Simulator and Model Test Coverage

* Code Generation using SCADE's Qualified Code Generator, KCG – removing the need for low-level testing other than the qualification of the User context and Compiler

* Compiling with Green Hills Software compilers – pre-qualified for SCADE generated code compilation thanks to SCADE's Compiler Verification Kit

* Integration with Green Hills Software's market-leading INTEGRITY RTOS – the standard for certified and certifiable RTOS's in the DO-178B Level A and IEC 61508 markets

* Debugging and optimization – The multi-source-level debugger is fully synchronized with the SCADE models

Availability

SCADE KCG and CVK integration with INTEGRITY and Green Hills Software compilers will be available in July 2006.

About Green Hills Software

Founded in 1982, Green Hills Software, Inc. is the technology leader in real-time operating systems (RTOS) and device software optimization (DSO) for 32- and 64-bit embedded systems. Our royalty-free INTEGRITY RTOS, velOSity microkernel, compilers, MULTI and AdaMULTI integrated development environments and TimeMachine debugger offer a complete development solution that addresses both deeply embedded and high-reliability applications. Green Hills Software is headquartered in Santa Barbara, CA, with European headquarters in the United Kingdom. Visit Green Hills Software on the web at www.ghs.com.

About Esterel Technologies

Esterel Technologies' tools create unambiguous specifications that produce correct-by-construction, automated implementation in software and/or hardware. Today, SCADE Suite is the standard for the creation of RTCA DO-178B, EUROCAE ED-12B, and IEC 61508 safety-critical embedded software in the civilian avionics and transportation industries; SCADE Drive is the emerging standard for the creation of safety-critical embedded software in the automotive industry. Esterel Studio allows electronics hardware designers to create golden specification models that can be automatically implemented in RTL or C.

Esterel Technologies is a privately held company with headquarters in Mountain View, California, USA, and Elancourt, France, with direct sales offices in Germany, the United Kingdom, and China. For additional information, visit the Esterel Technologies website at www.esterel-technologies.com.

Praxis HIS – SPARK Toolset 7.3

From: Rod Chapman <rod.chapman@praxis-his.com>
Date: 26 Jan 2006 06:20:09 -0800
Subject: ANN: SPARK 7.3 now available
Newsgroups: comp.lang.ada

Praxis High Integrity Systems are pleased to announce the immediate availability of release 7.3 of the SPARK language and toolset.

Complete details, including the revised language definition and the toolset release note are now available from www.sparkada.com

Supported customers are being upgraded now. Academic users and tool-partners will be upgraded shortly.

Tool upgrade packages for readers of the SPARK textbook are also available from www.sparkada.com

Highlights of this release include:

- VC Generation improvements in the presence of semantic and data-flow errors.
- Support for full-range of IEEE 64-bit floating point values in the configuration file.
- A new Examiner switch that produces

explanations of errors and warnings on-screen and in the listing files.
- Better error messages for common syntax errors.
- Relaxation of the rule requiring qualification of modular literals.
- Support for proof rules involving the 'Size attribute.
- Correct order or declaration in FDL files for type-announced and private types.
- Support for the use of pragma Import to complete an external own variable.
- Significant new Simplifier tactics for modular and rational inequalities.
- Support for user-defined proof rules for the Simplifier.
- Port of the Simplifier and Checker to the SICSTUS PROLOG compiler. Both are significantly faster as a result.

See www.sparkada.com for more details, including performance metrics for the new Simplifier.

Full details of all language and tool changes can be found in the release note.

Buyers of the "SPARK Book" by John Barnes can now download upgrade packages to bring their toolset and documentation up to release 7.3.

So how much better is the new Simplifier?

Well, here are the results of a simple experiment, conducted on the SHOLIS application software. SHOLIS comprises some 27000 lines of embedded, safety-critical, real-time code, so it's a "real-world" example for sure. We generated VCs using Examiner 7.3, then applied the following versions of the Simplifier:

2.17 (POPLOG) – as shipped with toolset release 7.2 (i.e. what you've got right now...)
2.22 (POPLOG) – with new tactics, but using the same compiler.
2.22 (SICSTUS) – as shipped with release 7.3 (i.e. what you'll be getting real-soon-now...)

We ran all three runs on a single 2.4GHz Pentium 4 Xeon machine, running Windows XP. Here are the results:

	2.17 (POPLOG)	2.22 (POPLOG)	2.22 (SICSTUS)
VC total	9685	9685	9685
VC proved	9134	9221	9221
VC un-proved	551	464	464
% Proved	94.31	95.21	95.21
Time	185 mins	144 mins	73 mins

We've just commissioned a new server, which contains a dual-core AMD Opteron64 model 275. This gives identical results in JUST FORTY MINUTES, using sparksimp's "/p=4" option to load both processor cores.

In summary:

1) Simplifier 2.22 is both smarter AND faster than 2.17 even using the same PROLOG compiler. In this example, it discharges an additional 87 VCs automatically – an improvement of 16% in the number of VCs left.

2) SICSTUS prolog nearly doubles the performance of the Simplifier with identical results. Not bad!

[See also "Praxis HIS - SPARK Release 7.2" in AUJ 26-1 (Mar 2005), p.17-18. --su]

Praxis HIS – SPARK Toolset for Mac OS X

From: Rod Chapman <rod.chapman@praxis-his.com>
Date: 15 Feb 2006 07:55:04 -0800
Subject: ANN: SPARK for Apple OS X now available
Newsgroups: comp.lang.ada

I'm pleased to announce that the "book" edition of the SPARK Toolset is now available for Apple Mac OS X.

This release is functionally identical to the release 7.3 already available for Windows and GNU/Linux.

This release includes the demonstration versions of the SPARK Examiner, Simplifier, POGS, SPARKFormat and SPARKMake tools, RavenSPARK examples, and full documentation. We still recommend that you read the SPARK book though!

This release has been compiled and tested on PowerPC/OS X 10.4.4 only at this stage. We haven't tried it on any Intel-based Mac. If anyone out there has an Intel-based Mac, then we'd be fascinated to hear if the SPARK tools run under Rosetta or not...

Downloads from http://www.praxis-his.com/sparkada/sparkbook.asp

Ada and GNU/Linux

Multi-architecture Support

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 15 Mar 2006 00:43:04 +0100
Subject: Re: [gnuada] gcc 4.1.0 available
Newsgroups: comp.lang.ada

> I had tried Debian, knowing that Ludovic Brenta was doing great work on GNAT support in Debian. But I didn't get on very well with Sarge, lacking hardware support and multi-arch. I wasn't sure if I could run the 32-bit applications on it properly.

As far as I can tell, multiarch support is immature in all distributions. Work is ongoing in Debian to provide good multiarch support, but currently we're restricted to biarch support on some architecture pairs (i386-amd64, powerpc-ppc64, and sparc-sparc64). I'm not actually that knowledgeable about biarch myself. The technicalities are already complex enough, but there are policy decisions to be made as well. Apparently, we're looking at generalising the tool chain, libraries, file system hierarchy, dynamic loader, package manager (dpkg), and I've probably forgotten some other things.

That said, have you looked at http://www.debian.org/ports/amd64/ ?

Yes, you can run 32-bit applications on it. In the worst case, you can always create a chroot containing a complete 32-bit userland running on top of a 64-bit system. But as I said, Debian developers are looking for ways to provide that out of the box.

If you want to do Ada on amd64 with Sarge, you need to use gnat-3.4 instead of gnat. In Etch, you get gnat-4.0 instead. I'm now working on providing gnat-4.1, which, when it stabilises, will become the default compiler for Ada 2005, C, C++, Fortran 95, Java, Objective C, and Objective C++.

> SuSE seemed very promising, and your creation of a complete set of GNAT packages makes it quite attractive. Perhaps this is what I should use?

I don't follow SuSE development, but I am under the impression that its otherwise good support for amd64 is uniarch only, i.e. support for 32-bit binaries is immature. Perhaps Martin can confirm or deny.

> As regards Annex E, I am still stuck with the version of Glade that goes with GNAT 3.15p, and there appears to be a nasty bug which I hit occasionally. I had got the impression that Glade development had halted in favour of PolyORB. Which of these should I be using? I guess I should stick with Glade if that is what you have packaged!

You can download recent sources of GLADE from AdaCore's CVS repository[1]. I see there is activity there, the most recent file was modified 6 days ago. The change seems to be related to 64-bit architectures.

[1] https://libre2.adacore.com/cvsweb

It is my intention to take these sources and port them to GCC 4.1.

Ada and Microsoft

MinGW vs. Cygwin

From: midgleyben@hotmail.com
Date: 30 Jan 2006 06:57:12 -0800
Subject: Mingw vs Cygwin
Newsgroups: comp.lang.ada

I have a question about the differences between Cygwin Adan Mingw, with reference to the Ada compiler.

I need to port Ada code to XP from Linux, link C code (maybe C++) to the project and create an exe which supports sockets (networking). So which way Mingw or Cygwin ? I have read so many reports of problems with c under Mingw and just problems with gcc-ada but non particularly up to date, any advice welcome.

Also if I get the whole Cygwin installation I get Mingw too, to my understanding if I use Mingw I statically link support for the windows API and if I use the gcc-ada compiler I need to provide Cygwin1.dll for distribution, is this accurate and can anyone add detail to this explanation?

From: Jeffrey Creem <jeff@thecreems.com>
Date: Mon, 30 Jan 2006 19:51:01 -0500
Subject: Re: Mingw vs Cygwin
Newsgroups: comp.lang.ada

If you go to any "help" newsgroup you will find nothing but problems about a topic so first of all I would not get too worked up about seeing "nothing but problems" with people doing C under Mingw.

Next, you do not need to provide Cygwin1.dll for Ada code that is built from a Mingw based distribution.

The question is, do you need capability that is present in the Cygwin dll or not. The Cygwin dll provides essentially a UNIX compatibility layer. Depending on the nature of your code, you might not need it at all.

If you do think you need it, you need to understand the licensing terms of the Cygwin dll which is essentially (last time I checked) GPL (not LGPL). (Though I think you can buy a license under different terms from RedHat).

If you need to link your code against the standard Cygwin dll then you will need to distribute your application under the terms of the dll (read the GPL for details. Short story, you have to give the source code to everyone you give the binary to and you can't limit who they give the source code to).

The standard AdaCore GNAT build executables that are Mingw based. The Cygwin.dll that is inside of most of the GNAT windows binary distributions is there (I think) to support the GDB install which is Cygwin based.

In general, I would not recommend the Cygwin approach unless it is critical to a successful port.

References to Publications

Programming in Ada 2005 by John Barnes

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 27 Feb 2006 18:18:22 -0600
Subject: Re: Programming in Ada 2005 book
Newsgroups: comp.lang.ada

> I've just ordered my copy.
> John is clearly far too modest to advertise here, so I'll just pass on this link :-)
> http://www.amazon.co.uk/exec/obidos/ASIN/0321340787/ref=br_lf_b_8/203-8725360-1116746

John told me last week that he didn't have the publishing details yet (which is why the Ada 2005 section of http://www.adaic.com/learn/textbook.html is empty). How did you find it out before he did?? :-)

From: Randy Brukardt <randy@rrsoftware.com>
Newsgroups: comp.lang.ada
Subject: Re: Programming in Ada 2005 book
Date: Tue, 28 Feb 2006 16:32:08 -0600

> By the way, Amazon are advertising a 30th June 2006 release date, but the AdaIC site is listing an April 2006 – is that the release date in the USA?

I used the date on the publisher's website (follow the link on the AdaIC page). (That was March 30th). John sent me a cryptic note this morning that seems to imply that date is wrong. Hopefully, I'll find out the real answer.

From: Barbara Barnes <llsbarns@rdg.ac.uk>
Date: Thu, 2 Mar 2006 08:21:07 -0000
Organization: University of Reading
Subject: Re: Programming in Ada 2005 book
Newsgroups: comp.lang.ada

Here is a bit more info.

> It is nearly done (index to be done still). At least the cover has been chosen which is always a huge step forward. It has grown from 702 to 830 pages and that has been kept down by putting some Answers on the CD. The 23 chapters are now 25. Clearly some of it is as in the 95 book but every chapter has been messed with – probably every section has been messed with.
> I am hoping to have copies at Ada-Europe in June.
> Incidentally, looking at the above link to Amazon and then clicking on "Other books by the same author" brings up a whole lot of stuff I didn't write! Pointers are dodgy!
> Also please note that a new printing of the Spark book is just coming out. The book itself is much the same except that a few semicolons have been added. But the key point is that the CD has some new versions of the Spark tools.
> John

Ada Inside

Ada inside the High Speed Train

From: trg <trg@world.std.com>
Date: Tue, 14 Feb 2006 12:14:27 +0100
Subject: Re: Is Ada inside the Bullet Train?
Newsgroups: comp.lang.ada

John McCormick wrote:

> My publisher just sent me a draft of a cover design for the 2nd edition of my data structures textbook (updated to Ada 2005). It includes a photograph of the Bullet Train going through rural Yonezawa, Japan. I've always been under the impression that the Bullet Train had some Ada inside. However, it does not appear on Mike Feldman's Web page listing the commercial uses of Ada. Can anyone confirm that Ada is on board this train?

I don't know about the Japanese train, but your publisher could use a picture of the French TGV, the Eurostar, or the Korean TGV if he wants a picture of a high speed train system that relies on Ada.

Green Hills – Boeing 777

http://www.ghs.com/news/20051214_smith_boeing.html

Smiths Aerospace Selects Green Hills Software for New Boeing 777 Systems

FAA DO-178B Certified Systems Deployed in New Boeing 777-300ER and Being Retrofit into Earlier 777 Models

Santa Barbara, CA – December 14, 2005 – Green Hills Software, Inc., the technology leader in operating systems and development tools for safe and secure systems, today announced that Smiths Aerospace successfully certified two new systems to the FAA's RCTA/DO-178B safety critical standard using Green Hills Software products.

Smiths Aerospace used Green Hills Software's GMART run-time system and AdaMULTI development environment to develop the software for the Electrical Load Management System (ELMS2) and Fuel Quantity Indicating System (FQIS) for the new Boeing 777 300ER aircraft. The software running in both systems has been certified to the Federal Aviation Administration's (FAA) standard for safety-critical software, RTCA/DO-178B. Both the FAA and European Joint Aviation Authority (JAA) were involved in the certification process.

"Smiths Aerospace selected Green Hills Software's AdaMULTI development environment and GMART run-time system for the ELMS2 and FQIS because these systems met our safety requirements and supported the low power and low cost Freescale ColdFire 5307 processor that we were using," said Dave Bolton, principal software engineer at Smiths Aerospace.GMART also reduced our development effort because we were able to use Green Hills Software's certification package, including design and verification data, in our FAA certification submittal."

We are pleased that Smiths Aerospace, like other major avionics manufacturers, is using Green Hills Software's solutions to optimize development of their safety-critical devices," commented Dan O'Dowd, founder and chief executive officer of Green Hills Software. "Avionics manufacturers are increasingly recognizing that they can reduce the time, cost and risk of software development and certification by using our robust and proven off-the-shelf DO-178B solutions. This is why, in addition to the Boeing 777-300ER, Green Hills Software has also been selected for multiple safety-critical systems on both the Boeing 787 and Airbus A380, among others."

About AdaMULTI and GMART

Green Hills Software's AdaMULTI is a complete software development environment for embedded computer-based applications developed using the Ada 95, C, C++ and Embedded C++ (EC++) programming languages. AdaMULTI contains an integrated set of tools that maximize software developers' productivity while enabling them to optimize the reliability, performance and resource requirements of their devices.

Green Hills Software's GMART supports the SPARK safety critical subset of the Ada language. It is ideal for those applications requiring a small fast and deterministic run-time environment. Further SPARK facilitates the development and certification of safety-critical software. GMART has been proven in numerous systems certified to DO-178B, including flight-critical systems that require the most stringent, Level A certification.

Green Hills – Military Trainer Aircraft

http://www.ghs.com/news/20060214_hawk.html

Green Hills Software Kernel and Development Environment Selected for New HAWK Military Trainer Aircraft

Santa Barbara, CA/Nurnberg, Germany February 14 , 2006 – Green Hills Software, Inc., the technology leader in operating systems and development tools for safe and secure systems, today announced that BAE Systems has selected the Green Hills GMART SPARK Ada-compliant kernel and AdaMULTI development environment for the new Hawk military trainer development aircraft.

The GMART kernel for PowerPC is being used for the next-generation Hawk Mission Systems for the aircraft consisting of two new open architecture mission computers. The second unit allows the two cockpits of the aircraft to operate independently, each being used for different purposes to fulfill the required training needs. Instructors in the rear seat can monitor the trainees in the front or configure the system for their own requirements. The second computer also provides extensive back-up capability in the event of any failure. This system provides graphics for all six cockpit display panels and a heads-up display.

"Green Hills Software is very pleased that BAE Systems selected our GMART safety critical kernel and our AdaMULTI development environment for the new Hawk program," said Dan O'Dowd, founder and chief executive officer of Green Hills Software. "Green Hills Software offers several different kernels within our safety critical product line. These were specifically developed to meet individual program needs. The GMART kernel is a SPARK-compliant, small and deterministic kernel that is statically verifiable to be correct."

Green Hills Software offers a complete line of safety critical products. This includes the Green Hills Minimal Ada Run-Time (GMART) product used here by BAE Systems, the Green Hills Small Tasking Ada Run-Time (GSTART) product, the INTEGRITY-178B real-time operating system (RTOS) and all the support tools necessary for safety critical development. GMART and GSTART are both small, fast and deterministic kernels for executing single applications on an embedded computer. INTEGRITY-178B is a time and memory partitioned operating system, certified to DO-178B Level A and to full ARINC 653-1 compliance. Support for ARINC 653-1 with its partitioning definition allows developers to deploy multiple applications on a single processor, at potentially multiple safety certification levels. This powerful capability enables developers to reduce the number of on-board computers needed to support multiple software systems. Further, INTEGRITY-178B is the only safety critical RTOS to be certified for multiple languages, including: Ada, C, MISRA C and Embedded C++, allowing developers to choose the language and kernel best suited to meet their development needs.

Green Hills – FMCDU system

http://www.ghs.com/news/20060307_CMC.html

CMC Electronics Selects Green Hills Software Platform for Avionics

Chosen Solution Includes INTEGRITY-178B RTOS, Ada Ravenscar Kernel and AdaMULTI Development Environment

Santa Barbara, CA – March 7, 2006

Green Hills Software, Inc., the technology leader in operating systems and development tools for safe and secure systems, today announced that CMC Electronics Inc. has selected the Green Hills Software Platform for Avionics, including the INTEGRITY-178B real-time operating system (RTOS), GSTART Ravenscar compliant Ada kernel and AdaMULTI development environment, for CMC's Flight Management Control and Display (FMCDU) system.

"The Green Hills Platform for Avionics provides a single vendor solution that satisfies our advanced development environment requirements," said Patrick Champagne, vice-president engineering of CMC Electronics. "Furthermore, the dual redundant FMCDU system will require certification to the FAA's DO-178B safety critical standard. The Green Hills Software INTEGRITY-178B operating system has been previously proven as certifiable to the highest level for DO-178B."

The Green Hills Platform for Avionics will also be utilized for the development of a CMC Electronics Aircraft Management System product line, featuring PCI open system architecture.

"Green Hills Software is pleased that CMC Electronics has selected our Platform for Avionics with the INTEGRITY-178B RTOS, GSTART kernel and AdaMULTI development environment," said Dan O'Dowd, founder and chief executive officer of Green Hills Software. "Green Hills Software has developed a complete Platform for Avionics to support the diverse needs of our avionics customers. These include a full time and memory partitioned RTOS, Ada language specific kernels, integrated multi language support, all of which have been previously proven as certifiable to the avionics safety critical standard DO-178B Level A. Only the Green Hills Platform for Avionics offers all these capabilities, developed and certified by in-house experts."

The Green Hills Software Platform for Avionics offers a complete line of safety critical products. These include Green Hills Minimal Ada Run-Time (GMART), Green Hills Safe Tasking Ada Run-Time (GSTART), the INTEGRITY-178B real-time operating system (RTOS) and all the support tools necessary for safety critical development. GMART and GSTART are both small, fast and deterministic kernels for executing single applications on an embedded computer. INTEGRITY-178B is a time and memory partitioned operating system, certified to DO-178B Level A with full ARINC-653-1 compliance. Support for ARINC-653-1 with its partitioning definition allows developers to deploy multiple applications on a single processor, at potentially multiple safety certification levels. This powerful capability enables developers to reduce the number of on-board computers needed to support multiple software systems. Furthermore, the Platform for Avionics with INTEGRITY-178B is the only safety critical RTOS certified for multiple languages, including: Ada, C, and Embedded C++, allowing developers to choose the language and kernel best suited to meet their development needs.

About CMC Electronics

CMC Electronics designs and produces leading technology electronics products for the aviation and global positioning markets. CMC's focus is on delivering innovative cockpit systems integration and avionics solutions to its customers worldwide. CMC's principal locations are in Montreal, Quebec; Ottawa, Ontario; and Chicago, Illinois. Formerly known as Canadian Marconi Company, CMC Electronics has designed and built innovative communication and electronics systems since 1903.

Praxis HIS – High-Grade Programmable Cryptographic Engine

http://www.praxis-his.com/sparkada/pdfs/praxis_rockwell_final_pr.pdf

Rockwell Collins selects SPARK Ada for High-Grade Programmable Cryptographic Engine

Bath, England – 28th February 2006

Praxis High Integrity Systems today announced that Rockwell Collins has chosen Praxis' SPARK Ada language and toolset for its Janus high-grade programmable cryptographic engine.

Rockwell Collins will utilize SPARK for development and verification of the Janus security-critical application software. SPARK allows Rockwell Collins to verify security requirements and to provide a quality product with minimal defects in a constrained budget and schedule. SPARK provides the capability to perform a formal mathematical analysis of the control data and user information flow within a virtual machine.

Rod Chapman, SPARK products manager at Praxis, commented, "We're pleased that Rockwell Collins has chosen SPARK and recognized the unique strengths that it brings to the development of ultra-secure software systems. In particular, SPARK prevents large classes of software defect and allows a rigorous and traceable approach to generating the evaluation evidence required by the Common Criteria scheme."

About SPARK

SPARK is a programming language, design approach and toolset designed for the construction and verification of high-integrity software application. The language is an unambiguous, annotated subset of Ada95. The annotations embody "design-by-contract" information in a program that can be cross-checked and verified by the tools. The language is free from all undefined, unspecified or ambiguous constructs and so can be compiled with any standard Ada compiler. This property also enables verification that combines soundness with depth and efficiency. SPARK programs are immune from a wide variety of defects, including data-flow errors and all so-called "runtime errors" including buffer-overflow. SPARK has an enviable track record in meeting the requirements of the most stringent software standards in the world, including UK Def Stan 00-56, DO-178B, CENELEC 50128, and the Common Criteria at the highest assurance and integrity levels. More information about SPARK can be found at www.sparkada.com

About Praxis High Integrity Systems

Praxis High Integrity Systems has developed a global reputation in the fields of high integrity software development, systems engineering, systems safety and security. The Company's roots are in the application of sound engineering principles to the development of high-integrity software systems whether safety, security or business critical. Its unique approaches, tools and products have evolved from practical experience in the most effective approaches to developing such systems. The Company operates in the defence, aerospace, transport, telecommunications, finance and automotive markets. For more information, please visit: www.praxis-his.com.

Ada in Context

Comparing Floating Point Numbers

From: Matthias Kretschmer <mccratch@gmx.net>
Date: Sun, 4 Dec 2005 11:33:32 +0100
Subject: Floating-Point Numbers and Internal Representation
Newsgroups: comp.lang.ada

I had a problem in one of my programs, that was caused by the internal representation of floating-point numbers in the FPU. Concrete: calculating the value of an optimum for some large number of objects, then in a second doing something with all optimal objects. The problem was, that when doing the calculation the second time, the compiler left the floating-point number in the FPU which had a higher precision than the representation I chosen, so comparing for equality returns always "False". The problem would be solved by some operation truncating the floating-point number to the precision I originally wanted or used. I could of course put all the values in an array or list and then finding optimum and optimal objects, but I don't want to go this way. In C if I remember correctly I could use a volatile variable to ensure the compiler will put the value in and read from the variable before comparing, but to achieve something similar (truncating the precision to that of the type used) in Ada?

My current solution is to enhance precision to compiler maximum which seems to be the machine maximum. But I would like to know if there are any better solutions?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 4 Dec 2005 15:50:04 +0100
Subject: Re: Floating-Point Numbers and Internal Representation
Newsgroups: comp.lang.ada

> Independent of CPU/language used I would always suggest to use:
> `abs (X - Y) < epsilon`
> with a sufficient but not to small epsilon instead.

Epsilon above can well be relative that depends solely on the algorithm. Usually epsilon is estimated in the course of calculations together with X and Y, as a part of the algorithm. So the answer is still the same. (:-))

BTW, I wouldn't use division to evaluate relative errors as the paper suggest. Rather:

```
Half_Epsilon * (abs X + abs Y)
```

A really different answer would be interval arithmetic. If X and Y were intervals they would carry the accuracy estimation with them. So one could directly compare them:

```
case X<Y or X>Y is
-- The result is not Boolean!
  when False | Uncertain =>
     -- The difference cannot be
     -- distinguished from
     -- accumulated inaccuracy
  when True =>
     -- They are sufficiently
     --  different
end case;
```

From: Steve <steved94@comcast.net>
Date: Mon, 5 Dec 2005 18:54:15 -0800

Subject: Re: Floating-Point Numbers and Internal Representation
Newsgroups: comp.lang.ada

> Suggested reading: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm
> There is more than one answer to this problem

Interestingly enough I ran across this site a couple of months ago, and am using the AlmostEquals function in some C++ code.

For Ada, I would think you could make use of the 'Adjacent attribute to achieve a similar result, but would be independent of the floating point representation.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 5 Dec 2005 17:38:57 -0600
Subject: Re: Floating-Point Numbers and Internal Representation
Newsgroups: comp.lang.ada

If you're only interested in using the memory precision, you can use the Machine attribute, see A.5.3(60-62). http://www.adaic.com/standards/95lrm/html/RM-A-5-3.html

But, as others have said, that may not be the best solution, as direct comparison of float values for equality is often dubious. It's also relatively expensive on some machines (such as the Intel Pentium processors), where values in registers are always kept in extended precision; dropping that precision usually requires writing the values to memory and back. So it's best to avoid this attribute in performance critical code portions.

From: Gautier de Montmollin <gdemont@hotmail.com>
Date: Sun, 04 Dec 2005 22:29:03 +0100
Subject: Re: Floating-Point Numbers and Internal Representation
Newsgroups: comp.lang.ada

As a complement to other answers about comparing floating point numbers, here is a page explaining the use of appropriate epsilons in Ada according to the circumstances: http://www.adaic.com/docs/95style/html/sec_7/7-2-7.html

Portability among Windows, Linux and MacOS

From: Szymon Guz <alpha@skynet.org.pl>
Date: Thu, 02 Feb 2006 11:53:54 +0100
Subject: Ada & MacOS
Newsgroups: comp.lang.ada

I'm going to develop a small (later it will be bigger as further modules come up) application for small business. It is going to work under Windows, Linux and Mac OS having very similar GUI and using common database (it will be PostgreSQL). I wanted to use C++/wxWindows/PostgreSQL for that but I'd rather go in the Ada direction, so I've got some questions:

1. Is it possible to make such a program in Ada with as less work as possible while creating a GUI for another OS ? I wanted to develop it under windows, and then try to run it under Mac OS without too much work, I want to avoid the situation when I have to create another GUI from the beginning.

2. Is there any compiler for Ada under Mac OS that is compatible with a compiler for Linux and Windows.

3. Does anybody have any experience with using PostgreSQL with Ada on all of these systems (or maybe choose another database) ?

4. How about printing with GtkAda ?

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Fri, 03 Feb 2006 19:38:10 GMT
Subject: Re: Ada & MacOS
Newsgroups: comp.lang.ada

Jean-Pierre Rosen wrote:

> For libraries (and especially GUI libraries like GTK), it is not a matter of compiler, it is a matter of... libraries. Simply choose a library that has been ported to the OSs you want to target.

I haven't checked the latest version, but GtkAda relied on GNAT-specific features.

From: Simon Williams <williams@ntlworld.com>
Date: Fri, 03 Feb 2006 23:55:57 GMT
Subject: Re: Ada & MacOS
Newsgroups: comp.lang.ada

I am the keeper of the macada.org website, and one of the maintainers of the system for Mac OS X. GNAT GCC works fine with Mac OS X. I develop and maintain GUI and server programs for my work in GCC version of GNAT that we build/use/sell for both Linux and Mac OS X. There is no difference in code. We use GtkAda as our GUI, and that works well for us. We have not done a Windows port (though my boss keeps making threats), but in theory it should pretty much just work. I did do a partial port of the server several years ago and it came up and seemed to work. But I didn't have any customers so company wasn't interested in paying to test etc then.

If you have questions on the Mac compatibility on GNAT check out the GNAT for Mac mailing list at: http://hermes.gwu.edu/cgi-bin/wa?SUBED1=gnat-osx&A=1%22

From: Adrian Hoe <abyhoe@gmail.com>
Date: 4 Feb 2006 08:22:40 -0800
Subject: Re: Ada & MacOS
Newsgroups: comp.lang.ada

Visit http://macada.org. This GNAT integrates with Xcode and the Apple's Interface Builder with appropriate plug-in from macada will allow you to develop Apple's native GUI applications. But then, it will not be portable to Windows and Linux. You can overcome this by using Darwin X11 and GTK (GtkAda) to develop your Mac applications.

There is a GTK native port to Mac OS X somewhere (I can't remember the URL but Google should give you some results). If this native port is successful, developing cross-platform GUI will be even more pleasant on Mac.

The Xcode also have nice and convenient integration with Subversion and other SCM. I have just configured my SVN to work with Xcode.

GNAT from macada does not support universal binary yet but soon.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Mon, 06 Feb 2006 09:20:44 +0100
Organization: CERN - European Laboratory for Particle Physics
Subject: Re: Ada & MacOS
Newsgroups: comp.lang.ada

>>OK but how with the compatibility of code compiled on GNAT that is part of GCC between Mac OS, Windows and Linux ?

> It's Ada not C or C++. Ada compilers obey and not ignore the ISO standard. Ada compilers come without a 10 page list of ISO standard features not yet and probably never to be implemented. Ada does not only have an ISO standard – it also has an ISO standard test suite. BTW: The only programming language with an official standard test suite.
> Ada is most likely the most compatible programming language in existence that compiles into binary code.

Of course, not counting "features" of different compilers, right? If you say "Ada compiler", then is GNAT a good example? Doesn't it have any "features" that can affect compilability/behaviour of some code?

The words "compiler bug" appear in comp.lang.ada archive, don't they?

In the same way, C++ guys can say that C++ is an incredibly portable language, with compilers existing for almost every piece of silicon in existence. Well, except of some compiler "features" that spoil the picture, of course.

> That's for theory, in practice: I have used Ada with OS/2, MS-Windows, Linux, OpenVMS and the tendency is: if it runs in one OS it will run on any other as well. Well: unless you use OS specific features or grab deep down into the System packages.

Same for C++. One of the users of my recent code (non-trivial, I would say) compiled it on Mac OS by typing "make", even though I've never touched Mac. So?

Granted, the fact that Ada has a standard test suite is a Very Good Thing, really.

The lack of such test suite for C++ allowed various vendors to put big "C++" letters on whatever shi^H^H^H product they wanted to sell over the last decade or so and that's the cause for the C++ landscape to look so messy today. But don't present it to be entirely hopeless, because it isn't.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Mon, 06 Feb 2006 18:48:44 GMT
Subject: Re: Ada & MacOS
Newsgroups: comp.lang.ada

Using which compilers and what language features? Very few "C++" compilers implement the language defined by the ISO standard. Can you use all the features of the standard language and count on it compiling with all "C++" compilers?

With Ada, the entire standard language is compiled by every compiler; a compiler that didn't implement, say, generics, would be laughed out of existence. Yet "C++" users regularly use compilers that don't implement templates, exceptions, or namespaces.

Sure, Ada compilers are large programs and have errors. They're less common than back in the good old days when it seems I broke a compiler every time I turned around.

Ada compilers can implement compiler-dependent pragmas and attributes, and can supply compiler-dependent packages. Many also supply platform-dependent packages. If you use those, you're not writing portable code. If you stick to the standard language, though, portability is pretty much guaranteed.

GNAT is an interesting beast. Versions exist for a number of platforms, and it comes with a large library (GNAT.*) that is compiler dependent, and much of it (such as GNAT.OS_Lib) seems platform dependent, too. These work fine on all platforms that GNAT compiles to. So you have compiler-dependent but platform-independent packages with GNAT.

From: Hyman Rosen <hyman.rosen@gmail.com>
Date: 6 Feb 2006 12:44:26 -0800
Subject: Re: Ada & MacOS
Newsgroups: comp.lang.ada

Much more so now than in the past. The biggest missing feature is implementation of the "export" keyword, and that's more due to deliberate foot-dragging than anything else, by vendors who loathe the feature. There are also legacy features that remain in some implementations that would not be there in a completely conforming compiler. But you now have to be quite expert in C++ to find the missing features in modern compilers.

What about all the standard annexes which vendors may choose not to implement? If I write a standard-conforming distributed program in Ada, is portability pretty much guaranteed?

Java Exception Model and Ada

From: Peter C. Chapin <pchapin@sover.net>
Date: Fri, 18 Nov 2005 11:48:46 GMT
Subject: Java exception model. Was: Re: Ada Quality and Style book discussion ("_Type" suffix)
Newsgroups: comp.lang.ada

Brian May wrote:

> With respect to his complaint on exception handling – I like the Java model where every exception that can be raised by a function has to be declared – that way you don't have to check for exceptions that don't currently occur – and if the specifications change, the compiler can generate an error to let you know that you may not have considered an exception.

The problem with Java's model is that it forces the programmer to deal in some way with exceptions that semantically can't happen. Consider

```
procedure Outer is
begin
  if Some_Complicated_Check then
    Inner;
  end if;
end Outer;
```

Suppose procedure Inner raises an exception under certain conditions yet can't do so in the code above because Inner is only executed when the complicated check succeeds. Assume that under those particular conditions, it will never fail. The Java exception model would require us to either handle an exception that will never occur, or declare that Outer might raise an exception that we know it will never raise. Such a declaration will force Outer's callers to also do something about this impossible exception as well, etc, and so forth.

The example above is simplistic and contrived but it's my belief that in real programs this sort of issue comes up a lot.

I agree with the quoted article, though, in that using exceptions properly is surprisingly tricky and that it does require the programmer to think about non-local issues. I think there are times when the old fashion method of returning error codes is probably better. However, a blanket prohibition against exceptions is probably an over reaction.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 18 Nov 2005 14:18:43 +0100
Subject: Re: Java exception model. Was: Re: Ada Quality and Style book discussion ("_Type" suffix)
Newsgroups: comp.lang.ada

I can't tell for Java, but let consider Ada adopting a contract model of exceptions. Your reasoning above is flawed. If the designer of Outer knows that the exception X cannot propagate out of it then it should be:

```
procedure Outer is
begin
  if Some_Complicated_Check then
    begin
      Inner;
    exception
      when X =>
      -- This is not a state,
      -- it is a bug
       raise Program_Error;
    end if;
  end if;
end Outer;
```

I think it is a software design issue. When exceptions and exceptional states are considered as *valid* states, then there cannot be any good argument against contract model of exceptions. They belong to a *functional* part of the program. What is left, are the arguments like – it is too difficult to implement; there would be too big overhead; I don't know how to do it right – not much impressive. Alternatively you can say, OK, exceptions are exclusively for software bugs. But this is also a quite weak position, because if they are bugs, then why would you like to handle them? Bugs to be debugged!

From: Peter C. Chapin <pchapin@sover.net>
Date: Sat, 19 Nov 2005 18:06:05 GMT
Subject: Re: Java exception model. Was: Re: Ada Quality and Style book discussion ("_Type" suffix)
Newsgroups: comp.lang.ada

One of the arguments for exceptions is that they simplify error handling. I'm not sure code above is much of a simplification over traditional methods. For example, I don't really want to be forced into the style above for Constraint_Error every time I index into an array. In any case, using the Java model procedure Outer would have to declare that it might raise Program_Error. Do you really want to force programmers to put such declarations on every subprogram? That would render those declarations pointless, wouldn't it?

Actually in Java, some exceptions are unchecked like the null reference exception¦ because it's clear that forcing programmers to either handle it or declare it as thrown in every method would be excessive. A program would be littered with "useless" handlers or else every single method written would have to say that it might throw a null reference. Java thus has two classes of exceptions: those that are checked and those that are not. How does one decide into which class a new exception should go?

The whole issue seems like a nasty morass to me.

Porting from ObjectAda to GNAT

From: Per Sandberg <per.sandberg@bredband.net>
Date: Thu, 08 Dec 2005 21:49:51 +0100
Subject: Re: Gnat calls to Aonix DLL
Newsgroups: comp.lang.ada

> I am trying to convert a project written in Aonix Ada 7.2.2 to GNAT. I would like to call the existing dll libraries compiled in Aonix from gnat compiled code. I am doing this because the packages are large and it would help me convert the code in pieces. At this time some functions seem to work but others cause a segmentation fault. Can anyone help with a procedure for calling the Aonix Ada compiled dll from GNAT?

I have been running both ObjectAda and GNAT in parallel and I had found that the easiest way to move from ObjectAda to GNAT is the following approach.

The maybe tricky parts:

* Get a clear view of the build dependencies in the current system.
* Get all your source code to match the GNAT naming conventions.

The boring part:

* Set up a project structure matching the ObjectAda structures with GNAT project-files ".gpr" files (no library projects at this point).

The fun part:

* Build your programs using GNAT.
* Verify.

The final part:

* Change the "library" projects to be real library projects (static) in the GNAT environment and do a complete build.
* Change the desired static libraries to be dynamic.
* Recompile and copy the DLL:s to the correct directory for execution.
* Verify.
* Done.

Note: I have done this with a >2MSLOC system almost single handed.

The Use Clause: That is the Question

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 19 Nov 2005 20:28:25 GMT
Subject: Don't use the "use" clause
Newsgroups: comp.lang.ada

> You introduced a name collision by the 'use' clause; that can always cause name collisions. The proper solution in that case is to use the qualified name.

I agree. I personally take it an extreme step further, and never use 'use' clauses anywhere. They shouldn't even be in the language. Every Ada project I've worked on that had a coding standard banned the use clause, and rightly so.

I don't only oppose it because of the ambiguity, but even when there is no ambiguity, it's a severe inconvenience to have to grep a large tree to hunt down a declaration. Then to possibly get multiple hits and have to compare two lists of packages to discover which hit is the correct one. By the time you make it to the declaration you're looking for, you've forgotten why you need to look at it :)

Clearly the typing time saved by the use clause cannot possibly offset the time lost on all the resulting code searches.

From: Ed Falis <falis@verizon.net>
Date: Sat, 19 Nov 2005 20:35:36 GMT
Subject: Re: Don't use the "use" clause
Newsgroups: comp.lang.ada

If you were using a modern editor, with Ada cross-referencing, you wouldn't have this problem. GPS, Emacs Ada mode, ObjectAda and quite a few others provide this.

From: David Emery <demery@cox.net>
Date: Mon, 21 Nov 2005 12:36:47 -0500
Subject: Re: Don't use the "use" clause
Newsgroups: comp.lang.ada

I've been in the middle of several debates on this. My personal strong belief and experience has been that qualified names are very useful in comprehension, particularly trying to grasp the 'big picture' of software structure.

So in one previous life, when handed a package that did not have qualified names in it, the first thing I'd do is add the qualified names.

It's possible to construct programming environments that can show you the unambiguous source for each name/operator. But such information is transient, it only lasts for as long as you have the mouse/etc there. Often I'm sufficiently "Luddite" that I print out and scribble over hard copies of programs.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 19 Nov 2005 23:40:46 GMT
Subject: Re: Don't use the "use" clause
Newsgroups: comp.lang.ada

Dmitry A. Kazakov wrote:

> No, name collisions better be prevented by making "use" illegal when it hides anything.

The use clause is barred by coding standards for a good reason, and I don't think it's necessarily name collisions that drive this rule. Programmers are forced to resolve name collisions, with or without the USE clause. The use clause is banned because it makes code unreadable and difficult to use. As Joel Spolsky said, travel should be minimal when interpreting a line of code. Fully qualified naming is a better style because you know immediately, from the code itself, where the identifier lives. And if you need to see the declaration, you know immediately where to go.

> BTW, what about banning implicit "use" of "Standard"? Care to write an AI to make Integer, "+", "-" etc invisible? (:-))

The "use type" clause is a different beast, and I do not object to its use; nor is it banned in any coding standard I've read.

> IDE should have "go to declaration" button.

I agree, but not all IDEs have that luxury. The last IDE I worked in had the option, but it was broken. My current environment is Emacs, which doesn't offer that feature by itself. I've only seen it when Emacs is paired with Apex. Even under the best tools, where a mouseover might reveal the home for some declaration, it's still poor style to not have that information in the text, so the reader doesn't have to mouse around compulsively, as the keyboard is faster than the mouse.

>> Then to possibly get multiple hits and have to compare two lists of packages to discover which hit is the correct one. By the time you make it to the declaration you're looking for, you've forgotten why you need to look at it :)

> It is no matter "where", "what" does matter. If you need to frequently browse sources to determine "what", then the program is poorly designed.

Certainly not. I would say just the opposite. If you're repeating information from your declaration in your identifiers, then you've created a maintenance problem by introducing too much noise, also forcing identifiers to change whenever the declaration changes. A good design doesn't repeat this information.

>> Clearly the typing time saved by the use clause cannot possibly offset the time lost on all the resulting code searches.

> What about the time spent on reading something like A.B.C.D.E.F.G.H?

If you have something like that, then there's something wrong with the architecture of your project. A user should not need visibility into such a deep level within an external component.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sun, 20 Nov 2005 03:57:57 GMT
Subject: Re: Don't use the "use" clause
Newsgroups: comp.lang.ada

Stephen Leake wrote:

> Hmm. Even for operators?
> A := B + C;
> should work when A, B, C are

Cartesian Vectors, not just scalars. And a use clause in a small function can easily improve readability.

That's what the "use type" clause is for. I have no objection to the use type clauses. I use them myself, and coding standards always accommodate them.

The version of Emacs they've installed at work does not have this capability, and I'm stuck with what they provide; but it wouldn't matter anyway. Even with that feature (which I've had on previous projects), it wouldn't be worth it to constantly search like that, when disciplined Ada programmers can simply follow the coding standard and fully qualify external identifiers.

As a beginner I was tempted to use the use, but after I was forced to fully qualify, I've discovered that it's much easier to read code from others as well as old code of my own. I would never go back, even if Emacs had a mouseover cross reference.

> It's not the time typing I'm worried about, it's the time reading and understanding.

In that case the time you're worried about is what's reduced by fully qualifying your names. If you get the information as fast as you can read it, you know what's going on faster than you can even reach for your mouse. You just cannot beat instantaneously knowing which names are internal and which are not, and where they come from.

There is one case where I might be willing to tolerate the use clause on a project. If an editor existed that would fully qualify the names (inline) as it loads the buffer so I wouldn't even need to hover over them with a mouse, and the project supported such a tool, then it wouldn't matter to me either way whether I had to read code that used use clauses. AFAIK, no such tool exists, or at least it's certainly not mainstream.

A better approach would be to have the controlled code checked in without use clauses, and if some hacker wants the package names hidden, or other hidden information for that matter, then it would be easier to make that a check out for browse option.

At one point I worked on a project that did not enforce their prohibition on the use clause. They simultaneously mandated a lousy Windows-based tool set, which did not have cross referencing, and to worsen things, grep was not provided either. So we had to search using the crappy native Microsoft search tool, which does not support regular expressions. The folks on that project who used use clauses needlessly robbed me of copious man/hours.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sun, 20 Nov 2005 19:48:23 GMT
Subject: Re: Don't use the "use" clause
Newsgroups: comp.lang.ada

There is some justification for this. SPARK, for example, does not have the use clause (but does have the use type clause).

I tend to work from the idea of what the reader should know. I expect my reader to know Ada, and as such to be familiar with the standard library, so there's no real problem with using Ada.Text_IO. On a specific project, if there's a standard library used on the project, people working on the project should be familiar with the library, and using the library packages should not be a problem. However, I don't expect everyone to be familiar with the entire system, so application-specific packages should generally not be used.

From: Peter Amey <peter.amey@praxis-cs.co.uk>
Date: Wed, 25 Jan 2006 11:01:28 +0000
Subject: Re: Don't use the "use" clause
Newsgroups: comp.lang.ada

> What does SPARK gain by outlawing use clauses? Surely the examiner can handle the extra name resolution!

I haven't followed all of this thread but do wonder if poor initial name choice is sometimes a driver for the desire to employ "use"? I think the original topic was Ada Quality and Style so naming may be on topic anyway.

I think the trick is to choose names knowing that they will be read in sequences separated by dots. Then the desire to strip away chunks of the name becomes less pressing.

I often see (ghastly) things like:

```
Engine_Sensor_Class.Engine_Speed_
Sensors.Turbine_Speed.Read_Turbin
e_Speed
```

no wonder people want to employ use clauses to shorten it!

If instead we had:

```
Sensors.Speed.Turbine.Get
```

then a use clause is less useful and might even be positively misleading.

A side benefit of banning "use" (but not "use type") which we do in SPARK, is it encourages this kind of naming because nobody ever has to worry about what a name might look like with bits of it missing.

Ada to C++ Translator

From: Jeffrey R. Carter <jrcarter@acm.org>
Newsgroups: comp.lang.ada
Subject: Re: Ada to C++ translator
Date: Tue, 24 Jan 2006 22:39:16 GMT

> I am looking for an Ada to C++ translator. The converter will only be used as an intermediate step and not used on sections of code we will be re-architecting to make use of C++ functionality.

Such a beast is impossible, since there is no translation for tasks and protected objects.

From: Gautier de Montmollin <gdemont@hotmail.com>
Date: Wed, 25 Jan 2006 00:25:15 +0100
Subject: Re: Ada to C++ translator
Newsgroups: comp.lang.ada

It seems you are indeed looking for an Ada compiler producing C or C++ code (since you mention that the converter is an intermediate), and such tools exist. For instance: http://www.sofcheck.com/products/adamagic.html

From: David Emery <demery@cox.net>
Date: Tue, 24 Jan 2006 18:26:14 -0500
Subject: Re: Ada to C++ translator
Newsgroups: comp.lang.ada

That's not true. There's no 1-1 translation, but a POSIX-based runtime certainly can show how to translate Ada tasking to a sequence of C/POSIX primitives (Mutexes, Semaphores, etc). It's certainly non-trivial, but it can be done.

A good recent paper on sequential Ada is:

Audsen, Howard & Nyberg, "Using ASIS to Generate C++ Bindings", Proc SIGAda 2005

For tasking constructs, search for papers by Ted Baker and/or Ted Giering.

If you're a SIGAda member, it's in the proceedings you got last month :-)

From: James Alan Farrell
Date: Wed, 25 Jan 2006 17:30:11 -0500
Subject: Re: Ada to C++ translator
Newsgroups: comp.lang.ada

I used such a tool on a previous project (which proves it can be done). I would say if it could not be done (because of tasking or other considerations) then Ada probably could not be compiled or run. Since it obviously can be, why can it not be compiled to another language such as C++? (To my mind, compiling means to a specific runtime environment, such as to Linux on a PC, so I do not buy the argument that converting to C++/POSIX is different somehow from converting to C++)

Unfortunately I do not recall the name of the tool we used.

I do recall that after converting a substantial amount of Ada code, a number of programmers were employed full time for six months fixing up the C++ to make it readable. My office mate was one of them. It is also possible that they were making corrections to mistranslated code, but the majority of the effort was simply to make it readable.

From: Charlie McCutcheon <charlie.mccutcheon@hp.com>
Date: Fri, 27 Jan 2006 15:01:26 GMT
Organization: Hewlett-Packard Company
Subject: Re: Ada to C++ translator
Newsgroups: comp.lang.ada

Recently, I'd heard of such a thing, seems to be at: http://www.softresint.com/expe.htm

I'm skeptical that the translation would be very good. I'd predict lots of cost for hand fixing problems. They do at least acknowledge that Ada and C++ are "different".

Mr. Safety and Mr. Sloppy

From: Anonymous Coward <anonymous@coward.org>
Date: Thu, 16 Feb 2006 02:15:17 GMT
Subject: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

I'd like to start with a war story:

"Despite the lack of coding standard, Mr. Safety wrote a well constructed package that uses private types, then wrote some packages that use those types. After these packages all reach a mature and refined (and tested) state, another developer (Mr. Sloppy) finds that they need to use Mr. Safety's package, which requires private types in the interface. Mr. Sloppy refuses to work with private types. It's the typical anti-strong typing mentality, where the developer refuses to accept anything that might limit their power.

Mr. Safety was forced to introduce support for duplicate public versions of these types to accommodate Mr. Sloppy's skill limitations; which obviously produced a sloppy free-for-all in the work product."

The understanding that most s/w developers seem to have is that they design their own interfaces for packages they create, and users of that Ada spec only have a say in whether it meets standards and requirements. I always bend to accommodate types that other developers require in their spec, because it's theirs. Maybe I'm wrong about what I think is typical. It was explained to me that interfaces are "shared" and are no more controlled by the author than the users of it.

That story is just a sample of what I encounter too frequently in the Ada workforce. It seems a /majority/ of Ada developers have no formal Ada training, and are primarily C developers who picked up the Ada syntax on the job. Consequently, Ada principles are lost, and much of the Ada code out there is only slightly safer than C code (but still safer primarily because even a poor Ada developer cannot write ambiguous code like they can in C).

I've only worked on four or so workplace Ada projects. The projects with elaborate coding standards produced substantially better code, but I think it was just chance that those projects also had Ada enthusiasts who used private types, as the coding standard did nothing to promote private typing.

Do you folks encounter this frequently? And what's the solution? Management can never appreciate the benefits of concepts like type safety. Strong typing is incorrectly viewed as "academic" and counter to progress.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Thu, 16 Feb 2006 09:32:43 +0100
Organization: Adalog
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

Ada was designed to induce a change in mentalities. A change in mentality is much harder to introduce than a change of programming language.

Moreover, training is felt by management as a waste of time and money. Why should we pay to get our engineers not working for a week or two? This newly hired guy does not know Ada? Just give him some code to read (Ada is so readable after all); he's a talented guy, he will learn quickly. In only rare cases, the guy is given Barnes' book (not to be read during work hours).

Currently, we see many projects in Ada, and at the same time the attendance to my training sessions has never been so low. I'm worried, not for my business (the said projects provide me with enough occupation), but for this growing idea that proper training is not necessary.

I just added a new rule to AdaControl (available in the next release) to check "while true loop .. end loop;". I did so because I found it in actual programs. It is a great indicator of modules written by people without any Ada education, and which certainly deserve peer review!

From: Peter C. Chapin <pchapin@sover.net>
Date: 16 Feb 2006 16:10:08 GMT
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

I agree with this. It also works in another direction! I believe I became a significantly better C programmer after studying Ada. Like all good programming techniques, the concepts end up being language independent.

From: stephe_on_the_web@toadmail.com
Date: Thu, 16 Feb 2006 05:20:34 -0500
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

My view is that interfaces are always negotiated between the implementers and the users, preferably with input from the system architect; they both need the interface, but have different needs from it. The interface represents the best compromise they can achieve.

They do have to be controlled; you can't let just anybody force a change to an interface.

In my experience, most programmers have inadequate training. Most of my current job is teaching people how to write good code. I find it helps that I'm also teaching them Ada; it helps them to abandon their preconceptions. It does give Ada a rep of being "hard to learn", but I can live with that.

I suspect that's a mindset issue; if you like elaborate coding standards, you are likely to also like private types and strong typing.

[The solution is] programmer education and strong project management. Managers need to be educated along with the programmers.

The best way to educate managers is by demonstrating an impact on the bottom line. If you can show that good programming actually saves time and therefore money, they will listen.

But you need support in getting that process started. That's when you need stories from others that have used good programming and saved money; check http://www.adaic.com for good stories.

On the other hand, some programmers will never be "good", and you need to get them off your project.

Unfortunately, we don't have a good database of examples to show it is wrong that [strong typing is counter to progress]. In my job, I have sufficient clout that I can say "on my projects we use Ada, because it is the best language". However, other project managers don't want to be bothered with learning new tools, or training their staff; they are unconcerned about productivity.

Since I work for the government, it's hard to use the profit motive. Instead, it helps to remember that our real purpose is to spend the taxpayer's money in the congress-critter's district. That explains why we often do things in the most inefficient way :)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 16 Feb 2006 21:17:14 +0100
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

At Barco, I am part of a small team that has been using Ada for the past 8 years or so. I've been there for only two years myself. My colleagues are software engineers, not just coders. They all understand the power of type safety, and their desks have badges with the

Countess' effigy and the motto: "In strong typing we trust – Ada – 1983 – 1995 – 2005". Last year, I wrote the new version of the coding standard, and everyone on the team has a culture of following it. So, my experience is quite exactly the opposite of yours, and as a consequence I'm a happy software engineer :)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sat, 18 Feb 2006 00:09:16 +0100
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

> What would one put on their resume to selectively stand out to these companies, and simultaneously deter the sloppy projects from offering an interview?

I think you can just explain what you're looking for in your CV. Then add a couple of keywords such as "quality", "coding standards" and "software engineering".

If you evolve to that point (I haven't yet but am planning to), you can replace these keywords with "formal methods", "proof of correctness", "SPARK".

It has worked for me at least.

From: Richard Riehle <adaworks@sbcglobal.net>
Date: Thu, 16 Feb 2006 23:57:01 GMT
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

I was at an Ada conference many years ago, before the end of the mandate, and visiting the booth of a well-known CASE tool publisher. They began to demonstrate the tool for me.

At one point the person doing the demonstration said something such as, "Well let's get rid of these limited private types since they cannot be used for anything useful." He then continued with his demonstration of the tool.

I was horrified, but decided to remain polite. It would have been no good to try to educate him to the contrary.

From an engineering perspective (although not from a programmer's point-of-view) it is quite valuable that we cannot overload the assignment operation and do other little things that are easy in languages without the equivalent of limited private. Yes, to do assignment on a limited type we must create a procedure, but that is not a bad thing – it is a good thing.

One of my early mentors in Ada, Doug Bryan, once said, "Until you understand 'limited' you don't understand Ada." I eventually learned just how right he was.

With Ada, we are not trying to appeal to the programmer. Rather, we are concerned with good engineering practice. Ada continues to be the best language available when one is focused on engineering rather than programming.

From: Marc A. Criley <mc@mckae.com>
Date: Fri, 17 Feb 2006 07:39:33 -0600
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

An Ada seminar run by Doug Bryan that I attended quite some time ago continues to exert a profound influence on my Ada programming practice and understanding. In that seminar he focused on Ada's "type model" and how that was the foundation of the language's definition. The proper definition of types embeds mountains of useful information about your program that can be programmatically extracted by querying the type model for that information, which is mostly done via attributes.

That's why I occasionally get on the soapbox that while "strong typing" is a strength of Ada, it's only _part_ of the story, the rest is having access to information about types and their instances that is being implicitly encoded into the software due to Ada's type model based definition.

With the fallacious dismissal of strong typing, not only the defensive aspects of Ada are being thrown out, but the revelatory ones as well.

From: Brian May <bam@snoopy.apana.org.au>
Date: Fri, 17 Feb 2006 12:57:54 +1100
Subject: Re: Working with incompetent adaists / unsafe typing war story
Newsgroups: comp.lang.ada

A big hurdle I find (not just in Ada software) is that the API Mr Safety carefully designed and implemented is insufficient for the requirements of the project. As a result, and due to demands from management to get the project finished Yesterday, Mr Safety is forced to expose a lot of the inner workings which he never intended.

The reason why the API was insufficient? Because Mr. Safety didn't understand all of the requirements. The reason? Because management considers the design phases of the very complicated software a complete waste of time and money. Bugs will occur anyway. What is the point? However, Mr. Safety wanted to try to do the right thing. So he tried to do the design. Unfortunately he couldn't see into the future for what would be required, as Management considers each modification as a totally isolated project.

Not only that, but Mr. Safety wasn't given time to document the API. As such other programmers tied themselves up in knots, either by continuing to do things in obsolete ways, or by making changes to the API that aren't required and break other things in horrible ways. This can lead to conflicts between Mr. Safety and the other programmers in doing things the "correct way" vs. the "quickest way" with management supporting the later. Not only that, but even Mr. Safety wasn't sure how the API was meant to work, as he wrote it years ago and hasn't had an opportunity to look at it since. During this time other developers have gradually been changing it in ways which look totally inappropriate and Mr. Safety doesn't understand.

At the end of the day, management gets code that appears to work, and they are happy. Other code might be completely broken and need fixing, but that is rule rather then the exception in such projects.

These issues occur regardless of language – admittedly this isn't Ada, and isn't even I strongly typed language, but I think Ada wouldn't help without significant culture change.

I have seen web pages dedicated to discussing why strong typing systems are bad and slow implementation, and the world would be a much better place if everyone used typeless scripting languages instead.

Then people ask how come so many web pages have obvious and known security holes. There was a talk at the Linux conference (LCA2006), New Zealand, in fact. The speaker wrote a program designed to check websites against obvious attacks, such as not quoting user input before displaying it back as HTML to the user, or displaying unquoted user input (meaning HTTP post variables) in SQL error messages. He found so many security problems in common websites around the Internet he refuses to distribute the code for fear that attackers might use it.

These things shouldn't happen …

Oh well, such is life.

I only hope that software written for mission critical applications is better.

Exception Safety in Ada

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 30 Nov 2005 14:57:07 +0100
Organization: CERN - European Laboratory for Particle Physics
Subject: Controlled types and exception safety
Newsgroups: comp.lang.ada

I try to understand how the Controlled types work and I spotted one "small issue" that makes it difficult to write exception-safe code.

The "exception-safe" means that code behaves "correctly" in the presence of exceptions, for some chosen definition of "correctly".

In C++ we define the following levels of exception-safety:

- level 0 (no guarantee) – in the presence of exception, anything can happen, memory may become corrupted, data structures may become completely mangled, etc.

- level 1 (basic guarantee) – in the presence of exception, no resources are leaked and objects are in a coherent, but not necessarily predictable state.

- level 2 (strong guarantee) – in the presence of exception, the program state (to the relevant extent) remains unchanged. This is similar to the commit-or-rollback semantics known from databases.

- level 3 (nothrow guarantee) – the code simply guarantees that there are no exceptions.

Why is this classification useful? Let's say that I have an abstract data type that implements some data structure – a stack, for example. I can classify the stack's operations by assigning them any of the above four levels, so that I know what can be expected when an exception is thrown for any reason (like inability to allocate more memory, or alike). For example, if the Push method of the stack gives me the strong guarantee (level 2 above), then I *know* that by calling this method either the new element will be appended to the stack, or the stack will remain unchanged, so that even if the exception is thrown, I don't have to worry about the stack's internal consistency.

This is useful.

This is useful also in assignment operations. Since stack can be a dynamic data structure, assigning one stack object to another may involve destroying one existing data structure *and* creating a new one (a copy) in its place. Similarly, the quality implementation should provide the strong guarantee, so that I *know* that either the stack was properly copied, or there was a problem during assignment and an exception was thrown, but nothing changed in any of the objects involved.

Let's say that I want to write a stack in Ada. Making it a Controlled type seems to be a good idea, so that we have hooks for initialization, adjusting and finalization. Let's say that I have two stack objects, X and Y. These objects were populated with some data, so that each of them manages its own internal dynamic data structure. Now, I do this:

X := Y;

and the following happens (this is what I understand, please correct me if I'm wrong):

1. X is finalized. This allows me to clean up (free) its internal data.

2. Y is *shallow-copied* to X, so that in effect X and Y share their state.

3. X is adjusted. This allows me to duplicate its internal structure so that it becomes independent from Y.

Later:

4. Both X and Y are finalized. This allows me to clean up (free) their resources.

For everything to work correctly it's important that two separate stack objects *never* share their internal dynamic data structure, otherwise bad things can happen. It would be also fine not to leak memory.

Now, the interesting part: let's say that during adjustment (3.) some error happened (like low memory condition or whatever) that resulted in raising an exception (note: this exception might be actually risen not by the stack code, but by the assignment operation of the stack elements, even somewhere in the middle of this process). Bad things will happen in subsequent finalization of those objects, unless I handle it by cleaning up everything that I already managed to duplicate (but still, this leaves me with the empty stack).

I think that the inherent problem comes from the fact that the finalization of X was forced *before* its adjustment. The canonical C++ way is to *first* make a copy of new value (because this is when errors might occur, so that even if they occur, there was no change in the destination object) and *then* inject the duplicate into the destination object, getting rid of its old state (and this is assumed to be nothrow).

The "Ada way" looks like selling the house *before* looking for the new one.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 30 Nov 2005 16:06:52 +0100
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

ARM 7.6.1 reads: "It is a bounded error for a call on Finalize or Adjust to propagate an exception. For an Adjust invoked as part of an assignment operation, any other adjustments due to be performed are performed, and then Program_Error is raised."

Here the semantics of "copy", "inject", "duplicate" is ill-defined. In general, you can copy a set of bits, but you cannot an object without defining it in the terms copy-constructor. In Ada's case copy-constructor is defined as Bitwise copy + Adjust. It is an atomic operation. Which is equivalently means that in general case you cannot define any reasonable semantics for its partial completion.

I don't let exceptions propagate.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 30 Nov 2005 17:52:24 -0600
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

It's a bug to let Finalize or Adjust propagate an exception. If they do, the only reasonable assumption is that the object is corrupted. The language bends over backwards to insure that a failure of one of these operations for an object does not corrupt any other object (or component), which is a strong guarantee in itself.

In just plain old (no controlled types around):

A := B;

the raising of an exception during the assignment leaves A abnormal if A is composite. In other words, Ada says that objects that are being assigned are corrupted by an exception.

The solution is to not allow exceptions to be raised by Adjust. Yes, that's not completely practical, because of Storage_Error, but even there you should handle the exception and do what you can to prevent corruption of the object. (Claw leaves the object invalid in this case, so future operations on it, other than recreating it, will fail.) And this also suggests that you should try to avoid allocating memory in Adjust (not always possible, of course).

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 30 Nov 2005 17:19:23 +0100
Organization: CERN - European Laboratory for Particle Physics
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

When I said "copy" above (C++), I meant create a new object as a copy. This involves copy constructor. The point is that this new object is *separate* from the destination object (from what's on the left side of assignment operator), so even if there are errors, they do not influence any of the two objects which were originally involved. After this new helper object is constructed (which means: *successfully* constructed), it's "injected" in the destination object by means of swapping the bowels. This idiom is very effective.

[In Ada,] the assignment is defined as Finalize + Bitwise copy + Adjust. And it's the fact that Finalize comes first that bothers me.

Note that in the example above there is no "partial completion". On the contrary – either the operation completes successfully or it fails *without* modifying anything. Moreover, the scheme does not force me to ignore the error nor anything like this, I can let it go to the place where there's enough context to really handle it.

What do you mean by "don't propagate"? What if there is an exception that was raised by the run-time (like low memory condition) in the middle of adjusting the whole stack? What should I do with the part that was already adjusted (duplicated)? What should I do with the part that was not yet adjusted? Should I clean up what's already done and leave the destination stack as empty and shut the exception up, thus preventing the higher-level code from properly handling it?

Is it possible to have assignment with strong exception guarantee?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Dec 2005 10:21:35 +0100
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

The real problem is that you cannot both look into an atomic abstraction and pretend that this look is consistent with the abstraction. The problem comes with user-defined constructors. The language generated ones are composed out of parts which can be reversible, provided the language designer knows how to do it. But a user-defined constructor is irreversible, otherwise than by, again, a user-defined destructor. Now you are sitting in a rocket, a user-defined constructor has just turned on the ignition, and oops, you notice that you have left your hat at home!

Consider an exception raised while construction of the copy. The copy is corrupt. Both to destruct or to just deallocate it could be wrong.

[Also, a] user-defined constructor as a concept is useless if I cannot construct in-place. Consider construction of non-movable objects containing self references or bound to definite memory locations.

Exception-safety is irrelevant to the issue. If Ada should ever have user-defined constructors and assignment (because Ada.Finalization is not), then I would really like to have an access to the left part of the assignment. In my opinion, the model could be:

1. Compiler-generated assignment is generated as Finalize + Copy-constructor.

2. User-defined assignment can override it. However, there are many tough problems. The assignment should be able to change the constraints (i.e. bounds, discriminants, tags.) It should be composable against aggregation. It should have access to the left part, but also be able to override it in-place.

As far as I know, there is no language which does it right.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 1 Dec 2005 22:17:20 -0600
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

Ada doesn't really have user-defined assignment; if you *really* need that you have to use a procedure.

And in any case, what you are asking for would be contrary to the efficiency goals of Ada. You're saying that all assignment *have to be* made to temporaries. I can imagine some language working that way, but it's performance would be several times slower than C++. We have enough trouble with people thinking Ada is slow as it is, without duplicating all of the effort twice. There is a lot of words in the Ada standard specifically to allow implementers to optimize (eliminating the assignment temporary, for instance).

Ada's model is that a failed assignment leaves the target corrupt. You can mitigate that, but not completely eliminate it. If that is unacceptable for your application (and I can think of few for which that would be the case), then you have to avoid ":=" (most likely by using a limited type). At least Ada 200Y improves the support for limited types a lot, so that you no longer have to give up fancy constructors, aggregates, and constants when you use them.

In your example of a failed stack assignment, the Adjust routine ought to clean up the mess if Storage_Error is raised, and leave the target Stack empty. Is this ideal? Possibly not, but it hardly matters, because there is no clean way to know what might or might not have been done when the assignment fails (read 11.6 if you believe otherwise); recovery means exiting out and rolling back far more than one object. (And there really is no safe, portable recovery from out-of-memory conditions – you can only figure out what works with a specific compiler and target and do that.)

Probably not specifically. Obviously it depends on the implementation. I just looked at [our implementation of unbounded string], and it does nothing at all to handle memory issues in Adjust. That means that the object would fail the invariants after such an assignment. (I didn't actually realize that; it would be better to null the pointer in that case!) And presumably, it would eventually access through a deallocated pointer. But that's all a correct (if unfriendly) implementation of Ada, because the object is abnormal, and any access to it is erroneous – see 13.9.1. (Humm, this actually isn't as clear as it ought to be; one could argue that Storage_Error isn't a "language-defined check" (its not called that in 11.1(6)). But surely it is intended to be covered; it's hard to imagine a case that is more likely to corrupt things than running out of memory. And it is indexed as a check. So I apply Dewar's rule [the Standard never says anything silly]. Anyway, sorry about the language lawyer musings :)

Moral: don't touch the left-hand side of any assignment after it failed raising an exception, other than to assign a new value to the *entire* value. If you want some other semantics, don't fool yourself and others by calling it ":="; use limited types and appropriate copying procedures.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Fri, 02 Dec 2005 10:29:40 +0100
Organization: CERN - European Laboratory for Particle Physics
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

OK, and now it's bright clear to me. I got an impression that Controlled types can buy me the same syntax sugar with the same flexibility in exception-safety guarantees that I have with assignment operators in C++. It's not bad that they don't – but I have to know it. [...]

Interestingly, in the Stack example there is no performance tradeoff – you *have* to do both cleanup and state duplication anyway, no matter what's the provided guarantee, but by introducing the temporary object I can force the specific *order* of those operations (first duplicate, then clean up) that gives me the strong guarantee – which means commit-or-rollback. It's a free lunch in C++ and therefore there's no reason not to have it in types like string, stack, etc. In particular, there's no efficiency loss. OK, you can argue that in this scheme you have to first create a duplicate and then destroy the old state, which means that for some short period of time we consume more memory (which, funny, makes it more likely to fail because of memory shortage :)) and that can result in lower cache hit rates and this kind of stuff. But as already said – it's *my* responsibility to judge the tradeoffs for each case separately. It's not true that this should be done everywhere.

From: Maciej Sobczak <maciej@msobczak.com>
Date: Tue, 06 Dec 2005 10:00:39 +0100
Organization: CERN - European Laboratory for Particle Physics
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

>> But X := Y overwrites X before calling Adjust on it, so you can't store the backup copy, or any way of accessing the backup copy, in X.

> Right. But Adjust, in case of problems, could still find the copy of the old X in the "to be deleted" backup queue and restore X from there. I didn't say this was nice, just that it was possible. ;)

Except that it doesn't solve anything. The whole issue with this commit-or-rollback implementation is that it should not just suppress the exception and pretend that nothing happened – it should guarantee the old state and at the same time let the exception fly out to the place where it could be actually handled, whatever that means in the given context. I've started with the assumption that function ":=" is allowed to fail – in the sense that it can raise exceptions. It's not, and therefore there is no point in implementing any failover features in it. It has to either guarantee the success or not be provided at all and the type should be limited.

This brings me to the next problem. Let's say that I provide a separate procedure Duplicate or Copy or Assign or whatever with the commit-or-rollback guarantees for some type (like Stack). Now, some of the types in my program will have ":=" for assignment, and some others will have the Copy procedure, but not ":=". I want to create a generic container or some other component that will copy things around internally. It has to use ":=" for some types (like Integer) and Copy for others (like Stack). In C++ I solve this problem (aside the fact that there is no problem in the first place) with template type traits or some other application of template specializations.

What about Ada?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 6 Dec 2005 10:50:22 +0100
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

```
generic
  type Object is limited private;
  with procedure Deep_Copy
   (Left : in out Object;
    Right : Object) is <>;
package Container is
   …
end Container;
with Container;
generic
  type Object is private;
package Specialized_Container is
  procedure Deep_Copy
   (Left : in out Object;
    Right : Object);
  pragma Inline (Deep_Copy);
  package Copying_By_Assignment
    is new Container (Object);
end Specialized_Container;
package body
Specialized_Container is
  procedure Deep_Copy
   (Left : in out Object;
    Right Object) is
  begin
    Left := Right;
  end Deep_Copy;
end Specialized_Container;
```

Note also that your example is not much realistic. Transaction model is expensive. One usually does not compose transactions. This means that components of a container will be copied destructively, *after* necessary memory allocation. Only the upper level will take care of a possibility to roll things back. Thus it makes much sense to distinguish light-weight ":=" (which can't fail) and heavy-weight "Copy".

The container itself could be a red-black tree, which supports roll-backs after mutations.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 06 Dec 2005 18:34:38 GMT
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

[...] For Container to work correctly for all possible actual types, the assignment procedure must have Left be mode "out". Now the uninitialized actual for Left is not checked on entry to the procedure, and it works correctly for scalars. For composite types, there is a whole collection of situations in which "out" really means "in out", so the user can still write a meaningful procedure that can inspect the contents of Left.

Personally, I would have preferred

```
procedure R'Assign
  (To : in out R;
   From : in R);
```

for any record type R. This can be redefined by the user:

```
 for R'Assign use
    My_Assignment_Procedure;
```

[...]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 6 Dec 2005 13:34:29 -0600
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

This was the original idea for Ada 95, but it doesn't work. That's because the object on the left-hand side may come into existence because of the outer assignment, or disappear because of the assignment. The beauty (and curse) of Adjust is that it can be called by itself when needed, or with an appropriate Finalize.

You can't, in general, read the object that you're assigning into. That means that user-defined assignment in Ada can never be as powerful as that in other languages (unless you somehow prevent the types from being used in discriminant-dependent components – which would probably be a generic contract problem).

For an another explanation of this, see ARM 7.6(17.a-17.h).

http://www.adaic.com/standards/95aarm/html/AA-7-6.html

The other issues are solvable, but this one is not.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 7 Dec 2005 18:50:14 -0600
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

[...] These user-defined assignments have to compose (otherwise, you'd be breaking the invariants of the component types – remember, these components are likely private types, and you might not have any idea how they're implemented). So, you have to be able to *automatically* do the right thing for each component. (This, BTW, is why Ada insists that an exception in one Adjust routine be delayed until all other Adjusts have completed – we don't want a failure in one abstraction to destroy another, unrelated one.)

Say you have an assignment for type R as described above, and a function F returning an object of type R. And you have type S defined as:

```
type S (D : Boolean := False) is
  record
    case D is
      when False => null;
      when True => C : R;
    end case;
  end record;
O : S; -- D = False here.
O := (D => True, C => F);
```

Now, how is this assignment performed if we're using the default assignment here? Since we need to component, we need to call the Assign procedure on the component C, but what left-hand side to pass as To? There isn't a component O.C in the left-hand side!

Now, you could try to (a) require this also have a user-defined Assign [but that's very unfriendly and error-prone] or (b) ban components that have user-defined assignment from being discriminant dependent [but this would be a big contract model problem – or, a lot of things that are currently done in generic bodies could no longer be. For instance, if R was a generic private type, the above type S would have to be illegal in a generic body - not matter what the actual type of R is.]

So there is no solution in the framework of Ada. To solve the problem, you'd have to get rid of discriminants and discriminant-dependent components – and that's not an option for Ada.

Maybe Ada 200Y limited types and Assign procedures would be adequate, but certainly not the Ada 95 variety. Ada 95 limited types don't allow (1) aggregates; (2) constants; (3) useful functions; or (4) any sort of complex initialization. Which means that you can't use many of the techniques that help reduce bugs in Ada (such as letting the compiler check that all components have been given in an aggregate). And limited types also block most optimizations by their very nature. That's useful in some cases, but in others you'd rather let the compiler eliminate extra temporaries and Finalizes. (That's allowed for non-limited types, but never for limited types.)

From: Robert A Duff
Date: 02 Dec 2005 18:51:41 -0500
Subject: Re: Controlled types and exception safety
Newsgroups: comp.lang.ada

> You can get Storage_Error if Adjust allocates storage. Other sorts of exceptions should have occurred during the adjustment of the intermediate object, and so corrected before the assignment to X.
> Storage_Error is a strange beast; there's no guarantee that you can do anything about it. There may not even be enough storage available to execute an exception handler.

Right. In practice, you can get away with handling Storage_Error, but it's annoying that it is pretty-much impossible to write a handler for Storage_Error in Ada that is guaranteed (portably) correct by the RM. Ada is better than some languages, where a stack overflow can go entirely undetected, and the program just starts overwriting who-knows-what data.

But I think I've got a better way, which I would use in my own language design.

My idea is that stack overflow is like an abort or a hardware interrupt. An abort is asynchronous with the running code – it can happen anywhere, even in the middle of "X := Y". If X:=Y is aborted in the middle, we say that X becomes abnormal – perhaps its discriminants are nonsensical, so later code can't even determine the size of X. The solution is to have abort-deferred regions – regions of code where abort can't happen. Inside such a region, you can say X:=Y, and be sure that X is unchanged, or Y is fully copied into it. If somebody attempts to abort in the middle of X:=Y, the abort will take effect at the end, and all is well.

Same thing for hardware interrupts – the solution is to allow (hopefully short) regions of code where interrupts can't interrupt.

Stack overflow is asynchronous in the sense that it can happen pretty much anywhere. So in my fictitious language, you can have regions of code where stack overflow can't happen. The compiler is required to calculate (at link time!) a static quantity that is the max stack usage for each procedure, task, and other relevant construct. This quantity is an integer ranging from 0 up to the max size of the address space (System.Memory_Size, in Ada). Calculations use saturating arithmetic.

When you enter a no-stack-overflow region, we allocate the max size for that region, and raise Storage_Error if that's not possible. So it's like an abort-deferred or interrupt-deferred region, except that the deferral goes backward in time – if Storage_Error _might_ be raised in that region, we instead raise it before entering the region.

Of course, the code in a no-stack-overflow region can't do stuff that allocates unknown amounts of stack space. If a procedure has a local variable of subtype String, with no compile-time-known bounds, the max size is perhaps 2**31 bytes or so. If a procedure is recursive, the max size is System.Memory_Size. If a procedure makes an indirect call (so it _might_ be recursive), the max size is System.Memory_Size. So you write no-stack-overflow regions with small numbers of known-size locals. But that's OK – all you want to do is log the error, clean up some things, and return to a more-global point in the program.

If the max size for such a region is System.Memory_Size, or close to it, the compiler should at least issue a warning, because at run time, every execution of that thing will raise Storage_Error.

Storage_Error also applies to "new", but that seems like an easier problem. The allocator can be "blamed", so heap overflow is not asynchronous like stack overflow. At least, for _explicit_ use of the heap. If the compiler is allocating activation records on the heap or some such, then that's still an issue.

Buffer Overflows and Ada

From: Richard Riehle <adaworks@sbcglobal.net>
Date: Sun, 13 Nov 2005 05:14:02 GMT
Subject: Buffer overflow Article
Newsgroups: comp.lang.ada

There is an interesting article in the current issue of the Communications of the ACM (Vol 48, No 11, page 50) about preventing stack buffer overflow attacks. The authors, Kuperman, Brodley, Ozdoganoglu, Viuakumar, and Jalote, write as if they have never heard of Ada.

In one paragraph, they criticize C as being vulnerable to such attacks and then dismiss Pascal as being unable to address low-level issues. As I read their solution, it became clear that simply choosing Ada for their development language would solve the vast majority of their concerns.

This kind of article appears every now and then. The authors of these articles write as if it is necessary to improve C or invent new tools when all they really have to do is discover Ada.

From: Tom Moran <tmoran@acm.org>
Date: Sun, 13 Nov 2005 01:35:10 -0600
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

They also say "the performance cost of bounds checking (reported in [the 'Cyclone' variant of C]) involves up to an additional 100% overhead."

I tried:

```
-- Lo, Hi, and A are procedure
-- parameters, so their values
-- and bounds are not known at
-- compile time.
for i in Lo .. Hi loop
  A(i) := 0;
end loop;
```

with GNAT 3.15p with bounds checking on or off, -O2, and got a 65% degradation, (Because the bounds are pushing the index out of a register?) In the real world, my impression is that 10-15% is a more common cost of all checking on vs all off. Even at 65%, if the 20% of the code that takes 80% of the time were hand checked and then compiled with checking suppressed, 65% would change to 13% or about 3 months of CPU age by Moore's law.

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Sun, 13 Nov 2005 12:55:39 +0100
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

Some quotes:

"One way to prevent programs from having such vulnerabilities is to write them using a language (such as Java or Pascal) that performs bound checking. However, such languages often lack the low-level data manipulation needed by some applications. Therefore, researchers have produced "more secure" versions of C that are mostly compatible with existing programs but add additional security features. Cyclone [5] is one such C-language variant. Unfortunately, the performance cost of bounds checking (reported in [5]) involves up to an additional 100% overhead."

"Dynamic protection techniques can be costly in terms of overhead, but some researchers are trying to move that functionality into faster, hardware-based protection schemes. As these techniques move from academic laboratories into mainstream software releases, computer users and software developers have become aware of what they can do, and what they can't do."

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sun, 13 Nov 2005 15:58:13 +0100
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

The tables in that paper do not justify the 100% figure, and the paper shows that some of the programs were incorrect, presumably because the authors failed to include run-time bounds checks. The "fat pointer" approach used by Cyclone is not representative of typical compiler implementations of bounds-checked array types, either.

Bounds checks are costly, so lets get rid of them and just use obfuscation techniques to prevent code injection. The truth is that you have to check things at some point, and manually coded bounds checks have been shown to be error-prone (more than compiler-generated ones). For most applications implicit bounds checks are probably a win.

The authors show a profound lack of industry experience. In real-world Internet applications, a typical non-exploitable buffer overflow is still a very serious defect because it affects availability. Shifting bugs from crash-and-control to crash-only isn't such a tremendous improvement, especially in environments which use multi-threading instead of multiple cooperating (but isolated) processes.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sun, 13 Nov 2005 16:02:20 +0100
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

Ada compilers are not designed to be secure in that sense, and there doesn't appear to be any commercial interest to change this. The needs of typical safety-critical software are completely different, and they do not help much with writing secure code (in the "buffer overflow" sense).

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sun, 13 Nov 2005 23:57:28 GMT
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

I found the article quite amusing. [There was only one] paragraph that addressed language choice in the entire article. Considering that language choice is the cause of buffer overflow vulnerabilities, you'll understand why I found the article amusing.

1st, they say languages such as Java and Pascal may not be low level enough. That's certainly not true of Ada, nor of most versions of Modula-2 and Pascal. So this is simply hand waving to justify their decision to use a C derivative.

Then they say that bounds checking adds 100% overhead. This may be true of trying to patch C, but it's certainly not true of all the checks Ada does, which is much more than simply bounds checking. In practice I have never found a case in which leaving checks in was too slow, nor where turning them off saved more than 10%.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 14 Nov 2005 09:35:11 +0100
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

> And then even if there was 100% overhead, what the problem ? For most applications this is not critical and at least for debugging the application this is invaluable. Running with a 100% overhead is equivalent to running with a computer 18 months old. Not that bad :) Again I understand that in some domains we are counting the CPU cycles, but this is not the majority of applications.

This is not the whole truth. I agree that overhead caused by run-time checks is not a big deal. But that is not the problem in my view. Let they be 0%! The real problem is that a check may fail while program crash is not an option. This means that there must be some error handling. More errors may happen at run-time more complex infrastructure one would require. Add here unit tests for these errors etc.

It is a design problem and design problems are in order of magnitude more expensive than any hardware.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 15 Nov 2005 09:49:54 +0100
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

> For information – is your point that we should design the program (using e.g. SPARK) so that there isn't any need for runtime checking?

Yes, when possible.

> (I don't think it likely that you want us not to bother to do any checking!)

We should draw line between "functional" and "non-functional" checks. When checks is an artefact of program/language/environment design then its penalty is more than just run-time overhead. Of course the distinction is not absolute. For example End_Error might look functional, but probably in a better OS with OO interface you will never have it working with a string container, XML document rather than with a raw byte stream. Buffer overflow checks is a clear "non-functional" candidate to me. This is also a reason why I'm skeptical about the design of Unbounded_String and Ada.Container. They don't support safe iteration constructs "for I in X'Range loop".

From: Christopher Browne <cbbrowne@acm.org>
Date: Fri, 25 Nov 2005 00:56:26 -0500
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

> Even in cases where it is critical, how fast does an incorrect program have to be in order to be acceptable? If a really fast, incorrect program is better than a slow, correct program, then I submit the following as the solution to all problems:

```
procedure Solution is
  -- null;
begin – Solution
  null;
end Solution;
```

Compile with all checks suppressed for maximum acceptability.

Reality does *not* lie there.

Something that is only approximately correct but that is super-fast may, in cases where time or computational effort *are* at a premium, be preferable to a slower "correct all the time" program.

The trouble with formal verification methods is that they consume time (for the analysis work) when you may well discover that the problem wasn't specified well enough in the first place for formal verification to actually do any material good.

Having super-well-specified problems is extremely necessary when doing "rocket science;" if it costs $1B to fire off a rocket, and you don't get a second chance, it's necessary to do whatever up-front effort is required to make sure the problem is super-well-specified.

But there are a lot of cases where that level of effort is not warranted, and it's NOT worth getting "super-detailed, super-correct" specifications, and it's NOT worth various of the efforts.

Where the CACM article has some things *right* is that there are plenty of systems where it would be way too costly to reimplement them in a buffer-overflow-immune language. People are not going to redo everything in PL/1 or Ada just because they have better specified string types. They don't have time.

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Sat, 26 Nov 2005 01:31:53 GMT
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

Good point, if they're willing to pay for the losses of others whose systems are hijacked because of buffer-overflow errors injected by their poor language choices. If that were required, I suspect they would suddenly have time.

From: Richard Riehle <adaworks@sbcglobal.net>
Date: Sun, 27 Nov 2005 21:36:01 GMT
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

I am reminded of the well-known comment that "We always have time to do it over but never time to do it right."

In the case of Ada, there is no need to "redo everything." The language is rather friendly to other languages. As for PL/I, there are entirely too many other problems with that language to use it for dependable software.

From: Peter Amey <peter.amey@praxis-cs.co.uk>
Date: Mon, 14 Nov 2005 10:17:21 +0000
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

Of course, using SPARK, we can statically prove the absence of buffer overflows (and many other potential exploits) and thus add precisely nothing in the form of a run-time overhead!

From: Peter Amey <peter.amey@praxis-cs.co.uk>
Date: Tue, 29 Nov 2005 10:48:37 +0000
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

> I though that with SPARK, you have to write your program with no moving parts (or dynamic data structures) and then supply a suite of proofs, which may be quite hard, even with the assistant? How often is this practical?

Obviously how hard it is it depends what it is you are trying to prove. Proving that a SPARK program is free from all run-time errors (of which "buffer overflow" is but one) is remarkably straightforward. A majority of SPARK users are using this technology in an industrial setting on a regular basis.

For exception-freedom proof, you do not have to add extra annotations, the Examiner generates the necessary proof obligations from the language rules of Ada; it essentially generates proof obligations that mirror what the LRM requires for run-time checks. It is necessary to provide some extra information, such as the bounds of the predefined types such as Integer using a configuration file mechanism.

The Simplifier tool, that comes with the SPARK Examiner, usually discharges over 95% of the resulting verification conditions (assuming the code is correct of course!). Sometimes the process prompts the addition of a precondition here and there which is, of course, extremely useful intelligence for future maintenance of the software.

This paper: http://www.praxis-his.com/pdfs/Industrial_strength.pdf gives an overview of the process. The Simplifier hit rate has gone up sharply since it was written but the principles are the same.

From: Stephen Leake <stephen_leake@acm.org>
Date: Sun, 13 Nov 2005 10:44:59 -0500
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

I hope you wrote to the editor of CACM pointing out this flaw in the article. It is up to the editor to ensure that articles are fair and balanced! And they won't know there's a problem if nobody tells them.

There have been similar problems with articles in Dr Dobbs; the editor has been quite gracious when I point them out. We'll have to see if that translates into more balanced coverage in the future [...]

From: Richard Riehle <adaworks@sbcglobal.net>
Date: Mon, 14 Nov 2005 14:40:21 GMT
Subject: Re: Buffer overflow Article
Newsgroups: comp.lang.ada

I did not contact CACM. Rather, I contacted the principal author and informed him about this thread. Perhaps he, or one of the other authors will find the time to post a note in this forum.

Further, given that their complaint was primarily focused on the languages they do know instead of those they don't, I think the article is well-reasoned. The problem, as I see it, is that so many academics spend so much time solving problems while existing solutions are either ignored or unknown to them.

The editor of Dr. Dobbs has a long-standing invitation for articles that describe, in depth and in detail, problems that have been solved in Ada that could not have been just as easily solved in some other language. Further, if someone wants to write an article about Ada 2005, I think Dr. Dobbs will print it – given that it is a well-written article.

From: Marc A. Criley <mc@mckae.com>
Date: Tue, 07 Mar 2006 10:08:20 -0600
Subject: Props to Jean-Pierre!
Newsgroups: comp.lang.ada

Thanks go to Jean-Pierre Rosen for taking the time to write a letter, "Lack of Ada Reflects Software Immaturity", to the editors of "Communications of the ACM" that they published in the March '06 issue.

It was in a response to an article titled "Detection and Prevention of Stack Buffer Overflow Attacks" (11/2005).

Money quote (as they say :-)...

"Many of the problems addressed in the article follow from not using appropriate programming languages, an issue Ada solved more than 20 years ago."

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Wed, 08 Mar 2006 09:19:53 +0100
Organization: Adalog
Subject: Re: Props to Jean-Pierre!
Newsgroups: comp.lang.ada

Yeap. Couldn't resist, when I saw people, in 2006, coming with the bright new idea that the programmer cannot always be trusted, and that it would be good to have the language perform automatically some checks.

Blaming Ada Wrongly

From: Robert Love <rblove@airmail.net>
Date: Thu, 22 Dec 2005 16:25:21 -0000
Subject: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

Let me quote from the December 19th "Space News" article entitled Pentagon Scales Back SBIRS Program. For those who don't know, SBIRS is a multi satellite program to detect missile launches via infra red sensors. It is way behind and billions over budget, mostly due to the sensor and bad initial design.

Air Force Secretary Michael Wynne is quoted as saying:

"One of the biggest problems with SBIRS lies with its operating software, which is based on a programming language called Ada that was developed in the 1970's, Wynne said.

"Ada is a program that is not popular any longer," Wynne said. "It is a software design that was literally invented around the time DOS was invented. DOS is no longer even being talked about nor should Ada be, but we still have Ada-based programmers trying to do it."

The Air Force hopes to use a more modern language like C+ (yes, they used a single +) for SBIRS follow-on system, Air Force Undersecretary Ronald Sega told reporters in a Dec. 15 briefing at the Pentagon."

Lord, there is so much wrong here. Where to start. Is it even worth it to try and educate the Air Force? I suppose I'll try and write one of these bozos once I calm down but I would say this is a huge slam against our favourite language.

From: Richard Riehle <adaworks@sbcglobal.net>
Date: Fri, 23 Dec 2005 00:12:29 GMT
Subject: Re: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

My experience with the USAF decision-makers is that they are doing their best to make responsible decisions on behalf of the National Defence. They are being misinformed by contractors whose self-interest sometimes preempts what ought to be their better judgment.

I must say that I have heard high-ranking DoD people denigrate Ada based on their own experience. One Admiral spoke to a group about how hard it was to teach people Ada in his command. He talked of the credentials of the person hired to do the teaching and noted that, "even this person, with many years of experience in computer science," could not make Ada clear to the students.

There are some unique differences in Ada from other languages. Unless we make those differences clear the students will be discouraged from using it. Too often the language is taught by people who do themselves understand it. To be fair, the same is true of much of what passes for C++ instruction. The difference is that C++ looks like C and it gets a lot of good press. This, in spite of its being one of the most error-prone languages ever to be used by anyone.

From: Steve Whalen <SteveWhalen001@hotmail.com>
Date: 23 Dec 2005 17:12:14 -0800
Subject: Re: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

I'd like to respectfully suggest to those who are going to try to set this record straight that you include these people in your correspondence:

Senator John Warner of Virginia
Senator Carl Levin of Michigan
Senator John McCain of Arizona
Senator Joseph Lieberman of Connecticut

All of the above Senators are members of the Senate Armed Services Committee which has "oversight" of the U.S. military, and their web sites can be found at: http://armed-services.senate.gov/

Since our Senators serve 6 year terms, they are somewhat more insulated from the immediate pressures of daily politics and vendor pressures (but only compared our House of Representatives)....

They are the people who can get such misrepresentations as were made by the Air Force civilian leadership investigated and corrected.

One thing that might be helpful to include in any correspondence to anyone in a position to help, would be a list of questions they could ask which would help them to see that the Air Force's civilian leadership's statements are on their face, incredibly nonsensical to anyone who knows anything about programming or managing large projects.

Questions like:

Were the specifications from which the Ada programming was to be done complete before coding began?

Has anyone independent of the vendor or project management made an assessment of why the project has gone so wrong?

How is it than many of the largest and most complex project have been successfully programmed in Ada?

etc.

From: Robert Love <rblove@airmail.net>
Date: Fri, 23 Dec 2005 17:01:57 -0000
Subject: Re: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

At the level of the Secretary and Undersecretary I expect them to be far removed from direct knowledge of the project but somebody dropped the ball. I expect it is in the USAF Project office. Those managers have an oversight responsibility and since it involves billions of tax payer dollars and national defence it is a task that should not just rely on the contractors information. Did this project have a V&V contractor? I'll have to look.

I do note that Secretary Wynne has previously worked for LockMart, the prime contractor for SBIRS. It may be that he is too cosy with his old employers but I doubt it.

It should be noted that the USAF has several big ticket satellite programs all well over budget and years behind schedule. T-Sat, Space Based Radar and others join SBIRS as projects in trouble. Most of them are due to sensor problems and general poor management. Some should be more like R&D programs that operational projects.

Since the Secretary has brought up software as an issue on this satellite I want to know what was the real cause of the problem. I can't believe Ada is a cause in its own right. Was it compiler/tool problems? Was it a bad software architecture? Was there a valid set of requirements that were stable?

From: Joseph Vlietstra <joevl@earthlink.net>
Date: Sat, 24 Dec 2005 18:49:18 GMT
Subject: Re: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

Some answers about the SBIRS program.

- The SBIRS program was plagued by poor initial systems engineering. Those responsible have been replaced and we were able to get the program somewhat back on track last fall. We can't undo all of the stupid mistakes but I think we can get at least 95% functionality. (The more optimistic think we can score 100%)

- Flight software is written in Ada 95 using Rational Apex compiler. The only problems we've had with the development environment were self-induced (e.g., attempting an Apex/ClearCase integration before it was released by Rational).

- We considered GNAT at the start of the project and contacted ACT. For whatever reason, they weren't interested in developing a GNAT compiler for us. (I don't think they realized that we would play for the development.) In any case, we're happy with Rational Apex.

- We also considered using a GNU C/C++ compiler but it ran slower than the Rational Ada code. This isn't an Ada is faster than C++ claim – Lockheed-Martin spent a lot of money to have a good Ada compiler available; the C++ compiler was an afterthought for the hardware test group.

- There were several subtle hardware glitches that required software fixes. This is a typical problem for a development program. We all learned Chapter 13 of the LRM by heart.

Anyone claiming that Ada was the problem is either ignorant of the circumstances or hoping to obscure the initial systems engineering problems. In fact, the Ada language features allowed us to get as far as we have.

From: Robert Klungle <bklungle@adelphia.net>
Date: Tue, 3 Jan 2006 22:33:43 -0800
Subject: Re: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

Joe presents a good partial description of what happened. There is a lot more to it which had nothing to do with coding. Five major problems leading up to the situation were:

1. Incomplete and incorrect requirements through PDR (which was failed previously.)

2. Incomplete and incorrect design mechanisms leading to individuals designing the same thing more than once, not knowing what the others were doing (multiple CPUs with similar functionality.) Eventually leading to a system which would not perform (loss of messages) at 70% loading. The system was completely redesigned from the ground up in a matter of 5 months. The system is now heading for success (disregarding the usual integration problems.)

3. Management chain failure due to little or no knowledge regarding software development.

4. Incorrect or misleading design specifications on the SBC containing the RH-32(s) from the supplier.

5. Low ball funding and late resource allocation to the project on the part of the government and contractor(s).

I could continue with a very long list but you should get the idea.

Bottom line, Ada had nothing to do with the problem(s), and in fact actually contributed to them having any success at all.

Incidentally, someone mentioned Wynne (in a later posting) casting aspersions on Ada. He is getting his information from others who have a specific agenda to remove Ada from the development list (been hearing it in PDRs and PDAs). There is a general belief that "No one can find any Ada developers. Ada is not being taught in schools. Systems Engineers and Mathematicians coming out of school only know c++ and refuse to learn Ada." Note this is a direct quote from a high level government person, which I took issue with.

The problem of Ada(s) reputation and viability is very big and going down hill rapidly, if observations are any indication.

Note I don't think this is a conspiracy, just a serious case of decisions being made by the wrong people with little or no correct information.

From: David Emery <demery@cox.net>
Date: Thu, 29 Dec 2005 16:47:44 -0500
Subject: Re: SBIRS, Ada and Ignorance
Newsgroups: comp.lang.ada

In my experience, Ada was often blamed for bringing system engineering failures to light before the program was ready hear about it. From a political perspective, being able to demonstrate an inconsistent design at PDR is NOT a good way to get the program through its next major government milestone.

Ada Code Formatting

From: Peter C. Chapin <pchapin@sover.net>
Date: 18 Feb 2006 12:19:59 GMT
Subject: Quick question about Ada code formatting.
Newsgroups: comp.lang.ada

I realize formatting style varies from person to person and from organization to organization. Nevertheless some programming language communities have definite community standards about how certain language constructs should be formatted. For example, it seems universal in this community to name variables This_Way.

When it comes to calling subprograms I've seen some sources that put a space between the name of the subprogram and the argument list.

```
My_Procedure (X, Y, Z);
A := My_Function (B);
```

When wrapping such calls the entire argument list is moved down to the next line.

```
My_Procedure
 (Very_Long, Argument_List,
  With_Many, Arguments);
```

In other communities (C/C++) it is more common to leave the space out and also to leave the opening '(' on the same line as the procedure name.

I'm wondering how universal the above style is among Ada programmers. In other words: would it be desirable for me to adopt it as part of my personal style guide?

P.S. Is there an accepted indentation depth among Ada programmers? I've seen three spaces in several places and I notice both Ada-mode in Emacs and GPS use three spaces by default.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sat, 18 Feb 2006 14:00:46 +0100
Subject: Re: Quick question about Ada code formatting.
Newsgroups: comp.lang.ada

This is quite universal, because of the Ada Quality and Style Guide. Most coding standards I've seen take the AQ&SG as a starting point and recommend, or even mandate, the same indentation style.

Another indentation style which I find quite common is: […]

Three shall be the number of spaces, and the number of spaces shall be three. No more, no less. Two shalt thou not indent, except that thou then proceedest to three. Four is right out.

This is also from the AQ&SG.

From: Simon Wright <simon@pushface.org>
Date: Sat, 18 Feb 2006 13:26:25 +0000
Subject: Re: Quick question about Ada code formatting.
Newsgroups: comp.lang.ada

On my current project we decided to use the formatting provided by GLIDE/GPS and to use the -gnaty switch to check it. You can give a whole slew of options to -gnaty but it seems far easier to just accept the defaults even if they aren't what you would be used to. The 3 space indent is a case in point; of course if your editor just does it for you it's not so hard to accept!

What always baffles me is how people will *not* train themselves to do what the style checker expects, and will *not* just fix the style warnings as they go.

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Sat, 18 Feb 2006 16:36:35 +0100
Subject: Re: Quick question about Ada code formatting.
Newsgroups: comp.lang.ada

A "hand writing" can work for one person, but not for another person. People can, and do fight over whose handwriting viz. Ada style is better. Even neglecting the strong influence that a font can have on appearance. But I think there are much more important issues, and just -gnaty is overly strict here.

Good in my (limited) experience:

- Consistent use of whatever non-typographical convention is in use.

- Comb structure, and then tough may use 4 or 2 spaces or whatever. (so does the GNAT default style in constructs that are continued on the next line, so...) There is no real harm done when one block uses just 2 spaces for the indented lines, and another uses 5. I understand that some people go crazy when they find knife, spoon, and fork not properly aligned, no matter how good or bad the meal is.

But programming has more to do with recipes and food than with the relative angle of eating tools during the "work". Also, play with the image of fish & chips in a paper box, framed by sterling cutlery

- Compare how easily something is looked up in the code versus how easily something is read (left-right, top-down) in the code. These two processes are very different. Their analog in regular texts is table versus paragraph. There will have to be a compromise, then, unless you think that code that looks ordered is code that actually is ordered.

Slightly different handwritings will work. And it makes team mates feel better, because they can adopt some of their very own hand writing style nevertheless, without doing harm to others provided everyone has retained some modicum of flexibility.

An additional proof of -gnaty being overly strict is when it comes function names in GUI programming. Many do not want to see '_'s in identifiers when they are use to them without the underscores. Note the "used to"... :-)

There is a study (from Kent?, comparing MISRA C and other style guides) that demonstrates how useless and wasteful typographic sophistry is.

From: Randy Brukardt <randy@rrsoftware.com>
Subject: Re: Quick question about Ada code formatting.
Date: Mon, 20 Feb 2006 16:10:19 -0600
Newsgroups: comp.lang.ada

> Between subprograms I prefer the "GNAT style" header box:
```
---------------------
-- Process_Whatever --
---------------------
procedure Process_Whatever is
  ...
```
Far clearer than 2 blank lines IMHO.

But extremely hard to maintain. We used a style like that in the Janus/Ada compiler, and we found that we were spending a lot of time lining up the closing hyphens ever time the comment changed in some way. We abandoned the whole idea with Claw, and went to two blank lines between subprograms.

After all, the idea isn't to carry any information, but to simply to separate the procedures. (Comments that repeat what is obviously known by reading the source code are evil, IMHO, so I find this comment wasteful.) Perhaps a better approach would be to include a dashed line separator between procedures – but that takes *three* lines, and since you never have enough screen real estate, so it seems like a waste.

From: Alex R. Mosteo <alejandro@mosteo.com>
Date: Thu, 23 Feb 2006 10:49:43 +0100
Subject: Re: Quick question about Ada code formatting.
Newsgroups: comp.lang.ada

> Yes. One of the projects I'm on (GWindows) also requires this style. After the first couple of days, I wrote an Emacs macro to write the comment for me. Customizable tools are essential.

GPS already has a macro for this, you simply have to give a key binding to it. I personally use Ctrl+B and voilá! subprogram box with null effort.

Otherwise I agree that it would be quite painful.

Conference Calendar

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 *Conference announcements for the international Ada community* at: http://www.cs.kuleuven.ac.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.

2006

April 02-06	4th Symposium on Design, Analysis, and Simulation of Distributed Systems (DASD'2006), Huntsville, Alabama, USA.	
☺ April 04-07	12th IEEE **Real-Time and Embedded Technology and Applications Symposium** (RTAS'2006), San Jose, CA, USA. Topics include: programming languages and software engineering for real-time or embedded systems; middleware for real-time or embedded systems; assessments of real-time and embedded technologies for particular application domains; technology transition lessons learned; etc.	
☺ April 10-11	10th **International Conference on Empirical Assessment in Software Engineering** (EASE'2006), Keele University, Staffordshire, UK. Topics include: any aspect of product and process evaluation and assessment, both qualitative and quantitative.	
April 17-19	13th Annual **European Concurrent Engineering Conference** (ECEC'2006), Athens, Greece. Topics include: Engineering of embedded systems, system development process, specification languages; Diagnostics and maintenance, Automated inspection and quality control; Architectures for building CE systems, CE languages and tools, Distributed computing environments; etc.	
☺ April 18-21	1st **EuroSys Conference** (EuroSys'2006), Leuven, Belgium. Topics include: Systems aspects of Programming language support, Distributed algorithms, Middleware, Parallel and concurrent computing, Embedded computers, Real-time computing, Dependable computing, etc. This should be of interest to the European languages community.	
April 18-21	17th **Australian Software Engineering Conference** (ASWEC'2006), Sydney, Australia. Topics include: Software Design and Patterns; Object-Oriented Software Engineering; Testing, Analysis and Verification; Formal Methods; Software Security, Safety and Reliability; Software Reuse, Components, and Product Line Development; Software Maintenance; Software Engineering Tools; Measurement, Metrics, Experimentation; Technology Transfer, Education; Standards and Legal Issues; etc.	
April 19	19th Conference on Software Engineering Education and Training (CSEET'2006), Oahu, Hawaii, USA.	
	☺ April 18	**Workshop on Secure Software Engineering Education & Training** (WSSEET'2006). Topics include: experience, current situation, and future of education and training in software engineering of (more) secure software
April 23-27	21st ACM **Symposium on Applied Computing** (SAC'2006), Dijon, France	
	☺ April 23-27	Track on **Programming Languages** (PL'2006). Topics include: Compiling Techniques, Formal Semantics and Syntax, Language Design and Implementation, New Programming Language Ideas and Concepts, Practical Experiences with Programming Languages, Program Analysis and Verification, Program Generation and Transformation, Programming Languages from All Paradigms, etc.
	☺ April 23-27	Track on **Object-Oriented Programming Languages and Systems** (OOPS'2006). Topics include: Programming abstractions; Advanced type mechanisms and type safety; Multi-paradigm features; Language features in support of open systems; Program structuring, modularity, generative programming; Distributed Objects and Concurrency; Applications of Distributed Object Computing; etc.
☺ April 24-26	9th IEEE **International Symposium on Object and component-oriented Real-time distributed Computing** (ISORC'2006), Gyeongju, Korea. Topics include: Programming and system engineering	

(ORC paradigms, languages, RT Corba, UML, application programming interface (API), specification, design, verification, validation, testing, maintenance, system of systems, etc.); System software (real-time kernels, middleware support for ORC, extensibility, scheduling, security, etc.); Applications (embedded systems (automotive, avionics, consumer electronics, etc), real-time object-oriented simulations, etc.); System evaluation (worst-case execution time, dependability, fault detection and recovery time, etc.); ...

April 24-28 — 30th Annual IEEE/NASA **Software Engineering Workshop** (SEW-30), Columbia, MD, USA. Topics include: Metrics and experience reports; Software quality assurance; Formal methods and formal approaches to software development; Real-time Software Engineering; Software maintenance, reuse, and legacy systems; etc.

☺ April 25-29 — 20th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2006), Rhodes Island, Greece. Topics include: all areas of parallel and distributed processing; including the development of experimental or commercial systems; applications of parallel and distributed computing; parallel and distributed software, including parallel programming languages and compilers, runtime systems, middleware, libraries, programming environments and tools, etc.

- ☺ April 25-26 — 14th **International Workshop on Parallel and Distributed Real-Time Systems** (WPDRTS'2006). Topics include: Applications, benchmark, and tools; Distributed real-time and embedded middleware; Fault-tolerance and security in real-time systems; Resource management and real-time scheduling; Programming languages and environments; Specification, modeling, and analysis of real-time systems; etc.

April 27 — **Technical Day of Ada-Spain**, Madrid, Spain. The event includes: selected technical presentations; an invited talk; the general assembly and the Ada-Spain award for the best academic project developed in Ada.

May 01-04 — **Systems and Software Technology Conference** (SSTC'2006), Salt Lake City, Utah, USA

May 01-05 — International Conference on Practical Software Quality and Testing (PSQT'2006 West), Las Vegas, NV, USA

May 03-05 — 6th International SPICE Conference on Software Process Improvement and Capability dEtermination (SPICE'2006), Luxembourg, Luxembourg

☺ May 20-28 — 28th International Conference on Software Engineering (ICSE'2006), Shanghai, China

- ☺ May 27-28 — 3rd **International Workshop on Software Development Methodologies of Distributed Systems** (SDMDS2006). Topics include: Fundamental issues and education issues related to distributed systems and technologies; Case studies for large-scale distributed systems development; Tools and practice experiences; Trends in Industry that effect design of distributed systems; etc.

☺ May 22-25 — **DAta Systems In Aerospace** (DASIA'2006), Berlin, Germany

May 25-27 — **International Conference on Dependability of Computer Systems** (DepCos'2006), Szklarska Poreba, Poland. Topics include: General aspects of dependability; Survivable systems; Coding and dependability; Fault tolerant computing; Software dependability; Software testing, validation and verification; etc.

May 28-31 — 6th International Conference on Computational Science (ICCS'2006), Reading, UK

♦ June 05-09 — 11th **International Conference on Reliable Software Technologies - Ada-Europe'2006**, Porto, Portugal. Sponsored by Ada-Europe, in cooperation with ACM SIGAda

☺ June 06-09 — **New Technologies for Distributed Systems** (NOTERE'2006), Toulouse, France. Topics include: software components, distributed architectures, models and tools, semi-formal and formal techniques, verification, etc.

June 08-10 — 2nd **International Conference on Open Source Systems** (OSS'2006), Como, Italy. Topics include: Software engineering perspectives on OSS development, Studies of OSS deployment, etc.

☺ June 10	**PLDI2006** - ACM SIGPLAN **Workshop on Programming Languages and Analysis for Security** (PLAS'2006), Ottawa, Canada. Topics include: Language-based techniques for security; Program analysis and verification (including type systems and model checking) for security properties; Applications, examples, and implementations of these security techniques; etc.
☺ June 11-13	5th IFIP **Working Conference on Distributed and Parallel Embedded Systems** (DIPES'2006), Braga, Portugal. Topics include: Design methodology for distributed and parallel embedded systems, Formal verification of embedded systems, Novel programming techniques for distributed real-time systems, Specific (parallel) architectures for distributed embedded systems, Dependability and fault tolerance of distributed embedded systems, Case studies of distributed embedded systems, etc.
June 12-14	7th **International Conference on Product Focused Software Process Improvement** (PROFES'2006), Amsterdam, The Netherlands. Topics include: Systems and Software Quality, Embedded Systems related Security and Safety, Measurement, SPI in different Software Development Areas, Empirical Studies, Industrial Experiences and Case Studies, Best Practices, Lessons Learned, etc.
June 12-14	**International Workshop on Engineering of Fault-Tolerant Systems** (EFTS'2006), Luxembourg, Luxembourg. Topics include: Software architecture and fault tolerance; OO frameworks and design patterns for fault tolerance; Error handling and fault handling in the software life-cycle; Fault tolerant software development processes; Error recovery through exception handling in the software life-cycle; Design and implementation of fault tolerant distributed systems; etc.
June 12-15	9th **International Conference on Software Reuse** (ICSR-9), Torino, Italy. Topics include: Processes to identify and select OTS components; Integration and evolution problems; Reliability and security of OTS components and legal issues; Software generators and domain-specific languages; Evolution of component-based software systems; Benefit and risk analysis of reuse investments; Generation of non-code artefacts; Quality aspects of reuse, e.g. security and reliability; Success and failure stories of reuse approaches from industrial context; etc.
June 13-16	6th IFIP WG 6.1 **International Conference on Distributed Applications and Interoperable Systems** (DAIS'2006), Bologna, Italy
June 14-16	8th IFIP **International Conference on Formal Methods for Open Object-based Distributed Systems** (FMOODS'2006), Bologna, Italy. Topics include: Semantics and implementation of object-oriented programming and (visual) modelling languages; Formal techniques for specification, design, analysis, verification, validation and testing; Model checking, theorem proving and deductive verification; Model transformations and refactorings; Software architectures; Component-based design; Experience report on best practices and tools; etc.
☺ June 18-21	**Workshop on State-of-the-art in Scientific and Parallel Computing** (PARA'2006), Umea, Sweden. Topics include: software, tools, environments as well as applications for Scientific Computing, High Performance Computing, Parallel Computing, Grid Computing, and Interactive and Scientific Visualization.
June 19-23	15th IEEE **International Symposium on High-Performance Distributed Computing** (HPDC-15), Paris, France. Topics include: Software environments, programming frameworks & language/compiler support; Fault tolerance, reliability and availability for HPDC applications; etc.
June 25-28	2006 **International Conference on Dependable Systems and Networks** (DSN'2006), Philadelphia, PA, USA. Topics include: Dependability Measurement and Analysis; Fault-Tolerance in Distributed and Real-Time Systems; Safety-Critical Systems; Software Reliability; Software Testing, Validation, and Verification; etc. Deadline for submissions: May 1, 2006 (demonstrations)
	☺ June 27 — **Workshop on Architecting Dependable Systems** (WADS'2006). Topics include: dependability modeling in software architectures; run-time checks of architectural models; dependability evaluation in software architectures; architectural patterns for dependable systems; exception handling in software architectures; dependable architectures and implementation; etc.
June 26-28	11th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'2006), Bologna, Italy.
June 26-29	The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing (WORLDCOMP'2006), Las Vegas, Nevada, USA

☺ June 26-29 **International Conference on Programming Languages and Compilers** (PLC'2006). Topics include: Design and processing of domain specific languages; Implementation of languages features; Language support for security and safety; Compiler construction techniques for modern systems; Program representation & Program analysis; Program optimizations and transformations techniques; Interaction between compilers and architectures; Compilation for distributed, concurrent, and heterogeneous systems; Languages and compilers for high performance computing; Object oriented programming techniques; Object-oriented languages; Run-time environment and storage management techniques; Compilation and interpretation techniques; Code generation and code optimization techniques for modern programming languages; Compilation techniques for embedded code; Security and safety techniques at compiler level; Design of novel language constructs and tool supports; etc.

☺ June 26-29 **International Conference on Parallel and Distributed Processing Techniques and Applications** (PDPTA'2006). Topics include: Parallel/Distributed applications; Reliability and fault-tolerance; Real-time and embedded systems; Object oriented technology and related issues; Software tools and environments for parallel and distributed platforms: operating systems, compilers, languages, debuggers, monitoring tools, software engineering on parallel/distributed systems, ...; etc.

☺ June 26-29 **International Conference on Real-Time Computing Systems & Applications** (RTCOMP'2006). Topics include: Software engineering for real-time computing and systems; Software technologies (real-time operating systems, middleware and distributed technologies, compiler support, component-based technologies, ...); Fault-tolerance; Embedded systems and ubiquitous computing; Distributed systems; Programming languages and run-time systems; Scheduling algorithms and analysis; Real-time kernel support; Real-time & embedded distributed algorithms & systems; Simulation of real-time systems; Real-time middleware systems; Object oriented methods for real-time systems; etc.

☺ June 26-29 **International Conference on Software Engineering Research and Practice** (SERP'2006). Topics include: Object-oriented technology (design & analysis); Software engineering methodologies; Reliability; Distributed and parallel systems; Legal issues and standards; High assurance software systems; Evolution and maintenance; Component-based software engineering; Software engineering standards; Interoperability; Software reuse; Verification, validation and quality assurance; Programming languages; Education (software engineering curriculum design); Novel software tools and environments; Real-time software engineering; Critical and embedded software design; Quality management; Software design and design patterns; Case studies; etc.

June 27-30 6th **International Conference on Application of Concurrency to System Design** (ACSD'2006), Turku, Finland. Topics include: design of complex concurrent systems, correct-by-construction design methods and integration of verification techniques with the design process, etc.

☺ July 02 5th **International Workshop on Constructive Methods for Parallel Programming** (CMPP'2006), Kuressaare, Estonia. Topics include: formal models, methods, and languages for parallel programming; parallelization and compilation techniques; parallel and distributed object-oriented programming; hardware-software codesign; etc. Deadline for registration: May 15, 2006

☺ July 03-07 20th **European Conference on Object-Oriented Programming** (ECOOP'2006), Nantes, France. Topics include: Patterns, Modularity, Adaptability, Separation of Concerns, Components, Frameworks, Concurrency, Real-time, Embedded, Distribution, Domain Specific Languages, Language Workbenches, Multi-paradigm Languages, Language Innovations, Compilation, Methodology, Practices, Metrics, Formal methods, Tools, etc. Deadline for submissions: May 5, 2006 (posters and demos) Deadline for early registration: May 23, 2006

☺ July 03 **Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems** (ICOOOLPS'2006). Topics include: implementation of fundamental OOL features: inheritance (object layout, late binding, subtype test, ...), genericity (parametric types), memory management; runtime systems: compilers, linkers, etc; optimizations: static and dynamic analyses, etc; resource constraints: real-time systems, etc: relevant choices and tradeoffs: separate compilation vs. global compilation, dynamic checking vs. proof-carrying code, etc.

☺ July 03 — 10th **Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts**. Topics include: experiences, ideas and resources to support the teaching and learning of basic object-oriented concepts.

☺ July 03 — 6th **Workshop on Parallel/High-Performance Object-Oriented Scientific Computing** (POOSC'2006). Topics include: tried or proposed programming language alternatives to C++; issues specific to handling or abstracting parallelism; etc.

☺ July 04 — 3rd **International Workshop on Practical Problems of Programming in the Large** (PPPL'2006). Topics include: Experience, positive or negative with technology transfer and cooperation of academia and industry; Negative results: what went wrong although it should have worked according to software engineering folklore; Success-stories for component-based software engineering; Keeping systems with large amounts of classes / objects / modules / components organised; Refactoring, Software Evolution and Migration; etc.

☺ July 05-07 — 18th **Euromicro Conference on Real-Time Systems** (ECRTS'2006), Dresden, Germany. Topics include: all aspects of real-time systems; special focus on industrial applications of real-time technology; compiler support; component-based approaches; middleware and distribution technologies; programming languages; real-time operating systems; model-driven development of embedded RT systems; formal methods; reliability, security and survivability in RT systems; scheduling and schedulability analysis; worst-case execution time analysis; validation techniques; etc.

July 09-16 — 33rd **International Colloquium on Automata, Languages and Programming** (ICALP'2006), Venice, Italy. Topics include: Principles of Programming Languages, Formal Methods, Models of Concurrent and Distributed Systems, Program Analysis and Transformation, etc.

☺ July 10-13 — OMG **Workshop on Distributed Object Computing for Real-time and Embedded Systems**, Washington, DC, USA. Topics include: Real-time systems; Embedded systems; Fault-tolerant systems; High-availability systems; Safety-critical systems; Design tools for real-time distributed systems; Real-time middleware, including real-time CORBA; Modeling notations, including UML; Model-Driven approaches, including MDA; High-level real-time programming models; etc.

July 10-14 — 2nd **European Conference on Model Driven Architecture: Foundations and Applications** (ECMDA-FA'2006), Bilbao, Spain. Topics include: Model Transformation - languages, tools; MDA for Large Scale Distributed Systems; Comparative studies of MDA tools; MDA for Legacy Systems; MDA for systems engineering; MDA for embedded systems; MDA for high-integrity systems (safety-critical and security-critical systems; etc.

☺ July 12-15 — 12th **International Conference on Parallel and Distributed Systems** (ICPADS'2006), Minneapolis, Minnesota, USA. Topics include: Parallel and Distributed Applications and Algorithms; Reliable and Fault-Tolerant Computing; Real-Time Systems; Tools; etc.

☺ July 23-26 — 25th Annual ACM SIGACT-SIGOPS **Symposium on Principles of Distributed Computing** (PODC'2006), Denver, Colorado, USA. Topics include: Concurrent programming, Distributed systems and middleware platforms, Correctness and verification of distributed and parallel programming, etc.

☺ August 14-18 — 35th **International Conference on Parallel Processing** (ICPP'2006), Columbus, Ohio, USA. Topics include: findings in any aspects of parallel and distributed computing; such as Compilers and Languages, Systems Support for Parallel and Distributed Applications, etc.

August 21-27 — 14th **International Symposium of Formal Methods Europe** (FM'2006), Hamilton, Canada. Topics include: Tools for formal methods (tool support and software engineering, environments for formal methods), Formal methods in practice (experience with introducing formal methods in industry, case studies), etc. Deadline for submissions: May 26, 2006 (posters, tools, doctoral symposium)

☺ August 26-27 — 11th **International Workshop on Formal Methods for Industrial Critical Systems** (FMICS'2006), Bonn, Germany. Deadline for submissions: May 26, 2006 (abstracts), June 2, 2006 (full papers)

☺ Aug 29-Sept 01 — 12th **International Conference on Parallel and Distributed Computing** (Euro-Par'2006), Dresden, Germany. Topics include: the promotion and advancement of parallel computing; Support Tools and

	Environments; Distributed Systems and Algorithms; Parallel Programming: Models, Methods, and Languages; Embedded Parallel Systems; etc.
☺ September 13-15	7th **Joint Modular Languages Conference** (JMLC'2006), Oxford, England. Topics include: programming language design and implementation; software tools and environments; software quality and testing; formal methods in modular and composable software development; modularity and composability in parallel and distributed systems; modularity and composability in safety-critical and real-time systems; software engineering education; case studies aligning with any of the above; etc.
☺ September 16-20	**Parallel Computing Technologies** (PaCT'2006), Seattle, Washington. USA. Topics include: Compilers and tools for parallel computer systems, Parallel programming languages and applications, Run time system support for parallel systems, Parallel processing in type safe languages, Support for correctness in hardware and software (esp. with concurrency), etc.
☺ September 18-20	20th **International Symposium on DIStributed Computing** (DISC'2006), Stockholm, Sweden. Topics include: concurrent programming and synchronization algorithms; fault tolerance; specification, semantics, and verification; distributed programming languages; distributed object-oriented computing; etc. Deadline for submissions: May 1, 2006
☺ September 20-22	**Real-Time and Networked Embedded Systems** Track of the 11th IEEE *International Conference on Emerging Technologies and Factory Automation* (RTNES-EFTA'2006), Prague, Czech Republic. Topics include: Real-time (distributed) embedded systems; Dependable embedded systems; Formal methods; Real-time executives and operating systems; Real-time scheduling; Real-time components and Middleware; Software engineering and programming languages; Case studies (industrial automation, automotive, avionics, communications...); etc.
September 25-28	26th IFIP WG 6.1 **International Conference on Formal Techniques for Networked and Distributed Systems** (FORTE'2006), Paris, France. Topics include: Practical experience with formal methods, etc. Deadline for submissions: April 10, 2006 (short abstracts), April 18, 2006 (papers)
October 01-06	9th **International Conference on Model-Driven Engineering Languages and Systems** (MoDELS'2006), Genoa, Italy. Topics include: Model-driven engineering methodologies, approaches, languages and tools; Domain-specific modeling languages; Programming language and meta-programming support for linking models to code; Modeling languages and tools; Semantics of modeling languages; Modeling and analysis of real-time, embedded, and distributed systems; etc. Deadline for submissions: May 8, 2006 (workshops, tutorials)
☺ October 02-04	25th IEEE **International Symposium on Reliable Distributed Systems** (SRDS'2006), Leeds, UK. Topics include: reliability, availability, safety, security, and real time; Security and 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. Deadline for submissions: April 24, 2006
☺ October 12-13	**Automotive - Safety & Security 2006**, Stuttgart, Germany. Theme: "Sicherheit und Zuverlässigkeit für automobile Informationstechnik". Organized by Gesellschaft für Informatik (GI), etc., in cooperation with Ada-Deutschland and Fachgruppe "Ada", etc. Topics include (in German): Zuverlässigkeit und Sicherheit für fahrbetriebs-kritische Software und IT-Systeme; Sichere Entwicklung, Aktualisierung und Freischaltung; Normen und Standardisierungsbestrebungen; Entwicklungsbegleitende Evaluation und Zertifizierung; etc. Deadline for submissions: April 15, 2006
☺ October 22-26	21st Annual **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2006), Portland, Oregon, USA. Topics include: diverse disciplines related to object technology. Deadline for submissions: June 30, 2006 (DesignFestR Posters, Onward! Films, Student Research Competition, Demonstrations, Doctoral Symposium), August 1, 2006 (Student Volunteers)
October 23-27	13th **Working Conference on Reverse Engineering** (WCRE'2006), Benevento, Italy. Theme: "Empirically Assessing Reverse Engineering Techniques and Tools". Topics include: Empirical studies in reverse engineering; Decompilation and binary translation; Redocumenting legacy systems; Reverse engineering tool support; Mining software repositories; Program analysis and slicing; Software architecture recovery; Program transformation and refactoring; etc. Deadline for submissions: June 7, 2006 (papers), June 20, 2006 (special tracks, workshops, tutorials)

October 25-27	5th **International Conference on Software Methodologies Tools, and Techniques** (SoMeT'2006), Quebec, Canada. Topics include: Software methodologies, and tools for robust, reliable, non-fragile software design; Automatic software generation versus reuse, and legacy systems, source code analysis and manipulation; Software evolution techniques; Formal methods for software design; Static and dynamic analysis, and software maintenance; Formal techniques for software representation, software testing and validation; Software reliability, and software diagnosis systems; etc. Deadline for submissions: May 15, 2006
☺ Oct 29-Nov 03	8th **International Symposium on Distributed Objects and Applications** (DOA'2006), Montpellier, France. Topics include: Application case studies of distribution technologies; Component-based software development; Design patterns for distributed systems; Integrated development environments; Middleware for distributed object computing; Real-time solutions for distributed objects; Technologies for reliability and fault-tolerance; Testing and validation of distributed object systems; etc. Deadline for submissions: May 15, 2006
Oct 30-Nov 03	8th **International Conference on Formal Engineering Methods** (ICFEM'2006), Macao SAR, China. Topics include: Abstraction and refinement; Tool development and integration for formal system design, analysis and verification; Integration of formal verification tools in CASE tools; Techniques for specification, verification and validation; Techniques and case studies for correctness by construction; Experiments of verified systems; Application in real-time, hybrid and critical systems; Emerging technologies; etc. Deadline for submissions: May 12, 2006 (papers), June 30, 2006 (tutorials)
♦ Nov 12-16	**2006 ACM SIGAda Annual International Conference** (SIGAda'2006), Albuquerque, New Mexico, USA. Sponsored by ACM SIGAda, in cooperation with SIGAPP, SIGCAS, SIGCSE, SIGPLAN, SIGSOFT, Ada-Europe, and Ada Resource Association (ACM approval pending, Cooperation approvals pending.) Topics include: reliability needs and styles; safety and high integrity issues; analysis, testing, and validation; standards; use of ASIS for new Ada tool development; mixed-language development; Ada in XML and .NET environments; quality assurance; Ada & software engineering education; commercial Ada applications: what Ada means to the bottom line; static and dynamic code analysis; software architecture and design; etc. Deadline for submissions: May 16, 2006 (technical articles, extended abstracts, experience reports, workshops, panel sessions, and tutorials)
☺ December 01-04	4th **International Symposium on Parallel and Distributed Processing and Applications** (ISPA'2006), Sorrento, Italy. Topics include: Parallel/distributed system architectures; Tools and environments for software development; Parallel/distributed algorithms; Distributed systems and applications; Reliability, fault-tolerance, and security; etc. Includes "Languages and Algorithms" and "Software and Applications" Tracks. Deadline for submissions: May 1, 2006 (workshops), May 31, 2006 (papers)
☺ December 05-08	27th IEEE **Real-Time Systems Symposium** (RTSS'2006), Rio de Janeiro, Brazil. Topics include: all aspects of real-time systems design, analysis, implementation, evaluation, and case-studies. Deadline for submissions: May 19, 2006
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

2007

June	12th Annual **Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2007), Dundee, Scotland, UK
☺ June 09-16	3rd **History of Programming Languages Conference** (HOPL-III), San Diego, CA, USA. Co-located with FCRC'2007. Deadline for submissions: August 2006 (reworked full papers)
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

2008

June	13th Annual **Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2008), Madrid, Spain

Rationale for Ada 2005: Epilogue

John Barnes

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

Abstract

This is the last of a number of papers describing the rationale for Ada 2005. In due course it is anticipated that the papers will be combined (after appropriate reformatting and editing) into a single volume for formal publication.

This last paper summarizes a small number of general issues of importance to the user such as compatibility between Ada 2005 and Ada 95. It also briefly considers a few potential changes that were considered for Ada 2005 but rejected for various reasons.

Keywords: rationale, Ada 2005.

1 Compatibility

There are two main sorts of problems regarding compatibility. These are termed Incompatibilities and Inconsistencies.

An incompatibility is a situation where a legal Ada 95 program is illegal in Ada 2005. These can be annoying but not a disaster since the compiler automatically detects such situations.

An inconsistency is where a legal Ada 95 program is also a legal Ada 2005 program but might have a different effect at execution time. These can in principle be really nasty but typically the program is actually wrong anyway (in the sense that it does not do what the programmer intended) or its behaviour depends upon the raising of a predefined exception (which is generally considered poor style) or the situation is extremely unlikely to occur.

As mentioned below in Section 2, during the development of Ada 2005 a number of corrections were made to Ada 95 and these resulted in some incompatibilities and inconsistencies with the original Ada 95 standard. These are not considered to be incompatibilities or inconsistencies between Ada 95 and Ada 2005 and so are not covered in this section.

1.1 Incompatibilities with Ada 95

Each incompatibility listed below gives the AI concerned and the paragraph in the AARM which in some cases will give more information. Where relevant, the section in this rationale where the topic is discussed is also given. Where appropriate the incompatibilities are grouped together.

1 – The words **interface**, **overriding** and **synchronized** are now reserved. Programs using them as identifiers will need to be changed. (AI-284, 2.9(3.c))

This is perhaps the most important incompatibility in terms of visibility to the average programmer. It is discussed in paper 1 section 2.

2 – If a predefined package has additional entities then incompatibilities can arise. Thus suppose the predefined package Ada.Stuff has an additional entity More added to it. Then if an Ada 95 program has a package P containing an entity More then a program with a use clause for both Ada.Stuff and P will become illegal in Ada 2005 because the reference to More will become ambiguous. This also applies if further overloadings of an existing entity are added.

Because of this there has been reluctance to extend existing packages but a preference to add child packages. Nevertheless in some cases extending a package seemed more appropriate especially if the identifiers concerned are unlikely to have been used by programmers.

The following packages have been extended with additional entities as listed.

Ada.Exceptions – Wide_Exception_Name, Wide_Wide_Exception_Name. (AI-400, 11.4.1(19.bb))

Ada.Real_Time – Seconds, Minutes. (AI-386, D.8(51.a))

Ada.Strings – Wide_Wide_Space. (AI-285, A.4.1(6.a))

Ada.Strings.Fixed – Index, Index_Non_Blank. (AI-301, A.4.3(109.a))

Ada.Strings.Bounded – Set_Bounded_String, Bounded_Slice, Index, Index_Non_Blank. (AI-301, A.4.4(106.f))

Ada.Strings.Unbounded – Set_Unbounded_String, Unbounded_Slice, Index, Index_Non_Blank. (AI-301, A.4.5(88.c))

There are similar additions to Ada.Strings.Wide_Fixed, Ada.Strings.Wide_Bounded and Ada.Strings.Wide_Unbounded. (AI-301, A.4.7(48.a))

Ada.Tags – No_Tag, Parent_Tag, Interface_Ancestor_Tags, Descendant_Tag, Is_Descendant_At_Same_Level, Wide_Expanded_Name, Wide_Wide_Expanded_Name. (AI-260, 344, 400, 405, 3.9(33.d))

Ada.Text_IO – Get_Line. (AI-301, A.10.7(26.a))

Interfaces.C – char16_t, char32_t and related types and operations. (AI-285, B.3(84.a))

It seems unlikely that existing programs will be affected by these potential incompatibilities.

3 – If a subprogram has an access parameter (without a null exclusion) and is not a dispatching operation then it cannot be renamed as a dispatching operation in Ada 2005 although it can be so renamed in Ada 95. See paper 2, section 2 for an example. (AI-404, 3.9.2(24.b))

4 – As discussed in paper 2 section 5, there are many awkward situations in Ada 95 regarding access types, discriminants and constraints. One problem is that some components can change shape or disappear. The rules in Ada generally aim to prevent such components from being accessed or renamed. However, in Ada 95, some entities don't look constrained but actually are constrained. The consequence is that it is difficult to prevent some constrained objects from having their constraints changed and this can cause components to change or disappear even though they might be accessed or renamed.

A key rule in Ada 95 was that aliased variables were always constrained with the intent that that would solve the problems. But loopholes remained and so the rules have been changed considerably. Aliased variables are not necessarily constrained in Ada 2005 and other rules now disallow certain constructions that were permitted in Ada 95 and this gives rise to a number of minor incompatibilities.

If a general access subtype refers to a type with default discriminants then that access subtype cannot have constraints in Ada 2005. Consider

```
type T(Disc: Boolean := False) is
  record
   ...
  end record;
```

The discriminated type T has a default and so unconstrained objects of type T are mutable. Suppose we now have

```
type T_Ptr is access all T;
subtype Sub_True_T_Ptr is T_Ptr(Disc => True);
                              -- subtype illegal in Ada 2005
```

The type T_Ptr is legal in both Ada 95 and Ada 2005 of course, but the subtype Sub_True_T_Ptr is only legal in Ada 95 and not in Ada 2005. The reason why the subtype cannot be permitted is illustrated by the following

```
Some_T: aliased T := (Disc => True, ...);
A_True_T: Sub_True_T_Ptr := Some_T'Access;
...
Some_T := (Disc => False, ...);
```

When Some_T'Access is evaluated there is a check that the discriminant has the correct value so that A_True_T is assigned a valid value. But the second assignment to Some_T means that the discriminant changes and so A_True_T would no longer have a valid value.

In Ada 95, all aliased variables were considered constrained and so the second assignment would not have been permitted anyway. But, as mentioned above, aliased variables are not considered to be constrained in Ada 2005 just because they are aliased.

Note that there is no similar restriction on types; thus we can still write

```
type True_T_Ptr is access all T(Disc => True);
```

because any conversion which might cause difficulties is forbidden as explained in one of the examples below.

The restriction on subtypes does not apply if the discriminants do not have defaults, nor to pool-specific types. (AI-363, 3.7.1(15.c))

Since aliased variables are not necessarily constrained in Ada 2005 there are situations where components might change shape or disappear in Ada 2005 that could not happen in Ada 95. Applying the Access attribute to such components is thus illegal in Ada 2005. Suppose the example above has components as follows

```
type T(Disc: Boolean := False) is
  record
    case Disc is
      when False =>
        Comp: aliased Integer;
      when True =>
        null;
    end case;
  end record;
```

Since objects of type T might be mutable, the component Comp might disappear.

```
type Int_Ptr is access all Integer;
Obj: aliased T;                      -- mutable object
Dodgy: Int_Ptr := Obj.Comp'Access;        -- take care
...
Obj:= (Disc => True);                     -- Comp gone
```

In Ada 95, the assignment to Dodgy is permitted but then the assignment to Obj raises Constraint_Error because there might be dodgy pointers.

In Ada 2005, the assignment statement to Dodgy is illegal since we cannot write Obj.Comp'Access. The assignment to Obj is itself permitted because we now know that there cannot be any dodgy pointers.

See (AI-363, 3.10.2(41.b)). Similarly, renaming an aliased component such as Comp is also illegal. (AI-363, 8.5.1(8.b))

There are related situations regarding discriminated private types where type conversions and the Access attribute are forbidden. Suppose we have a private type and an access type and that the full type is in fact the discriminated type above thus

```
package P is
  type T is private;
  type T_Ptr is access all T;
  function Evil return T_Ptr;
  function Flip(Obj: T) return T;
```

```
private
  type T(Disc: Boolean := False) is
    record
      ...
    end record;
  ...
end P;

package body P is

  type True_T_Ptr is access all T(Disc => True);
  subtype Sub_True_T_Ptr is T_Ptr(Disc => True);
                 -- legal in Ada 95, illegal in Ada 2005

  True_Obj: aliased T(Disc => True);
  TTP: True_T_Ptr := True_Obj'Access;
  STTP: Sub_True_T_Ptr := True_Obj'Access;

  function Evil return T_Ptr is
  begin
    if ... then
      return T_Ptr(TTP);            -- OK in 95, not in 2005
    elsif ... then
      return True_Obj'Access;    -- OK in 95, not in 2005
    else
     return STTP;
    end if;
  end Evil;

  function Flip(Obj: T) return T is
  begin
    case Obj.Disc is
      when True => return (Disc => False, ...);
      when False => return (Disc => True, ...);
    end case;
  end Flip;

end P;
```

The function Evil has three branches illustrating various possible ways of returning a value of the type T. The function Flip just returns a value of the type T with opposite discriminants to the parameter. Now consider

```
with P;  use P;
procedure Do_It is
  A: T;
  B: T_Ptr := new T;
  C: T_Ptr := Evil;
begin
  A := Flip(A);
  B.all := Flip(B.all);
  C.all := Flip(C.all);
end Do_It;
```

This declares an object A of type T and then two objects B and C of the access type T_Ptr and initializes them in different ways. Finally it attempts to change the discriminant of the three objects by calling the function Flip.

In Ada 95 all objects on the heap are constrained. This means that clients cannot change the discriminants even if they do not know that they exist. So the assignment to B.**all** raises Constraint_Error since B.**all** is on the heap and thus constrained whereas the assignment to A is fine since A is not constrained. However, from the client's point of view they both really do the same thing and so the behaviour is very curious. Remember that the client doesn't know about the discriminants and so both operations look the same in the abstract. This is unfortunate and breaks privacy which is sinful. There is a similar example in paper 2, section 5 where we try to change Chris but do not know that the new value has a beard and this fails because Chris is female.

To prevent such privacy breaking the rules are changed in Ada 2005 so that objects on the heap are unconstrained in this one case. So the assignments to B.**all** and C.**all** do not have checks on the discriminant. As a consequence Evil must not return an object which is constrained otherwise the assignment to C would result in True_Obj having its discriminant turned to False.

All three possible branches in Evil are prevented in Ada 2005. The conversion in the first branch is forbidden and the Access attribute in the second branch is forbidden. In the case of the third branch the return itself is acceptable in principle because STTP is of the correct type. However, this is prevented by the rule mentioned above since the subtype Sub_True_T_Ptr is itself forbidden and so the object STTP could not be declared in the first place.

See (AI-363, 3.10.2(41.e) and 4.6(71.k)).

5 – Aggregates of limited types are permitted in Ada 2005 as discussed in paper 3, section 5. This means that in obscure situations an aggregate might be ambiguous in Ada 2005 and thus illegal. Consider

```
type Lim is limited
  record
    Comp: Integer;
  end record;

type Not_Lim is
  record
    Comp: Integer;
  end record;

procedure P(X: LIm);
procedure P(X: Not_Lim);

P((Comp => 123));                    -- illegal in Ada 2005
```

In Ada 95, the aggregate cannot be of a limited type and so the type Lim is not considered for resolution. But Ada 2005 permits aggregates of limited types and so the aggregate is ambiguous. (AI-287, 4.3(6.e))

Another similar situation with limited types and nonlimited types concerns assignment. Again this relates to the fact that limitedness is no longer considered for name resolution. Consider

```
type Acc_Not_Lim is access Not_Lim;
function F(X: Integer) return Acc_Not_Lim;
type Acc_Lim is access Lim;
function F(X: Integer) return Acc_Lim;
F(1).all := F(2).all;                -- illegal in Ada 2005
```

In Ada 95, only the first F is considered for name resolution and the program is valid. In Ada 2005, there is an ambiguity because both functions are considered. Note of course that the assignment for the limited function is still illegal anyway but the compiler meets the ambiguity first. Clearly this is an obscure situation. (AI-287. 5.2(28.d))

6 – Because of the changes to the fixed-fixed multiplication and division rules there are situations where a legal program in Ada 95 becomes illegal in Ada 2005. Consider

```
package P is
  type My_Fixed is delta ... ;
  function "*" (L, R: My_Fixed) return My_Fixed;
end P;

use P;
A, B: My_Fixed;
D: Duration := A * B;              -- illegal in Ada 2005
```

Although this is legal in Ada 95, the new rule in Ada 2005 says that if there is a user-defined operation involving the type concerned then the predefined operation cannot be used unless there is a type conversion or we write Standard."*"(...).

So in Ada 2005 a conversion can be used thus

```
D: Duration := Duration(A * B);
```

See paper 5, section 3. (AI-364, 4.5.5(35.d))

7 – The concept of return by reference types has gone. Instead the user has to explicitly declare a function with an anonymous access type as the return type. This only affects functions that return an existing limited object such as choosing a task from among a pool of tasks. See paper 3 section 5 for an example. (AI-318, 6.5(27.g))

8 – There is a very curious situation regarding exporting multiple homographs from an instantiation that is now illegal. This is a side effect of adding interfaces to the language. (AI-251, 8.3(29.s))

9 – The introduction of more forms of access types has changed the rules regarding name resolution. Consider the following contrived example

```
type Cacc is access constant Integer;
procedure Proc(Acc: access Integer);
procedure Proc(Acc: Cacc);
List: Cacc := ... ;
...
Proc(List);               -- illegal in Ada 2005
```

In Ada 95 the call of Proc is resolved because the parameters Acc are anonymous access to variable in one case and access to constant in the other. In Ada 2005, the name resolution rules do not take this into account so it becomes ambiguous and thus illegal which is a good thing because it is likely that the Ada 95 programmer made a mistake anyway. (AI-409, 8.6(34.n))

10 – In Ada 2005, a procedure call that might be an entry is permitted in timed and conditional entry calls. See paper 4, section 3. In Ada 95, a procedure could not be so used and this fact is used in name resolution in Ada 95 but does not apply in Ada 2005. Hence if a procedure and an entry have the same profile then an ambiguity can exist in Ada 2005. (AI-345, 9.7.2(7.b))

11 – It is now illegal to have an allocator for an access type with Storage_Size equal to zero whereas in Ada 95 it raised Storage_Error on execution. It is always better to detect errors at compile time wherever possible. The reason for the change is to allow Pure units to use access types provided they do not use allocators. If the storage size is zero then this is now known at compile time. (AI-366, 4.8(20.g))

12 – The requirement that a partial view with available stream attributes be externally streamable can cause an incompatibility in extremely rare cases. This also relates to pragma Pure. (AI-366, 10.2.1(28.e))

13 – It is now illegal to use an incomplete view as a parameter or result of an access to subprogram type or as an access parameter of a primitive operation if the completion is deferred to the package body. See paper 3, section 2 for examples. (AI-326, 3.10.1(23.h, i))

14 – The specification of System.RPC can now be tailored for an implementation by adding further operations or by changing the profile of existing operations. If it is tailored in this way then an existing program might not compile in Ada 2005. See paper 6, section 7. (AI-273, E.5(30.a))

1.2 Inconsistencies with Ada 95

1 – The awkward situations regarding access types, discriminants and constraints discussed in paper 2 section 5, can also give rise to obscure inconsistencies.

Unconstrained aliased objects of types with discriminants with defaults are no longer constrained by their initial values. This means that a program that raised Constraint_Error in Ada 95 because of attempting to change the discriminants will no longer do so.

Thus consider item 4 in the previous section. We had

```
type Int_Ptr is access all Integer;
Obj: aliased T;                        -- mutable object
Dodgy: Int_Ptr := Obj.Comp'Access;        -- take care
...
Obj:= (Disc => True);                     -- Comp gone
```

We noted that in Ada 2005, the assignment statement to Dodgy is illegal because we cannot write Obj.Comp'Access. The assignment to Obj is itself permitted because we now know that there cannot be any dodgy pointers. Suppose that the assignment to Dodgy is removed. Then in Ada 95, the assignment to Obj will raise Constraint_Error but it will not in Ada 2005. It is extremely unlikely that any correct program relied upon this behaviour. (AI-363, 3.3.1(33.f) and 3.10(26.d))

A related situation applies with allocators where the allocated type is a private type with hidden discriminants. This is also illustrated by an earlier example where we had

```
with P;  use P;
procedure Do_It is
   A: T;
   B: T_Ptr := new T;
   C: T_Ptr := Evil;
begin
   A := Flip(A);
   B.all := Flip(B.all);      -- C_E in Ada 95, not in 2005
   C.all := Flip(C.all);
end Do_It;
```

The assignment to B.**all** raises Constraint_Error in Ada 95 but not in Ada 2005 as explained above. Again it is extremely unlikely that any correct program relied upon this behaviour. (AI-363, 4.8(20.f))

2 – In Ada 2005 the categorization of certain wide characters is changed. As a consequence Wide_Character'Wide_Value and Wide_Character'Wide_Image will change in some rare situations. A further consequence is that for some subtypes S of Wide_Character the value of S'Wide_Width is different. But the value of Wide_Character'Wide_Width itself is not changed. (AI-285, 3.5.2(9.h) and AI-395, 3.5.2(9.i, j))

3 – There is an interesting analogy to incompatibility number 2 which concerns adding further entities to existing predefined packages. If we add further entries to Standard itself then an inconsistency is possible. Thus if an additional entity More is added to the package Standard and an existing program has a package P with an existing entity More and a use clause for P then, in Ada 2005, references to More will now be to that in Standard and not that in P. In the most unlikely event that the program remains legal, it will behave differently. The only such identifiers added to Standard are Wide_Wide_Character and Wide_Wide_String so this is extremely unlikely. (AI-285, 3.5.2(9.k) and 3.6.3(8.g))

4 – Access discriminants and non-controlling access parameters no longer exclude null in Ada 2005. A program that passed null to these will behave differently.

The usual situation is that Constraint_Error will be raised within the subprogram when an attempt to dereference is made rather than at the point of call. If the subprogram has no handler for Constraint_Error then the final effect will be much the same.

But clearly it is possible for the behaviour to be quite different. For example, the access value might not be dereferenced or the subprogram might have a handler for Constraint_Error which does something unusual. And there might even be a pragma Suppress for the check in which case the program will become erroneous.

See paper 2, section 2 for an example. (AI-231, 3.10(26.c))

5 – The lower bound of strings returned by functions Expanded_Name and External_Name (and wide versions) in Ada.Tags are defined to be 1 in Ada 2005. Ada 95 did not actually define the value and so if an implementation has chosen to return some other lower bound such as 77 then the program might behave differently. (AI-417, 3.9(33.c)) See also 2.2 item 4 below.

6 – The upper bound of the range of Year_Number in Ada 2005 is 2399 whereas it was 2099 in Ada 95. See paper 6, section 3. (AI-351, 9.6(40.e))

2 Retrospective changes to Ada 95

In the course of the development of Ada 2005, a number of small changes were deemed to apply also to Ada 95 and thus were classified as binding interpretations rather than amendments. Accordingly they are not (generally) covered by the changes discussed in the previous papers. Note however, that AI-241 on exceptions was discussed in paper 5 even though it was eventually classified as a binding interpretation. Moreover, AI-329 on exceptions was split and the part stating that Raise_Exception never returns (also applying to Ada 95) was formed into AI-446.

AI-438 adds subprograms Read_Exception_Occurrence and Write_Exception_Occurence plus corresponding attribute definition clauses for streams to the package Ada.Exceptions thus

```
procedure Read_Exception_Occurrence
   (Stream: not null access Root_Stream_Type'Class;
    Item: out Exception_Occurrence);

procedure Write_Exception_Occurrence
   (Stream: not null access Root_Stream_Type'Class;
    Item: in Exception_Occurrence);

for Exception_Occurrence'Read use
                    Read_Exception_Occurrence;

for Exception_Occurrence'Write use
                    Write_Exception_Occurrence;
```

These attributes enable the type Exception_Occurrence to be streamed. Note that this is a limited type and so streaming is only possible if predefined. A survey of other existing and new predefined limited types showed that no others needed to be treated in this way.

No other retrospective AIs actually affect the specification of any units but typically add or correct a number of rules. Of these some are of special interest because they introduce minor incompatibilities or inconsistencies. They are

108 Inheritance of stream attributes for type extensions

(108 was actually in the 2001 Corrigendum)

133 Controlling bit ordering

195 Streams (this covers many issues regarding streams)

220 Subprograms withing private compilation units

225 Aliased current instance for limited types

229 Accessibility rules and generics

238 Lower bound of Ada.Strings.Bounded_Slice

240	Stream attributes for limited types in Annex E
242	Surprise behavior of Update
246	Conversions between arrays of a by-reference type
253	Pragmas Attach_Handler and Interrupt_Handler
268	Rounding of real static expressions
279	Tag read by T'Class'Input
283	Truncation of stream files by Close and Reset
306	Class-wide extension aggregate expressions
341	Primitive subprograms are frozen with a tagged type
360	Types that need finalization
377	Naming of generic child packages
378	The bounds of Ada.Exceptions.Exception_Name
403	Preelaboration checks and formal objects
435	Storage pools for access-to-subprogram types
446	Raise_Exception for Null_Id

These are briefly discussed in the following subsections.

2.1 Incompatibilities with original Ada 95

There are a small number of incompatibilities between the original Ada 95 and that resulting from various corrections.

1 – A limited type can become nonlimited. Applying the Access or Unchecked_Access attribute to the current instance of such a type is now illegal. (AI-225, 3.10(26.e))

This is fairly obscure. Remember that the current instance rule is about referring to a type within its own declaration such as

```
type Strange is limited
  record
    Me: access Strange := Strange'Unchecked_Access;
    ...
  end record;
```

This is fine. It only makes sense to permit the attribute if the type is limited. But a type can be limited by virtue of having a limited component. for example

```
type Limp is limited private;

type Strange is
  record
    Me: access Strange := Strange'Unchecked_Access;
    C: Limp;
  end record;
```

If the component is limited private and it turns out that the full type of the component is not limited after all then the enclosing type becomes nonlimited. In such a case the attribute is now not allowed. The cure is to make the enclosing type explicitly limited.

2 – Conversions between unrelated array types that are limited or (for view conversions) might be by-reference types are now illegal. This is because they might not have the same representation and they cannot be copied in order to change the representation. (AI-246, 4.6(71.j))

3 – The meaning of a record representation clause and the storage place attributes for the non-default bit order is now clarified. One consequence is that the equivalence of bit 1 in word 1 to bit 9 in word 0 for a machine with Storage_ Unit = 8 no longer applies for the non-default order. (AI-133, 13.5.1 (31.d) and 13.5.2(5.c))

4 – Various new freezing rules were added in order to fix a number of holes in the original rules for Ada 95. (AI-341, 13.14(20.p))

5 – The type Unbounded_String is defined to need finalization. If the partition has No_Nested_Finalization and moreover the implementation of Unbounded_String does not have a controlled part then it will not be allowed in local objects now although it was in original Ada 95. Clearly this is extremely unlikely. (AI-360, A.4.5(88.b)). The same applies to the type Generator in Numerics.Float_Random and Discrete_Random (AI-360, A.5.2(61.a)) and to File_Type in Sequential_IO (AI-360, A.8.1(17.b)), Direct_IO (AI-360, A.8.4(20.a)), Text_IO (AI-360, A.10.1(86.c)) and Stream_IO (AI-360, A.12.1(36.b)). See also D.7(22.a).

This problem is unlikely with types such as Unbounded_String which were introduced into Ada 95 at the same time as controlled types and thus are almost inevitably implemented in terms of controlled types. It is more likely with the file types that existed in Ada 83 since some implementations might not have changed them to use controlled types.

6 – It is now illegal to apply the Access attribute to a subprogram declared in the specification of a generic unit in the body of that unit. The usual workaround applies which is to move the use of the attribute to the private part. (AI-229, 3.10.2(41.f))

7 – It is now illegal for the ancestor expression in an extended aggregate to be of a class wide type or to be dispatching call (probably most readers would never dream of doing that anyway). Thus if we have tagged type T and a type NT extended from it and we declare

```
X: T'Class := ... ;
```

then the aggregate

```
NT'(X with ... )          -- illegal
```

is illegal. We have to use a type conversion and write

```
NT'(T(X) with ... )       -- legal
```

Similarly the ancestor part cannot be a dispatching call such as F(X) where the function F is

```
function F(Y: T) return T is
begin
  return Y;
end F;
...
NT'(F(X) with ... )       -- illegal since X class wide
```

Again it can be fixed by a suitable conversion to a specific type. (AI-306, 4.3.2((13.b))

8 – If a generic library unit and an instance of it both have child units with the same name then they now hide each other. Thus

```
generic package G is ... ;        -- a generic G

generic package G.C is ... ;      -- a child C

with G;
package I is new G;               -- the instance

package I.C is ... ;              -- child of instance

with G.C;  with I.C;              -- illegal, both hidden
package P ...
```

Originally it seems that this was allowed but it was not specified which package C would refer to. This was fairly foolish and confusing. (AI-377, 8.3(29.z))

9 – A subprogram body acting as a declaration (that is without a distinct specification) cannot with a private child. This was allowed by mistake originally and permitted the export of types declared in private child packages. (AI-220, 10.1.2(31.f))

10 – For the purposes of deciding whether a unit can be preelaborable a generic formal object is nonstatic. (AI-403, 10.2.1(28.f))

11 – Storage pools (and the attribute Storage_Size) are not permitted for access to subprogram types. Originally it looked as if they were allowed provided they were never used (or the size was zero). (AI-435, 13.11(43.d))

12 – The rules for the two pragmas Interrupt _Handler and Attach_Handler are the same with respect to where they are permitted. Originally it appeared that Interrupt_Handler could be declared in a place remote from the subprogram it was referring to. (AI-253, C.3.1(25.a))

13 – There are some changes regarding attributes in remote type and RCI units. These changes primarily concern streams for limited types. (AI-240, E.2.2(18.a), E.2.3(20.b))

2.2 Inconsistencies with original Ada 95

There are a small number of inconsistencies between the original Ada 95 and that resulting from various corrections.

1 – The function Exception_Identity applied to the value Null_Occurrence now returns Null_Id whereas it originally raised Constraint_Error in Ada 95. See paper 5, section 2. (AI-241, 11.4.1(19.y))

2 – The procedure Raise_Exception applied to the value Null_Id now raises Constraint_Error whereas it originally did nothing (and thus returned). See paper 5, section 4. (AI-446, 11.4.1(19.aa))

3 – Rounding of static real expressions is now implementation-defined whereas it was originally defined as away from zero. The reason for the change is to match the behaviour of the hardware; this also means that static and non-static expressions are more likely to get the same answer which is comforting. (AI-268, 4.9(44.s))

4 – The lower bounds of strings returned by functions Exception_Name, Exception_Message, and Exception_ Information (and wide versions) are now defined to be 1. (AI-378, 417, 11.4.1(19.z))

Similarly the bounds of the various functions Slice are now defined. (AI-238, A.4.4(106.e))

5 – There are some changes regarding stream attributes. (AI-108, 13.13.2(60.g) and AI-195, 13.13.2(60.h))

6 – There are changes regarding truncation of stream files. (AI-283, A.12.1(36.a))

7 – There is a potential inconsistency regarding the use of Internal_Tag outside of streaming. However, there was an implementation permission to do as is now required and so programs were not portable anyway. (AI-279, 3.9(33.b))

8 – The procedure Update in Interfaces.C.Strings no longer adds a nul character. (AI-242, B.3.1(60.a))

3 Unfinished topics

A number of topics which seemed to be good ideas initially were abandoned for various reasons. Usually the reason was simply that a good solution could not be produced in the time available and the trouble with a bad solution is that it is hard to put it right later. In other cases it is now felt that the topic deserved further consideration in the light of better understanding; sometimes there was fairly general agreement that the current situation was not ideal and ought to be improved, nevertheless there was no agreement on what should be done. And in some cases the good idea seemed a bad idea after further discussion.

So it might be that when Ada is next revised these further features might be reconsidered and so perhaps this section might be called forthcoming attractions. But on the other hand maybe other matters will need to be dealt with in the light of user experience with Ada 2005.

The following subsections briefly outline the main topics – for a fuller discussion, consult the text of the Ada Issue concerned.

3.1 Aggregates for private types (AI- 389)

The <> notation was introduced for aggregates to mean the default value if any. See paper 3 section 4. A curiosity is that we can write

```
type Secret is private;

type Visible is
  record
    A: Integer;
    S: Secret;
  end record;

X: Visible := (A => 77; S => <>);
```

but we cannot write

```
S: Secret := <>;                  -- illegal
```

The argument is that this would be of little use since the components take their default values anyway.

For uniformity AI-389 proposed allowing

```
S: Secret := (others => <>);
```

for private types and also for task and protected types. One advantage would be that we could then write

```
S: constant Secret := (others => <>);
```

whereas at the moment it is not possible to declare a constant of a private type because we are unable to give an initial value.

However, discussion of this issue lead into a quagmire concerning the related AI-413 and in the end both were abandoned.

3.2 Partial generic instantiation (AI-359)

Certain attempts to use signature packages lead to circularities. The AI outlines the following example

```
generic
  type Element is private;
  type Set is private;
  with function Union(L, R: Set) return Set  is <>;
  with function Intersection(L, R: Set) return Set is <>;
  ... -- and so on
package Set_Signature is end;
```

Remember that a signature is a generic package consisting only of a specification. When we instantiate it, the effect is to assert that the actual parameters are consistent and the instantiation provides a name to refer to them as a group.

If we now attempt to write

```
generic
  type Elem is private;
  with function Hash(E: Elem) return Integer;
package Hashed_Sets is
  type Set is private;
  function Union(L, R: Set) return Set;
  function Intersection(L, R: Set) return Set;
  ...
  package Signature is new Set_Signature(Elem, Set);
private
  type Set is
    record
      ...
    end record;
end Hashed_Sets;
```

then we are in trouble. The problem is that the instantiation of Set_Signature tries to freeze the type Set prematurely.

Other similar examples concern the use of access types with private types. The essence of the problem is that we want to instantiate a package with a private type before the full declaration of that type.

The solution proposed was to split an instantiation into two parts, a partial instantiation and a full (that is, normal) instantiation. The partial instantiation might take the form

```
package P is new G(Private_Type) with private;
```

and this can be done with the partial view of the type. The full instantiation can then be given after the full declaration of the type.

This fell by the wayside at the last minute largely because of fears that awkward situations might be introduced inadvertently.

3.3 Support for IEEE 559: 1989 (AI-315)

The proposal was to provide full support for all aspects of IEEE 559 arithmetic such as Nans (a Nan is Not A Number). This would have necessitated adding attributes such as S'Infinity, S'Is_Nan, S'Finite and so on plus a package Ada.Numerics.IEC_559.

The proposal was abandoned because it would have had a big impact on implementers and it was not clear that there was sufficient demand.

3.4 Defaults for generic parameters (AI-299)

Generic subprogram parameters and object parameters of mode in can have defaults. But other parameters such as packages and types cannot. This was considered irksome and untidy and efforts were made to define a suitable notation for all possible generic parameters.

However, it was abandoned partly because an appropriate syntax seemed hard to find and more importantly, it was not felt to be that important.

3.5 Pre/post-conditions for subprograms (AI-288)

This proposal was to add pragmas such as Pre_Assert and Post_Assert. Thus in the case of a subprogram Push on a type Stack we might write

```
procedure Push(S: in out Stack; X: in Item);
pragma Pre_Assert(Push, not Is_Full(S));
pragma Post_Assert(Push, not Is_Empty(S));
```

These pragmas would be controlled by the pragma Assertion_Policy which controls the pragma Assert (which was of course incorporated into Ada 2005). Optional message parameters were allowed as well.

The general idea was that when the procedure Push was called, the expression Is_Full(S) would be evaluated and if this were false then action would be taken as for an Assert pragma. Note that the key difference from assert is that the pragmas go on the subprogram specification whereas to use Assert it would have to be placed in the body.

There were other pragmas for dispatching subprograms and so this was not quite so simple as at first appeared.

The proposal was abandoned for a number of reasons. There were more important matters to deal with and we were running out of time. Moreover, it seemed just the sort of topic where user experience on a trial implementation would be helpful in deciding what was required. And there was some feeling that since this was all dynamic it was not helpful to the high integrity community where the emphasis was on static analysis and proof.

3.6 Type and package invariants (AI-375)

This defined further pragmas similar to those in the previous proposal (AI-288) but concerned with packages and types. Thus the pragma Package_Invariant identified a function returning a Boolean result. This function would be implicitly called after the call of each subprogram in the package and if the result were false the behaviour would be as for an Assert pragma that failed.

This proposal was abandoned for the same reasons as AI-288.

3.7 Exceptions as types (AI-264)

This AI originally arose out of a workshop organized by Ada-Europe. The proposal was quite complex and considered far too radical a change and probably expensive to implement. As a consequence it was slimmed down considerably. But having been slimmed down it seemed pointless and was then abandoned. The only part to survive was the idea of raise with message which became a separate AI and was incorporated into Ada 2005.

3.8 Sockets operations (AI-292)

This seemed a very good idea at the time but no detailed proposal was forthcoming and so it died.

3.9 In out parameters for functions (AI-323)

This is a really interesting topic. Ada functions are curious. On the one hand they look as if they are going to be well behaved since they only allow in parameters and thus it appears as if they cannot have side effects. But of course they can have any side effects they like by using global variables! And parameters can be access types and nothing prevents the accessed values from being changed. Indeed access parameters are a sort of sly way of getting in out parameters anyway.

The proposal was to allow functions to have parameters of all modes. The rationale for the proposal is well summarized in the problem part of the AI thus "Ada functions can have arbitrary side effects, but are not allowed to announce that in their specifications".

Clearly, Ada functions are indeed curious. But strangely, this AI was abandoned quite early in the revision process on the grounds that it was "too late". (Perhaps too late in this context meant 25 years too late.) In any event there was no agreement on a way forward since there are strong arguments both ways. But there was agreement that time would be better spent discussing and agreeing other matters.

One suggestion is that two kinds of functions should be supported. Absolutely pure side-effect free functions that merely deliver the value of some state. Functions in SPARK [1] are like this. And the other sort of function could be one that is just like a procedure and can do anything and have all modes of parameters but for convenience returns a result which can then be used in an expression.

It is interesting to note that Preliminary Ada [2] had value returning procedures as well as functions. The functions were pure but value returning procedures were much as current functions and could have side effects. But value returning procedures could not have out and in out parameters. The difference between the two was thus not enough and so pure functions were dropped and value returning procedures became functions.

This topic may deserve to be revisited at some time.

3.10 Application defined scheduling (AI-358)

The International Real-Time Ada Workshops have been a source of suggestions for improvements to Ada. The Workshop at Oporto suggested a number of further scheduling algorithms [3]. Most of these such as Round Robin and EDF have been included in Ada 2005. But that for application defined scheduling was not.

The reason is perhaps that it was felt desirable to see how those that had been included worked out before adding yet more burden for implementers.

4 Acknowledgements

This is the last of the papers in this series and so this seems a good moment to once more thank all those who have helped by reviewing various drafts and pointing out where I had gone astray. I am especially grateful to Randy Brukardt, Pascal Leroy and Tucker Taft for their diligence and patience.

I must also thank Ada-Europe and the Ada Resource Association and also the British Standards Institute for financial support for attending various meetings.

Writing this rationale has been a learning experience for me and I trust that readers will also have found the material useful in learning about Ada 2005. An integrated description of Ada 2005 as a whole including some further examples will be found in a forthcoming version of the textbook [4].

References

[1] J. G. P. Barnes (2003) *High Integrity Software – The SPARK Approach to Safety and Security*, Addison-Wesley.

[2] ACM (1979) *Preliminary Ada Reference Manual*, Sigplan Notices, Vol. 14, No. 6.

[3] ACM (2003) *Proceedings of the 12th International Real-Time Ada Workshop*, Ada Letters, Vol 32, No 4.

[4] J. G. P. Barnes (2006) *Programming in Ada 2005*, Addison-Wesley.

Generating and improving Ada components for reuse

Muthu Ramachandran

School of Computing, The Headingley Campus, Leeds Metropolitan University, LEEDS, UK; email; m.ramachandran@leedsmet.ac.uk

Abstract

Software component reuse is the key to significant gains in productivity. However, the major problem is the lack of identifying and developing potentially reusable components. This paper concentrates on our approach to the development of reusable software components. A prototype tool has been developed, known as the Reuse Assessor and Improver System (RAIS) which can interactively identify, analyse, assess, and modify abstractions, attributes and architectures that support reuse. Practical and objective reuse guidelines are used to represent reuse knowledge and to do domain analysis. It takes existing components, provides systematic reuse assessment which is based on reuse advice and analysis, and produces components that are improved for reuse. Our work on guidelines has been extended to a large scale industrial application.

Keywords: Software reuse, component reuse, Development for reuse, Development with reuse, Reuse improvement, Reuse assessment

1 Introduction

Software component reuse is the key to significant gains in productivity. However, the major problem against the widespread introduction of reuse is the lack of identifying and developing potentially reusable components. We have clearly seen the difficulties that are faced when trying to reuse a component or a tool that is not designed for reuse. Therefore the objectives of this research are to explore the general area of **Development For Reuse** (DFR) and to investigate the possibility of automatically identifying, assessing and improving reusable domain abstractions, attributes and architectures. An objective of this process is to produce components that are potentially reusable as opposed to the normal practice of **Development With Reuse** (DWR) which has an objective of producing a product [1].

To achieve the production of reusable components we need to address the fundamental issue of what makes a component more reusable. Earlier studies have addressed this issue but do not go far from providing reusable guidelines [2-6]. Therefore, we took a more practical approach to address this issue by automating reuse guidelines for identifying, assessing, analysing and improving domain abstractions and attributes (*Domain analysis for reuse*) as well as identifying language features that affect component reusability (*Language analysis for reuse*). For example, certain languages (such as Java, C++, Ada95) support reuse explicitly. Engineers often cannot think about reuse when working on a market-driven project. In our approach we aim to integrate guidelines on language features and on domain analysis.

The notion of domain analysis has emerged from the well-known work conducted by Neighbors [7] on his pioneering project on the Draco system. Domain analysis aims to identify and design reusable components for a family of products. It also defines domain roles, process, and domain models and architecture. Existing work on domain analysis provides interesting guidelines, methods, and techniques on how to do domain analysis [8]. However, they fail to address, in detail, the issue of design for reuse. We took the existing work as a starting point for formulating reuse guidelines.

In our work, we have taken a more practical approach to domain analysis for the development of reusable software components by automating reuse guidelines. We also have defined the process of DFR, identifying domain abstractions & classification (domain-oriented reuse), language-oriented reuse, reuse assessment, and reuse improvement. Recently we have extended our work on guidelines into the design of reusable architectures for a large scale industrial application [9].

Our approach includes not only identifying abstractions and attributes but also assessing and adding these to improve components' reusability. A prototype has been developed, known as the Reuse Assessor and Improver System (RAIS). The major objective of this system is to demonstrate how well-defined reuse guidelines can be used to automate the process of development of component reuse by providing support for language analysis and domain analysis. For example, this system takes an Ada component specification, assesses it through two analysis phases, estimates its reusability according to how well it satisfies a set of reuse guidelines and generates a component which is improved for reuse. Furthermore reuse improvement is done by performing various classes of structural and architectural transformations. Reuse assessment allows the identification of such structural abstractions early in the process.

In this context the system has demonstrated that it is possible to:

- identify reusable abstractions, attributes and architectures effectively based on domain classification and reuse guidelines.
- automate reuse guidelines which provide detailed advice on how to construct reusable components.
- assist software engineers in the process of reuse assessment and improvement.
- model reusable components based on templates (automated improvement)
- produce components that are potentially reusable.

In the following sections we discuss the process on development for reuse, reuse guidelines, the system that generates reusable components, an example, and an evaluation of the approach.

2 The Process of Development for Reuse

The main objective of this project is to provide a software system supporting the process of the development for reuse. In our work this process consists of various activities as shown in Figure 1:

- Identify business needs - assess your existing system and application from the business point of view. What is the effort of building a new product? How much do we need to develop from scratch? How many components are you able to reuse? Justify your planned investment on reuse. Identify the application domain and its business/market needs. Define its boundary so that we can avoid producing components beyond the scope of the domain.
- Identify & classify reusable abstractions, identify a list of components, frameworks, architecture, and utilities that share your business goals and can produce a high return-on-investment.
- Formulate and classify reuse guidelines - produce reuse guidelines and classify them into domain-oriented reuse (i.e. guidelines on how to do domain analysis, guidelines on which abstraction has potential for reuse), design guidelines (guidelines on how design details/rationale can support reuse), architectural design guidelines, and language-oriented reuse (guidelines on language features).
- Design components, make sure reuse engineers are familiar with reuse guidelines.
- Assessment for reuse, allow other engineers' to conduct a reuse walkthrough or we can call it reuse inspection. Produce a detailed report following the inspection. It is interesting to see that reuse inspection is more structured and systematic since we have already formulated reuse rules.
- Improvement for reuse, modify components based on the assessment report
- Deliver potentially reusable components.

In this paper we concentrate mainly on two major activities, reuse assessment which is a process of assessing the reusability of a components against a set of well-defined guidelines, and reuse improvement which is a process of automatically modifying components structures and adding attributes that improve reusability.

We then identify reusable abstractions and classify them. The next step is to formulate practical reuse guidelines that can characterise reusable components effectively and precisely. The mechanism is based on taking the existing components, assessing these according to a set of guidelines, and then making suggestions on how the reusability of these components could be improved.

3 Reuse Guidelines as Knowledge Representation Technique

Probably there is no best and easy method of domain representation. Research is underway on how to do domain analysis, and on domain representation [8]. In our work, the approach we take is rule-based representation. Reuse guidelines are represented as rules. An example rule is:

```
IF abstract structure is complex AND
   all operations are independent of
   the type of the structure element THEN
   Component should be implemented as a
   generic package with the element type as
   a generic parameter;
END IF;
```

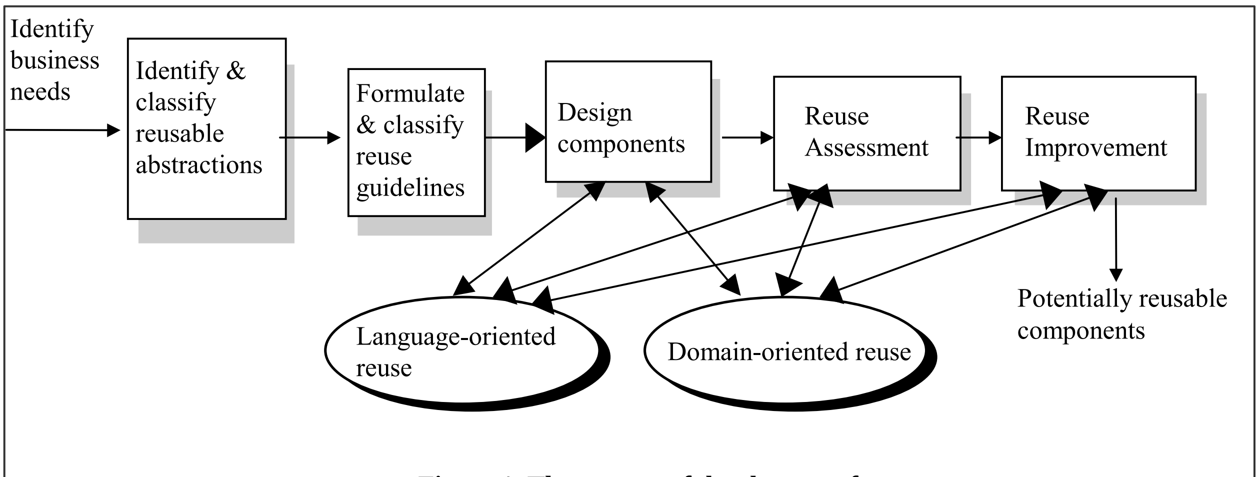


Figure 1 The process of development for reuse

However, automating some of these guidelines breaches this rule. For example, one of our guidelines on defining the list of operations on object creation, termination, object inquiry, and state change, involves more than one interaction and transformations. Hence it breaches our single if-then rule and depends on applying domain knowledge for further transformations. This information is modelled using a component template and the reusability is assessed and improved by comparing the component with that template.

Some of our guidelines are illustrated here:

1. *Design of abstract data types*. The notion of an abstract data type allows you to express real world entities of an application domain. It allows you to separate a specification from an internal representation of a structure (principle of information hiding). It means that we are able to specify an abstraction of a component in terms of its actual interface descriptions together which is useful to generalise that abstraction for reuse. It allows the designer to view a system at a more abstract level and to change the representation of ADS without affecting their use in other parts of the system.

One of our guidelines on ADS reads:

- For all complex structures, provide two representations such as static and dynamic structures for each domain abstraction.

This guideline says, for each structure, provide two abstractions such as static which is represented using an array structure and dynamic which is represented using dynamic structure (access/pointer). This provides a choice and maximum flexibility for the reuser with improved reuse potential. For example, in Ada, we can design two packages for each structure implemented statically and dynamically. If an abstraction is to be represented in Ada then we can apply various Ada reuse guidelines. For example, one on the rationale for choosing private types. That is, choose limited private for complex and dynamic structures, and choose private type for static structures. However, the Ada library mechanism is inadequate in that it rises naming conflict when there are two library units with similar names which means that the implementation of similar components must have different names.

Another important guideline [4] on the design of abstract data structures emphasises the need for providing methods for a list of operations such as object creation, object termination, state change, state inquiry, and input and output. They have not considered operations on exceptions that deal with error conditions. We believe that the operations on exceptions and handling are significant for reusable and reliable components. In our work we have extended this guideline to include operations on exceptions handling.

Our extended guideline on ADS reads:

- The components should be provided with the following operations on ADS.
    a. Creation
    b. Termination
    c. Conversion
    d. State inquiry
    e. State change
    f. Input/ output representation, and
    g. Exceptions

Creation involves both creating and initialising an object, termination is a means of making the object inaccessible for the remainder of its scope, conversion allows for the change of representation from one type to another, state inquiry functions allow the user to determine the state of the object and boundary conditions, state change functions allow modifying or changing the contents of the object, input/ output representations are primarily useful for debugging purposes, and exceptions deal with error conditions and exception handling procedures. Each operation emphasises one or more functionality so that the services offered by the component are increased thus leading to improved reusability. Sometimes components which do not provide all these operations may well be reused. In such cases, the component has to be measured based on the degree of reusability.

2. *Other guidelines*. Our guidelines on the design of reusable static and dynamic structures, and on space management are essential, objective and realisable. Complete set of guidelines can be found in [1 and 9]. Some of our important domain guidelines are:

- Always define a constrained array structure to represent a component of static structure.
- Always select dynamic object representation for all complex structures and hide detailed structural information.
- If the abstract structure is complex and all operations are independent of the type of the structure element then that component should be implemented as a generic package with the element type as a generic parameter.
- Always provide a procedure to record the maximum size of the free list with a counter so that the user may increase or decrease the size of the free list. when decreasing the free list size, space in excess of the new size is returned to the system.
- Always provide a procedure to release the free list, so that all space in the free list is returned to the system completely.
- For each exception, provide an exception handler.

In the following section we will see how these guidelines can be implemented as a tool for automated improvement and advisory system which can take Ada code and provides an assessment and improvement for reuse.

4 The Reuse Assessor and Improver System (RAIS)

Reuse assessment is concerned with assessing the reuse potential of a component against reuse guidelines. Reuse improvement has the goal of transforming an assessed component into a component that is improved for reuse, based on language-oriented and domain-oriented reuse guidelines. This system takes an Ada component specification and estimates its reusability according to how well it satisfies a set of reuse guidelines and generates a component which is improved for reuse. The system produces assessment reports based on the percent of guidelines satisfied and interacts with the user for making further improvements.

A general model of the tool for systematic reuse assessment and improvement has been developed as shown in Figure 2.

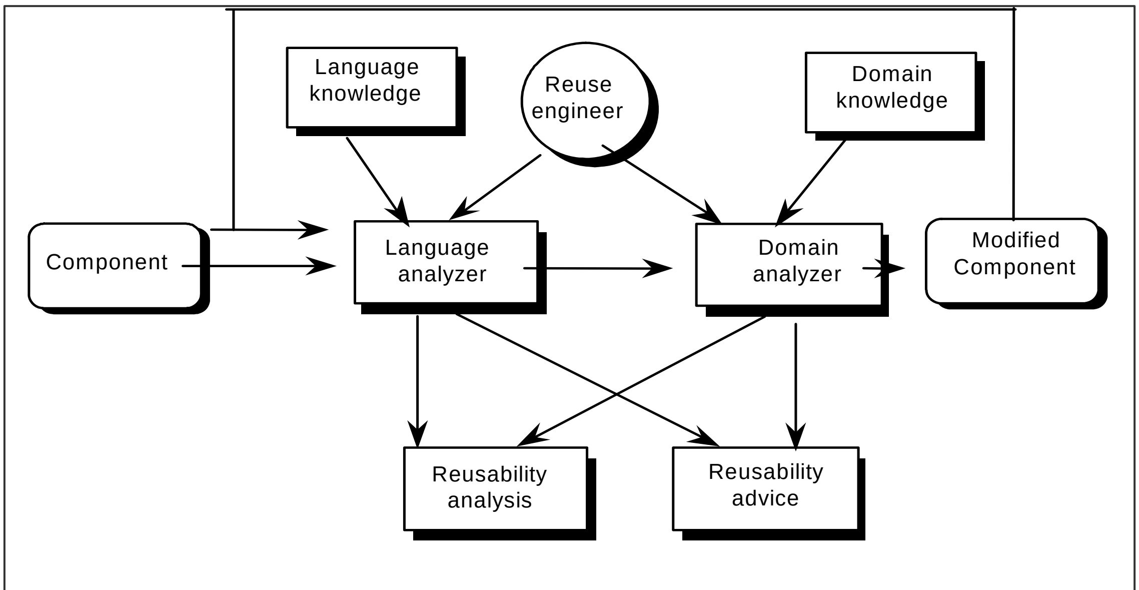


Figure 2 : Reuse Assessor & Improver System

The important features of this system are:

- Identifying domain abstractions, attributes and architectures, and language attributes and structures that affect component reusability.
- The integration of language knowledge (supporting language-oriented reusability) and domain knowledge (supporting domain-oriented reusability).
- Providing reusability advice and analysis,
- Assisting the reuse engineer in the process of assessing and improving his component for reuse.

RAIS considers a component specification rather than an implementation. However, this system can also generate implementation templates. We believe that reuse of specifications has definite advantages over reuse of implementations.

The RAIS system consists of a language analyser which is supported by built-in language knowledge and provides reusability analysis and advice, and a domain analyser which is supported by built-in domain knowledge and provides reusability analysis and advice.

An Ada component is firstly submitted to the language analyser which parses the component and applies the language-oriented guidelines to the code. Some of these guidelines require human input from the reuse engineer. RAIS predicts and records existing language constructs, and provides reuse advice and analysis. For example, the system can determine if the component processes arrays and if language attributes are used. However, it cannot automatically determine whether a component parameter refers to an array dimension and thus breaches the reuse guideline.

The language analyser assesses for reuse and changes the code after consulting the reuse engineer. The system interacts with the engineer to discover information that can't be determined automatically. The conclusion of this first pass is an estimate of how many guidelines are applicable to the component and how many of these have been breached. The report generator produces a report with all the information that has been extracted about that component and changes that have been made for reuse.

The second pass involves applying domain knowledge to the system. The component templates have been modelled representing static and dynamic structures. Their reusability is assessed by comparing the component against that template. Domain reuse improvement is done by adding methods automatically. Operation classes are identified by interaction with the reuse engineer. If some operations are found to be missing, skeleton implementations of these can be generated from the template for expansion to create a reusable component.

The support provided by the system ensures that the reuse engineer carries out a systematic analysis of the component according to the suggested guidelines. He or she need not be a domain expert. Again, an analysis is produced which allows the engineer to assess how much work is required to improve system reusability.

There are formulated reuse guidelines that emphasise the need for a packaging mechanism just like in Ada. Conceptually, packaging is a powerful mechanism for reuse. Some of these guidelines may only be possible with the Ada packaging mechanism such as private typing, the concept of specification which is independent of its body, and most importantly the concept of generics in order to achieve parameterisation. However, the approach and the methodology that are adopted by this system can easily be applied to any component. In this domain, RAIS uses the classification scheme in which each abstract data structure is classified into linear and non-linear structures and again these are classified into static, and dynamic structures.

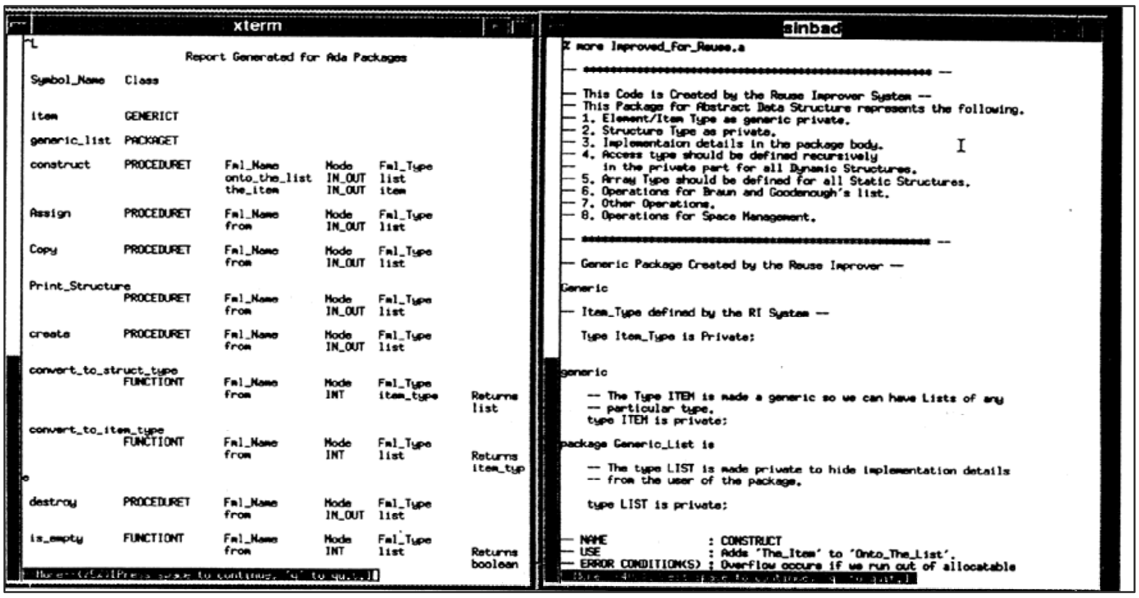
As well as this analysis, the system can also produce some reusability advice, generated from the guidelines, which is intended to assist the engineer in improving the reusability of the component. The knowledge of language and domain experts can be made available to the reuse engineer.

An ultimate objective is automatic reusability improvement where the system takes its own advice and some human guidance and modifies the component. A report and compilable code are produced. Clearly it is possible to use

- For all complex structures, the components should be implemented as a generic package with the element type as a generic parameter.

For instance, if a component of complex structure doesn't possess a generic package then the significance of this guideline becomes very important and therefore the system immediately reports to the reuse engineer that the component is weakly *reusable*. The system can make such structural modification automatically if the engineer decides to do so by responding to the dialogue.

In this way reuse assessment is being done by RAIS. The result of the assessment process is obviously arbitrary but it allows implementations to be compared, reuse improvements to be assessed, and it allows the reuse engineer to re-plan well before reusing components. The report generator produces the complete details of a component submitted to the systems in a tabular form which mainly consists of object name, its class, details of all the subprograms including the details of formal parameters and their class, and details of private types, etc. An example of a report is shown in a later section of this paper, see Figure 3.



```
^L
                    Report Generated for Ada Packages

Symbol_Name    Class

item           GENERICT

generic_list   PACKAGET

construct      PROCEDURET      Fml_Name         Mode     Fml_Type
                               onto_the_list    IN_OUT   list
                               the_item         IN_OUT   item

Assign         PROCEDURET      Fml_Name         Mode     Fml_Type
                               from             IN_OUT   list

Copy           PROCEDURET      Fml_Name         Mode     Fml_Type
                               from             IN_OUT   list

Print_Structure
               PROCEDURET      Fml_Name         Mode     Fml_Type
                               from             IN_OUT   list

create         PROCEDURET      Fml_Name         Mode     Fml_Type
                               from             IN_OUT   list

convert_to_struct_type
               FUNCTIONT       Fml_Name         Mode     Fml_Type
                               from             INT      item_type     Returns
                                                                       list

convert_to_item_type
               FUNCTIONT       Fml_Name         Mode     Fml_Type
                               from             INT      list          Returns
                                                                       item_typ
e

destroy        PROCEDURET      Fml_Name         Mode     Fml_Type
                               from             IN_OUT   list

is_empty       FUNCTIONT       Fml_Name         Mode     Fml_Type
                               from             INT      list          Returns
                                                                       boolean
--More--(75%)[Press space to continue, 'q' to quit.]
```

```
% more Improved_For_Reuse.a

-- ***************************************************** --

-- This Code is Created by the Reuse Improver System --
-- This Package for Abstract Data Structure represents the following.
-- 1. Element/Item Type as generic private.
-- 2. Structure Type as private.
-- 3. Implementaion details in the package body.
-- 4. Access type should be defined recursively
--    in the private part for all Dynamic Structures.
-- 5. Array Type should be defined for all Static Structures.
-- 6. Operations for Braun and Goodenough's list.
-- 7. Other Operations.
-- 8. Operations for Space Management.

-- ***************************************************** --

-- Generic Package Created by the Reuse Improver --

Generic

-- Item_Type defined by the RI System --

  Type Item_Type is Private;


generic

   -- The Type ITEM is made a generic so we can have Lists of any
   -- particular type.
   type ITEM is private;

package Generic_List is

   -- The type LIST is made private to hide implementation details
   -- from the user of the package.

   type LIST is private;


-- NAME               : CONSTRUCT
-- USE                : Adds 'The_Item' to 'Onto_The_List'.
-- ERROR CONDITION(S) : Overflow occurs if we run out of allocatable
[illegible]
```

Figure 3: Assessment report and improved

the language-oriented and domain-oriented guidelines to infer some code transformations which will improve reusability.

5 Reuse Assessment

Reuse assessment is a process of assessing the reuse potential of a component. It depends on the number of reuse guidelines that are satisfied by the component. RAIS predicts this and reports to the reuse engineer. RAIS measures the reusability strength of a component based on the percent of guidelines satisfied such as *weakly* (less than 50%), *strongly* (50-70%), *limitedly* (70-90%), *immediately reusable* (more than 90%) and also it takes into account the significance of a guideline (its importance for reuse).

For example, let us consider one of our domain guidelines:

6 Reuse Improvement

Reuse improvement is a stepwise process of improving a component for reuse through several transformations. Transformations can be simple, multiple, and cumulative. Because of the effort involved in this process, it has not been possible to implement for all the possible improvements. RAIS does most of the reuse improvements using reuse guidelines as domain rules and component templates. At present, RAIS can improve the component reusability by 50%.

Each abstract data structure is analysed and, by interaction with the user, the presence or absence of these operations is then identified. This information is modelled using a *component template* and the reusability is assessed by comparing the component against that template. Operation classes are identified by interaction with the reuse engineer. If some operations are found to be missing, skeleton implementations of these are generated from the template for expansion to create a reusable component.

Two types of templates are created supporting reuse of architectures, one for static structures and another for dynamic structures. After reuse assessment, the designer is given all the information captured from his component (a report generator for Ada has been designed for this purpose). Finally, RAIS generates the component that is assessed and improved for reuse after several transformations.

The system has taken a pragmatic approach to domain analysis supporting development for reuse. Figure 3 shows the details of a report generated by the system after an initial analysis and assessment. Finally, it generates the component that is improved for reuse.

7 Critical Evaluation

Existing approaches have not explored the issues of development for reuse and others have considered this as a management problem. In this context, our work has explored one of the major technical problems and the system has demonstrated that it is possible to assess and improve components reusability automatically. This work has also demonstrated that it is possible to formulate object and practical reuse guidelines that can assist and advise software engineers on how to construct components that are potentially reusable. This is one of the major practical steps taken in this work. Figure 4 illustrates how guidelines are classified and how many are automated.

RAIS has also demonstrated that the integration of language knowledge and the application domain knowledge is possible when modelling components for reuse. Therefore we feel that the various steps proposed for the process of development for reuse are important, practical and can be considered along with or before the normal software development process.

The system has also proved perhaps to a limited extent that it is possible to design for the highest form of reuse which is the reuse of components and architectures. The system models components effectively based on the templates for reuse of component architectures that are static and dynamic. It is not quite clear for example on what is probably the best technique for domain representation, what should be considered as a domain, and so on. In this context we might feel that the application domain chosen is perhaps inadequate in the commercial sense. However we believe that it is possible to extend the approach described here to other application domains, languages, and tools.

It has not been possible to automate all the guidelines that are formulated but it should be possible in a long-term project. The system does perhaps a limited number of domain-oriented reuse improvements. We believe that it is also possible to extend the approach described here to higher levels of reuse such as requirements definition and specification.

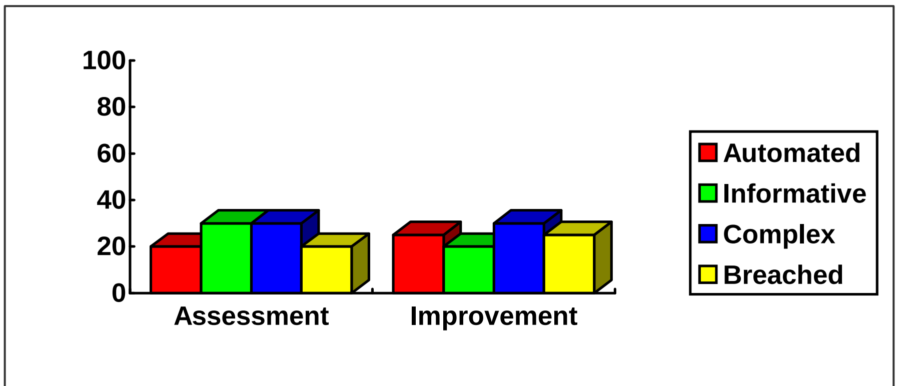


Figure 4 Automating reuse guidelines

Conclusions

The objectives of this project were to explore the general area of development for reuse and to investigate the possibility of automatically assessing the reusability of a software component and modifying that component to improve its reusability. In this context, the system has demonstrated that it is possible to identify, assess and improve components' reusability automatically based on domain knowledge and language knowledge.

In addition to these, more interesting results have evolved from this research, reusing generic component templates and generic architectures. Further work is needed to enhance the functionalities of RAIS. We believe that it is possible to extend the approach described here to other domains, languages and tools. Our work on reuse guidelines has been applied to a large-scale industrial application [9].

Acknowledgements

The author wishes to thank Prof Ian Sommerville for his support during this work at Lancaster University, Lancaster, UK.

References

[5] Sommerville, I. and Ramachandran, M. (1991), Reuse Assessment, First International Workshop on Software Reuse, Dortmund, Germany, July.

[6] Hooper, J. W. and Chester, R. O. (1991). Software Reuse: Guidelines and Methods, Plenum Press.

[7] Gautier, R.J. and Wallis, P.J.L. (Editors) (1990), Software Reuse with Ada, Peter Peregrinus Ltd for IEE/BCS.

[8] Braun, C.L. and Goodenough, J.B. (1985), Ada Reusability Guidelines, Report 3285-2-208/2, USAF.

[9] Booch, G. (1987), Software Components with Ada, Benjamin/Cummings.

[10] Dennis, R.J.St. (1987), Reusable Ada(R) software guidelines, Proc. of the 12th annual Hawaii International conference on system sciences, pp.513-520.

[11] Neighbors, J.M. (1984), The Draco Approach to constructing Software from reusable components, IEEE Trans. on Software Engineering, vol.SE-10, No.5, pp.564-574, September.

[12] Prieto-Diaz, R and Arango, G (ed) (1991), Domain Analysis and Software Systems Modeling, IEEE Computer Society Press Tutorial.

[13] Ramachandran, M. and Fleischer, W. (1996). Design for large scale reuse: an industrial case study, Proceedings of the 4th Intl. Conf. on Software Reuse, IEEE CS press, Orlando, Florida, USA.

[14] Tracz, W. (1991), Reuse through parameterization, ACM SIGSOFT Software Eng. Notes.

Ada-Europe 2005 Sponsors

AdaCore *Contact: Zépur Blot*	8 Rue de Milan, F-75009 Paris, France Tel: +33-1-49-70-67-16 Email: sales@adacore.com	 Fax: +33-1-49-70-05-52 URL: www.adacore.com
Aonix *Contact: Jacques Brygier*	66/68, Avenue Pierre Brossolette, 92247 Malakoff, France Tel: +33-1-41-48-10-10 Email : info@aonix.fr	 Fax: +33-1-41-48-10-20 URL : www.aonix.com
Artisan Software Tools Ltd *Contact: Emma Allen*	Suite 701, Eagle Tower, Montpellier Drive, Cheltenham, GL50 1TA, UK Tel: +44-1242-229300 Email : info.uk@artisansw.com	 Fax: +44-1242-229301 URL : www.artisansw.com
Esterel Technologies *Contact: Ian Hodgson*	PO Box 7995, Crowthorne, RG45 9AA, UK Tel: +44-1344-780898 Email : sales@esterel-technologies.com	 Fax: +44 1344 780898 URL : www.esterel-technologies.com
Green Hills Software Ltd *Contact: Christopher Smith*	Dolphin House, St Peter Street, Winchester, Hampshire, SO23 8BW, UK Tel: +44-1962-829820 Email :	 Fax: +44-1962-890300 URL : www.ghs.com
I-Logix *Contact: Martin Stacey*	1 Cornbrash Park, Bumpers Way, Chippenham, Wiltshire, SN14 6RA, UK Tel: +44-1249-467-600 Email : info_euro@ilogix.com	 Fax: +44-1249-467-610 URL : www.ilogix.com
LDRA Ltd *Contact: Brenda Pedryc*	24 Newtown Road, Newbury, Berkshire, RG14 7BN, UK Tel: +44-1635-528-828 Email: info@ldra.com	 Fax: +44-1635-528-657 URL: www.ldra.com
Praxis High Integrity Systems Ltd *Contact: Rod Chapman*	20 Manvers Street, Bath, BA1 1PX, UK Tel: +44-1225-466-991 Email : sparkinfo@praxis-his.com	 Fax: +44-1225-469-006 URL : www.sparkada.com
Silver Software *Contact: Steve Billet*	Riverside Buisness Park, Malmsebury, SN16 9RS, UK Tel: +44-1666-580-000 Email: enquiries@silver-software.com	 Fax: +44-1666-580-001 URL: www.silver-software.com
TNI Europe Limited *Contact: Pam Flood*	Triad House, Mountbatten Court, Worrall Street, Congleton, CW12 1DT, UK Tel: +44-1260-29-14-49 Email: info@tni-europe.com	 Fax: +44-1260-29-14-49 URL: www.tni-europe.com