ADA USER JOURNAL

Volume 26
Number 3
September 2005

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

News and Product Announcements

Ada User Journal is one of the ways in which people find out what is going on in the Ada community. Since not all of our readers have access to resources such as the World Wide Web and Usenet, or have enough time to search through the information that can be found in those resources, we reprint or report on items that may be of interest to them.

Reprinted Articles

While original material is our first priority, we are willing to reprint (with the permission of the copyright holder) material previously submitted elsewhere if it is appropriate to give it a wider audience. This includes papers published in North America that are not easily available in Europe.
We have a reciprocal approach in granting permission for other publications to reprint papers originally published in *Ada User Journal.*

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

All material for publication should be sent to the Editor, preferably in electronic format. The Editor will only accept typed manuscripts by prior arrangement.
Prospective authors are encouraged to contact the Editor by email to determine the best format for submission. Contact details can be found near the front of each edition.
Example papers conforming to formatting requirements as well as some word processor templates are available from the editor. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

I take pleasure in presenting this AUJ issue to our readership. Once again we had to go to 80 pages to accommodate all of the valuable material that was in schedule. A sizeable proportion of this issue is devoted to the new instalment of John Barnes' Rationale for Ada 2005. (Our readers will recall that the June 2005 meeting of the WG9 deliberated that "Ada 2005" should be the vernacular name of the new Ada language revision, so the title of John's volume is quite right!) As in the past, this new instalment is made of two chapters, one addressing my favourite topic, that is "Tasking and Real-Time", the other covering exceptions, generics and (restriction) pragmas. In addition to continuing the publication of Rationale instalments, starting with this issue we also commence the publication of material from the *Industrial Presentation track* of the Ada-Europe 2005 conference. The industrial track was a primer in the history of our yearly conference and those who attended know how successful the whole conference was, including the industrial track itself. While the regular papers feed the formal proceedings, published by Springer in the prestigious Lecture Notes in Computer Science series, the industrial-track presentations are being turned into short articles for publication in the AUJ. In this issue we host two such publication: one by Alan Marriott and Urs Maurer, lead members of the revised Ada-in-Switzerland association, who report about a utility of their own conception named "Ada Bug Finder"; and another by Burkhard Stadlmann, from the Upper Austrian University of Applied Sciences, which presents the Ada development of a operational train control system. The remainder of this issue is taken by the usual wealth of News and Calendar sections, respectively prepared by Santiago Urueña, our young news editor, and Dirk Craeynest, a long-timer of the AUJ.

Enjoy the reading and long live Ada!

Tullio Vardanega
Padova
September 2005
Email: tullio.vardanega@math.unipd.it

News

Santiago Urueña

Technical University of Madrid. Email: suruena@datsi.fi.upm.es

Contents

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]

Nov 16 - SIGAda Awards

From: John McCormick <mccormick@cs.uni.edu>
Date: 31 Aug 2005 07:11:11 -0700
Subject: Invitation to Nominate Candidates for SIGAda Awards
Newsgroups: comp.lang.ada

On Wednesday, 16 November 2005, the 2005 SIGAda Awards will be presented in a special morning plenary session at the SIGAda 2005 conference in Atlanta, Georgia. (See http://www.acm.org/sigada/conf/sigada2005 if you have somehow missed announcements of this year's annual SIGAda international conference.)

We welcome your nominations of deserving recipients.

The ACM SIGAda Awards recognize individuals and organizations who have made outstanding contributions to the Ada community and to SIGAda. The two categories of awards are:

(1) Outstanding Ada Community Contribution Award -- For broad, lasting contributions to Ada technology & usage.

(2) ACM SIGAda Distinguished Service Award -- For exceptional contributions to SIGAda activities & products.

Please consider who should be nominated this year. You may nominate a person for either or both awards, and as many people as you think worthy. One or more awards will be made in both categories.

Please visit http://www.acm.org/sigada/exec/awards/awards.html#Recipients and peruse the names of past winners. This may help you think about the measure of accomplishment that is appropriate. You may be aware of people who have made substantial contributions that have not yet been acknowledged. Nominate them. Consider what you believe to be the best developments in the Ada community or SIGAda in the last year; the last 5 years; since Ada's inception. Who was responsible? Nominate them.

Please note that anyone who has received either of the two awards remains eligible for the other. Perhaps there is an outstanding SIGAda volunteer who has won our Distinguished Service Award and who has also made important contributions to the advance of Ada technology, or visa versa. Nominate him or her!

The nomination form is available on the SIGAda website at http://www.acm.org/sigada/exec/awards/awards.html. (You need to visit this website to see past award winners' names, and also a picture of the statuette which is the award among other things, so you don't nominate someone who has already won an award in a category.) Submit your nomination as an e-mail attachment to SIGAda-Award@acm.org. You may also submit nominations on-line at: http://www.acm.org/sigada/cgi-bin/ICRS-Register.cgi?Awards

The ACM SIGAda Awards Committee, comprised of volunteers who have previously won an award, will determine this year's recipients from your nominations.

Call our attention to the people who are most deserving, by nominating them. And please nominate by OCTOBER 15!

Your participation in the nominations process will help maintain the prestige and honor of these awards.

John McCormick, Chair ACM SIGAda

[See also same topic in AUJ 25-1 (Mar 2004), p.5. --su]

Jun 5-9 - Ada-Europe 2006

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Date: 11 Aug 2005 23:20:47 +0200
Subject: 2nd CFP Conf. Reliable Software Technologies, Ada-Europe 2006
Organization: Ada-Europe, c/o Dept. of Computer Science, K.U.Leuven
Summary: Now is the time to prepare your submissions!
Newsgroups: comp.lang.ada,fr.comp.lang.ada

2nd CALL FOR PAPERS

11th International Conference on Reliable Software Technologies - Ada-Europe 2006, 5 - 9 June 2006, Porto, Portugal.

http://www.ada-europe.org/conference2006.html

Organised, on behalf of Ada-Europe, by Instituto Superior de Engenharia do Porto, in cooperation with ACM SIGAda (approval pending).

Ada-Europe organizes annual international conferences since the early 80's. This is the 11th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain ('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04), York, UK ('05).

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.

Schedule

30 October 2005: Submission of papers, workshop/tutorial proposals
20 January 2006: Notification to authors
20 February 2006: Camera-ready papers required
5-9 June 2006: Conference

Topics

[…] 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

Submissions

Authors are invited to submit original contributions. Paper submissions shall be in English, should be complete and should not exceed 20 double-spaced pages in length. Authors should submit their work via the Web submission system accessible from the conference Home page. The preferred format for submission is PDF. Postscript can also be accepted, as long as it was generated selecting the "optimize for portability" option in the used printer driver. Submissions by other means and formats will *not* be accepted. If you do not have easy access to the Internet, or you do not have an appropriate Web browser, please contact the Program Co-Chair Luís Miguel Pinho, whose address details are on this call as well as on the conference Home page.

Proceedings

The authors of accepted papers shall prepare their camera-ready submissions in full conformance with the LNCS style, not exceeding 12 pages and strictly by *February 20, 2006*. Authors should refer to: http://www.springer.de/comp/lncs/authors.html for format and style guidelines. Failure to comply will prevent the paper from appearing in the conference proceedings. The conference proceedings including all accepted papers will be published in the Lecture Notes in Computer Science (LNCS) series by Springer Verlag, which will be available at the start of the conference.

Awards

Ada-Europe will offer honorary awards for the best paper and the best presentation, which will be presented during the banquet and at the close of the conference respectively.

Call for Tutorials

Tutorials should address subjects that fall within the thrust of the conference and may be proposed as either half- or a 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.

Call for Workshops

Workshops on themes within the conference scope may be arranged to discuss matters of immediate technical interest as well as to foster action on longer-term technical objectives. Proposals may be submitted for half- or full-day workshops, to be scheduled on either ends of the main conference. Workshop proposals should be submitted by e-mail to the Conference Chair Luís Miguel Pinho. The workshop organizer shall also commit to preparing proceedings for timely publication in the Ada User Journal.

Exhibition

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.

Reduced Fees for Students

A small number of grants are available for students who will (co-)author and present papers at the conference. A reduction of 25% will be made to the conference fee.

Contact the Conference Chair Luís Miguel Pinho for details.

[Also see the full Call for Papers in this AUJ issue. --su]

Ada and Education

Tutorial about the Ada 2005 standard container library

From: Matthew Heaney <matthewjheaney@earthlink.net>
Date: Tue, 28 Jun 2005 13:28:07 GMT
Subject: Ada 2005 standard container library tutorial available
Newsgroups: comp.lang.ada

The Ada 2005 standard container library tutorial that I presented at Ada-Europe 2005 is now available:

http://charles.tigris.org/ai302_tutorial.ppt
http://charles.tigris.org/ai302_tutorial.pdf

The links appear under the "Related resources" section in the Description part of the Charles library CVS repository home page: <http://charles.tigris.org/>

The standard container library is specified by AI-302, the drafts of which are maintained here: http://www.ada-auth.org/cgi-bin/ cvsweb.cgi/AIs/AI-20302.TXT

From: Matthew Heaney <mheaney@on2.com>
Date: Tue, 28 Jun 2005 14:14:02 -0400
Subject: Re: No deques?
Newsgroups: comp.lang.ada

> Have deques been dropped from the container library? I seem to remember seeing them mentioned in the past, and both Charles and the C++ STL have them.

There are no deques in the Ada 2005 standard container library. (They were in my original proposal, but were quickly dropped in order to keep the library to a manageable size.)

Public Ada 95 Courses

From: Ed <colbert@abssw.com>
Date: 11 Aug 2005 08:46:39 -0700
Subject: Public Ada 95 Courses 19-23 September in Carlsbad CA
Newsgroups: comp.lang.ada

Absolute Software will be holding a public Ada 95 course during the week of 19 September 2005 in Carlsbad, CA. You can find a full description and registration form on our web-site, www.abssw.com. Click the Public Courses button in the left margin. (We also offer courses on software architecture-based development, safety-critical development, object-oriented methods and other object-oriented languages.)

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

[See also same topic in AUJ 25-4 (Dec 2004), p.184. --su]

How difficult is Ada to learn?

From: Ludovic Brenta <ludovic.brenta@insalien.org>
Date: Thu, 30 Jun 2005 07:11:17 +0200
Subject: Re: How difficult is Ada to learn?
Newsgroups: comp.lang.ada

> How difficult is the Ada programming language to learn? I have Pascal/Object pascal experience, and want to move to a new language soon. C/C++ is too scary for my tastes. I was considering either Ada or Modula-2 as my next language. How does Ada compare in power to Modula-2 and the Borland's dialect of Pascal?

I would say:

Pascal < Modula-2 < Object-Pascal < Ada

But, if you know Object-Pascal, learning Ada won't be too difficult for you. The main areas where you'll learn new things are tasking and the Ada way of doing OOP.

I suggest you read the following article, which will help you:
http://homepage.sunrise.ch/mysunrise/gdm/pascada.htm

From: Gene <eugene.ressler@frontiernet.net>
Date: 30 Jun 2005 07:19:10 -0700
Subject: Re: How difficult is Ada to learn?
Newsgroups: comp.lang.ada

There are really two answers. If you intend to structure programs as in the other languages, then it's quite easy to shift sideways into Ada syntax. If you want to exploit Ada's full power, then you'll have to figure out the Ada way to do things and learn the parts of the language that are not found elsewhere---task types for example. In that regard, one of the best things about Ada is that the language designers put their thinking in the rationale documents and then Cohen took the whole art of explaining how to use a language to a new level. There is so much glop out there on the language-of-the-week that these two books really stand out.

Most people who start with Borland Pascal miss the built-in set and string data types. Ada gets the same effects with packages, but the syntax is far less elegant and readable. (Disclaimer: I have no real Modula-2 experience.)

From: svaa@ciberpiula.net
Date: 30 Jun 2005 12:36:20 -0700
Subject: Re: How difficult is Ada to learn?
Newsgroups: comp.lang.ada

If you have experience with Pascal, I don't think that will be very difficult to learn. There are some new concepts like generics, discriminants and tasks. You will find things different like objects and pointers, and you'll miss some things like an easy strings management, sets, and little more.

In the other hand, I like Ada syntax for blocks, that is, you don't see a cascade of wild "end", you see "end nameprocedure" or "end if" etc. I like the restrictions of types, harder than Pascal (and much harder than C). I like that the standard has a lot, a big lot of functions that are standard in any Ada compiler.

There is also a new way of doing things in Ada. You will need some experience to get the touch.

If you use Borland's products, you will miss a good IDE. Nowadays languages have libraries (packages in Ada) some standard, some added by vendor, and some added by yourself. There are thousands of functions, procedures and data structures, all that information is difficult to handle without good tools. Borland is a master in this matter, good integrated help, fast access to record fields and to methods of an object, etc. There is nothing like that in Ada, and that's a big problem, specially for a beginner.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Fri, 01 Jul 2005 05:39:53 GMT
Subject: Re: How difficult is Ada to learn?
Newsgroups: comp.lang.ada

> Not very much. I found it easy to learn from a pascal background, so you should have no problem.

The US Military Academy (West Point) did a controlled study that concluded that Ada was a better 1st language than Pascal. Presumably that means Ada's easier to learn than Pascal.

From: Dennis Lee Bieber <wlfraed@ix.netcom.com>
Date: Fri, 01 Jul 2005 06:54:57 GMT
Subject: Re: How difficult is Ada to learn?
Newsgroups: comp.lang.ada

Or might just mean that they didn't have to "unlearn" all the bad habits of Pascal, afterwards.

The confusing "prefetch" of Pascal I/O (at least in the UCSD version my college used). And how many blocks end with a nil statement because one ended a statement with a ;

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Thu, 30 Jun 2005 18:43:12 +0200
Subject: Re: How difficult is Ada to learn?
Newsgroups: comp.lang.ada

Well I have Pascal/Modula-2 experience and found Ada a perfect replacement for the two. Ada really is the Pascal for grown ups having all the features which where missing in Pascal. Even Turbo-Pascal comes close to Ada's Power. As for Modula-2: Apart from the nice support for modules the feature set was in fact reduced when compared to Pascal.

If you want to learn more about Ada look at:
http://en.wikibooks.org/wiki/Programming:Ada

There are 192 Web Pages of tutorial material out there.

Ada-related Resources

Ada Wikibook is "Book of the month"

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Fri, 02 Sep 2005 17:18:50 +0200
Subject: Wikibook Ada Programming won "Book of the month"
Newsgroups: comp.lang.ada

The Wikibook "Ada Programming" has won the "Book of the month for September 2005" contest.

"Ada Programming" had twice a many votes then the next contestant - including many votes from outside the programming comunity for which we are particular grateful.

"Ada Programming" will be featured on the main page of Wikibooks for the duration of the next month which hopefully will raise awareness of Ada.

Links

Wikibooks Main Page:
http://en.wikibooks.org/wiki/Main_Page

Ada Programming:
http://en.wikibooks.org/wiki/Ada_Programming

Book of the month winner for 2005:
http://en.wikibooks.org/wiki/Wikibooks:Book_of_the_month/2005

Votes and Results September 2005:
http://en.wikibooks.org/wiki/Wikibooks:Book_of_the_month/September_2005_voting

From: Stephane Riviere <stephane@rochebrune.org>
Date: Thu, 08 Sep 2005 09:24:21 +0200
Subject: Re: Wikibook Ada Programming won "Book of the month"
Newsgroups: comp.lang.ada

Very good news!

I congratulate you, Martin, and also all other contributors!

I wasn't really aware about this wiki book. It's a great work...

[See also "Ada at Wikipedia & Wikibooks" in AUJ 26-1 (Mar 2005), p.8. --su]

Online Ada Books

From: PeterK <Peter_Kitson@hotmail.com
Date: 12 Aug 2005 13:49:49 -0700
Subject: Collection of Online Ada Books
Newsgroups: comp.lang.ada

A collection of published Ada programming books that the authors have generously allowed to be available for free downloads: http://www.computer-books.us/ada95.php

Let me know of other Ada books to add.

From: Manuel G. R. <mgrojo@ya.com>
Date: Wed, 17 Aug 2005 22:55:13 +0200
Subject: Re: Collection of Online Ada Books
Newsgroups: comp.lang.ada

I recently got news of this one: http://stwww.weizmann.ac.il/g-cs/benari/books/index.html#ase

Ada for Software Engineers. John Wiley & Sons, 1998. ISBN 0-471-97912-0.

But take into account the copyright permissions that I paste here for reference:

"You may download, display and print one copy for your personal use in non-commercial academic research and teaching. Instructors in non-commercial academic institutions may make one copy for each student in his/her class. All other rights reserved. In particular, posting this document on web sites is prohibited without the express permission of the author."

Ada-related Tools

Simple components

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 24 Jul 2005 16:30:37 +0200
Organization: cbb software GmbH
Subject: Simple components v2.0
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, sets, maps, stacks, tables, string editing, unbounded arrays and expression analyzers: http://www.dmitry-kazakov.de/ada/components.htm

In the version 2.0 tools were added to ease creation of parsers matching a word from a list keywords. Get_Token procedure does this using a user-provided table. The package Keywords generates a keyword matching parser from an enumeration type, which literals are the keywords to match.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 14 Aug 2005 19:35:22 +0200
Subject: ANN: Simple Components v 2.1
Organization: cbb software GmbH
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, sets, maps, stacks, tables, string editing, unbounded arrays and expression analyzers. It is here: http://www.dmitry-kazakov.de/ada/components.htm

Changes made:

Behavior of arguments sublists separators was clarified. The call-back Enclose is now called for each sublist with the parameters indicating the brackets or separators that starts and ends the sublist. Parsers.Multiline_Source.Get_Line raises End_Error at the source end (instead of Constraint_Error).

[See also same topic in AUJ 26-1 (Mar 2005), pp.9-10. --su]

Strings Edit

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Organization: cbb software GmbH
Date: Sat, 25 Jun 2005 11:46:06 +0200
Subject: ANN: Strings Edit v1.7
Newsgroups: comp.lang.ada

This library provides I/O facilities for integer, floating-point, Roman numbers, and strings. Both input and output subroutines support string pointers for consequent stream processing. The output can be aligned in a fixed size field with padding. Numeric input can be checked against expected values range to be either saturated or to raise an exception. For floating-point output either relative or absolute output precision can be specified. UTF-8 encoded strings are supported.

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

Changes to the previous version:

- Ada-style quoted strings support was added;

- Bug fix in Trim (Trim raised Constraint_Error with an empty string).

XML EZ Out - XML-formatted output

From: Marc A. Criley <mc@mckae.com>
Date: Tue, 14 Jun 2005 08:29:27 -0500
Subject: ANN: XML EZ Out 1.03 posted
Newsgroups: comp.lang.ada

XML EZ_Out is a small set of packages intended to aid the creation of XML-formatted output from within Ada programs. It basically wraps the tags and data provided to it with XML syntax and writes them to a user-supplied medium.

XML EZ Out is available at http://www.mckae.com/xml_ezout.html.

Changes since 1.02:

Fixed bug dealing with "&" for "&" substitution.

From: Marc A. Criley <mc@mckae.com>
Date: Thu, 16 Jun 2005 10:26:05 -0500
Subject: XML EZ Out 1.04 Posted
Newsgroups: comp.lang.ada

XML EZ_Out is a small set of packages intended to aid the creation of XML-formatted output from within Ada programs. It basically wraps the tags and data provided to it with XML syntax and writes them to a user-supplied medium.

XML EZ Out is available at http://www.mckae.com/xml_ezout.html.

Changes since 1.03:

Added quote and apostrophe substitution (""" and '") within attribute values.

From: Marc A. Criley <mc@mckae.com>
Date: Wed, 15 Jun 2005 08:13:56 -0500
Subject: Re: ANN: XML EZ Out 1.03 posted
Newsgroups: comp.lang.ada

> One question. The code is GMGPL as the source suggests? Is it OK to distribute the library source code with a program that uses it? Do you want any special files to be included or it is ok as is?

Yes, it is GMGPL. You can distribute the source code with a program using it. Please keep it together in a single directory and keep the String_Stream and Text_File child packages in there, even if you're not using them, so that the users of your app would see XML EZ Out as a utility they could pick up and use as well.

In that folder you can omit the tmeztf.adb test program, as well as the README, but I would appreciate a link somewhere in your README or other documentation pointing back to www.mckae.com.

From: Marc A. Criley <mc@mckae.com>
Subject: Re: ANN: XML EZ Out 1.03 posted
Date: Wed, 15 Jun 2005 10:58:25 -0500
Newsgroups: comp.lang.ada

> I was thinking to put all in a mckae folder. Could you perhaps make the packages with a bottom folder. If one unpack the packages now all files goes into current directory. A bit more tidy if there is a hierarchy like:
> - mckae/README
> - mckae/mckae.ads
> - ....

Actually, that's exactly the hierarchy I have on my developmental machine, I've just neglected to distribute it in that form. So that's a good point, and I'll ensure that gets done for future releases.

PragmARC - PragmAda Reusable Components

From: PragmAda Software Engineering <pragmada@earthlink.net>
Date: Tue, 14 Jun 2005 13:59:59 GMT
Subject: Ann: PragmAda Reusable Components Release
Newsgroups: comp.lang.ada

PragmAda Software Engineering is proud to announce a new release of the PragmAda Reusable Components. This release corrects an error with PragmARC.Get_Line when used with standard input, modifies PragmARC.Matrix_Math to not assume that multiplication is commutative for the element type, and adds two new "safe" components.

You may find the PragmARCs at http://home.earthlink.net/~jrcarter010/pragmarc.htm

Error reports, comments, and suggestions are always welcome at pragmada@earthlink.net.

[See also same topic in AUJ 26-1 (Mar 2005), p.10. --su]

Tables for Ada

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 6 Jun 2005 21:23:29 +0200
Organization: cbb software GmbH
Subject: ANN: Tables for Ada v1.4
Newsgroups: comp.lang.ada

This library provides an implementation of tables indexed by strings. The binary search is used for names of known length. It is also possible to search a table for names of unknown length, i.e. to parse a string using some table. Table elements can be of any private type. Key-insensitive tables are supported.

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

Changes to the version 1.3:

- Function IsIn was added to provide membership test;
- Bug fixes in the test programs.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Organization: cbb software GmbH
Date: Sat, 20 Aug 2005 19:54:03 +0200
Subject: ANN: Tables for Ada v1.5
Newsgroups: comp.lang.ada

The library provides an implementation of tables indexed by strings. The binary search is used for names of known length. It is also possible to search a table for names of unknown length, i.e. to parse a string using some table. Table elements can be of any private type. Key-insensitive tables are supported.

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

Changes:

The package Tables.Names can now be instantiated with the parameter specifying the set of characters considered blank.

GNAT GPL 2005 Edition

From: Jamie Ayre <ayre@adacore.com>
Date: Thu, 15 Sep 2005 08:50:39 +0200
Subject: GNAT GPL 2005 Edition is now available
Newsgroups: comp.lang.ada

AdaCore announces the immediate availability of the GNAT GPL 2005 Edition.

AdaCore is pleased to announce the release of the GNAT GPL 2005 Edition to provide Free Software developers, that is developers that distribute their work under the GPL (GNU General Public License), the latest and most advanced Ada 2005 software development environment.

As many of you know, the Ada programming language is undergoing a revision, called Ada 2005. Many of the new features in the revision are available in the GNAT GPL 2005 Edition, notably:

New language level features:

- Abstract interface types to provide multiple inheritance
- Task, protected and synchronized interfaces
- Limited-with and Private-with clauses
- Prevention of accidental overloading when overriding
- Object.operation notation
- General use of anonymous access subtypes
- Limited aggregates
- Access to constant parameters and null-excluding access subtypes
- Unchecked_Unions for C interfacing
- Nested type extensions
- Support for 16-bit and 32-bit characters.

New standard libraries:

- Container library
- Complete definition of string subprograms (fixed, bounded, unbounded)
- Directory operations.

The GNAT GPL 2005 Edition, which is available free of charge from http://libre.adacore.com/, is licensed for Free Software development under the terms and conditions of the GNU General Public License (GPL). Implementation of the new Ada 2005 features is also available in GNAT Pro, which is licensed for all types of software development.

For more information visit the following links:

* Ada 2005:
http://www.adacore.com/ada_2005.php/

* GNAT GPL 2005 Edition:
http://libre.adacore.com/

* GNAT Pro:
http://www.adacore.com/gnatpro_summary.php/

[See also "ACT - New Public Release GNAT 3.15p" in AUJ 24-1 (Mar 2003), p.17. --su]

From: Jeffrey Carter <jrcarter@acm.org>
Date: Tue, 13 Sep 2005 10:03:43 -0700
Subject: Re: GNAT GPL 2005 Edition
Organization: Raytheon Company
Newsgroups: comp.lang.ada

Interesting. In the past, AdaCore's public releases were suitable for producing non-GPL SW. I wonder if "This new Edition will provide Free Software developers, that is developers that distribute their work under the GPL" means the runtime and libraries will be GPL, rather than GMGPL, forcing the resulting executables to be GPL.

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Date: Wed, 14 Sep 2005 00:00:39 GMT
Subject: Re: GNAT GPL 2005 Edition
Newsgroups: comp.lang.ada

David Trudgett wrote:

> Does the libstdc++ exception permit dynamic linking?
> Yes. The intent of the exception is to allow people to compile proprietary software using gcc.
> [...] that would be a matter of library licensing, and not of GNAT's licence itself.

Exactly. Until now, the license of Gnat's implementation of the standard Ada library has had an exception just like libstdc++. Now this announcement appears to say that the new edition will only be useful for building GPL software, and that makes Jeffrey Carter and me wonder if perhaps that exception has been removed.

From: David Trudgett <wpower@zeta.org.au.nospamplease>
Date: Wed, 14 Sep 2005 09:33:38 +1000
Subject: Re: GNAT GPL 2005 Edition
Newsgroups: comp.lang.ada

Even if it is the case (and I have no idea) that the modified GPL is not used, it still does not mean that resulting executables are "GPL". It would only mean (at most -- see below) that those executables couldn't be formally distributed except under the GPL. This means that in-house proprietary software is completely unaffected by the GPL.

"The GPL permits anyone to make a modified version and use it without ever distributing it to others." ...

"It is essential for people to have the freedom to make modifications and use them privately, without ever publishing those modifications."

http://www.gnu.org/licenses/gpl-faq.html

That said, I believe the spirit of the GPL (applied to compilers like GNAT) is (or should be) to protect the software itself (compiler) from being hijacked by proprietary interests. The GPL doesn't apply to the output of programs, and an executable binary is just the output of the compiler. Therefore, according to this logic, programs compiled by a GPL'ed compiler are not themselves under the GPL (unless you make them so).

In that regard, note:

Can I use GPL-covered editors such as GNU Emacs to develop non-free programs? Can I use GPL-covered tools such as GCC to compile them?

Yes, because the copyright on the editors and tools does not cover the code you write. Using them does not place any restrictions, legally, on the license you use for your code.

Some programs copy parts of themselves into the output for technical reasons--for example, Bison copies a standard parser program into its output file. In such cases, the copied text in the output is covered by the same license that covers it in the source code. Meanwhile, the part of the output which is derived from the program's input inherits the copyright status of the input.

As it happens, Bison can also be used to develop non-free programs. This is because we decided to explicitly permit the use of the Bison standard parser program in Bison output files without restriction. We made the decision because there were other tools comparable to Bison which already permitted use for non-free programs.

So, the only possible grey areas with GNAT would be (a) if it copies parts of itself into its output; or (b) statically or dynamically links to GPL'ed libraries. In regard to libraries, we note in the case of GCC:

Does the libstdc++ exception permit dynamic linking?

Yes. The intent of the exception is to allow people to compile proprietary software using gcc.

So, can anyone comment on whether (a) or (b) actually applies to GNAT? I have my doubts that (a) would apply, since it doesn't seem to apply to GCC. As for (b), that would be a matter of library licensing, and not of GNAT's licence itself.

VC_View and PCHIF - SPARK Proof Tools

From: JP Thornley <jpt@diphi.demon.co.uk>
Date: Mon, 6 Jun 2005 17:51:52 +0100
Subject: A couple of tools for SPARK Proof
Newsgroups: comp.lang.ada

I've developed a couple of tools to help when doing Spark proof - called VC_View and PCHIF.

The purpose of VC_View is to make it easier to read and interpret Spark VCs. It does this in two ways:

1. Only immediately relevant hypotheses are initially shown
2. User identifiers are replaced by upper case letters.

The hypotheses that are shown initially are those that share an identifier with a conclusion. These can then be selectively extended.

Other features allow the hypotheses to be restricted to just those for a single conclusion, and give access to the rules file for the VCs.

The main purposes of PCHIF are to make it easier to recall and edit previously entered commands and to give better control over the commands that are stored.

As well as making it easier to create proof scripts, PCHIF should also simplify the creation of 'clean' scripts - i.e. ones without failed inferences, etc. (A long term ambition for PCHIF is to simplify the maintenance of proof scripts by providing a 'stop on fail' feature - to make it easier to pinpoint commands that need changing.)

At present PCHIF is an experimental prototype and has a number of limitations - the main one is that it only handles input to the Checker. (In order to handle the output from the Checker it must be piped to another program - when this is done the Checker prompts don't appear down the pipe until after the Checker has received the input that it is prompting for.)

Both tools are 100% Ada, and compiled with GNAT 3.15p. Only MS Windows executables are provided (XP Pro, but may work on other versions).

However they use GtkAda so it should be possible to compile them for other platforms. (The sources aren't currently provided for download but I intend to make them available once they've been made fit for public view.)

The download page can be reached from www.sparksure.com.

If you try the tools, please use the contact email address given on the sparksure pages to send me comments.

Profiling GNAT programs with gprof

From: Alex R. Mosteo <alejandro@mosteo.com>
Date: Tue, 07 Jun 2005 19:00:12 +0200
Subject: Re: Profiling GNAT programs with gprof
Newsgroups: comp.lang.ada

Alfred Hilscher wrote:

> Has someone experience with gprof and GNAT? It seems to work when I profile sequential programs, but if I have tasks in my code then the result of gprof seem invalid. E.g. a procedure called only three times at all is listed as about 10000 calls. And while the over all runtime of the prog is about 2 sec, gprof shows the consumed time for this procedure with a few thousands seconds. It looks like as if the statistic counters were not initialized. What have I to do to get correct results? I use GNAT 3.15p and Windows 2000.

I did this in the past in the same platform as you, also with a multitasking program. I don't remember seeing unexpected counter values. Have you a simple test program to highlight this problem? I could try to run it and compare results.

If I'm not mistaken, the timing problem can arise from the way gprof computes time: each call is given a fixed differential duration. So that can artificially inflate the times reported. But you should check this in the documentation, I may be confused about this behavior.

From: Jeff Creem <jcreem@yahoo.com>
Date: Wed, 08 Jun 2005 07:31:26 -0400
Subject: Re: Profiling GNAT programs with gprof
Newsgroups: comp.lang.ada

I have had good luck with gprof on non-tasking programs though in most cases I ended up either having to increase the systems default interrupt clock rate to get good results or wrapping the main code I cared about in a loop to get good results (this was needed even for non-trivial programs that were not all that fast to complete).

Also to get reasonable results I often have to revert to using the -F option (better when possible) or a series of -f options (still ok but not great when -F is really needed)

As for tasking programs... There is a lot of info on the web about gprof failing with threads/forks/etc and a few platform specific workarounds appear to exist.

The failure mechanisms also seem somewhat platform specific as well.

http://sam.zoy.org/writings/programming/gprof.html

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Wed, 08 Jun 2005 09:22:59 +0200
Subject: Re: Profiling GNAT programs with gprof
Organization: Adalog
Newsgroups: comp.lang.ada

Marius Amado Alves wrote:

> I also have experienced erroneous Gprof results. Functions *never* called listed as called. I gave up on Gprof. Sorry not being of help. Just telling my experience. Next time I need profiling I think I'll try another tool e.g. RootCause, or instrument my code. Good luck to you with Gprof.

I had obviously wrong results with gprof both with tasking and non-tasking programs, and I gave up using it.

Note that if you just want to time some procedure calls (not full profiling), you can use the facilities from Debug.Timing package, available as part of Adalog's Debug package. See http://www.adalog.fr/compo2.htm

Ada Plug-in for Eclipse

From: Jeff Creem <jcreem@yahoo.com>
Date: Thu, 28 Jul 2005 07:27:08 -0400
Subject: Re: Ada Plug-in for Eclipse

Newsgroups: comp.lang.ada

> Are there any open source Ada plugins for Eclipse? I've been developing one and I wonder if it is the only available.

There has been some discussion on the eclipse CDT plug-insite that they would like to support Ada and there appears to be some people working on it (hard to tell how much).

It might be nice if you drop in there and see if you can help.

I think many of us would be interested.

From: Adrien Plisson <aplisson-news@stochastique.net>
Date: Wed, 27 Jul 2005 16:58:56 +0200
Subject: Re: Ada Plug-in for Eclipse
Newsgroups: comp.lang.ada

An Ada Plug-in for Eclipse is currently developed by Aonix. Unfortunately, it does not seem to be freely available.

[See also "Aonix - Eclipse-based Ada IDE for Mission- and Safety-Critical Development" in AUJ 25-4 (Dec. 2004), p.194. --su]

From: "Dan McLeran" <danmcleran@yahoo.com>
Date: 27 Jul 2005 07:30:22 -0700
Subject: Re: Ada Plug-in for Eclipse
Newsgroups: comp.lang.ada

I was working on this as well. I'd like to see how far you've gotten. Maybe I have some code that can be used.

From: Phill <crazyphill@gmail.com>
Date: 27 Jul 2005 15:14:03 -0700
Subject: Re: Ada Plug-in for Eclipse
Newsgroups: comp.lang.ada

Like I said earlier, I plan on having a builder (using GNAT) and syntax highlighting working. Right now, syntax highlighting is almost complete, I just have a few problems with numerical literals in certain situations--these are easy to fix, but I'm working on the builder now.

The builder will hopefully auto compile the sources as they're modified (Like the Java plug-in) and will build the executable on command.

I've also created a barebones perspective and will be creating a project wizard to help support the builder.

What have you done so far?

From: Dan McLeran <danmcleran@yahoo.com>
Date: 27 Jul 2005 15:22:16 -0700
Subject: Re: Ada Plug-in for Eclipse
Newsgroups: comp.lang.ada

I got pretty much as far as you've gotten. My plug-in was creating gnat project files to manage the build as the user set certain options. I also planned to create wizards for dlls, exes, etc.

Ada Application Support Component Library

From: Michael Erdmann <michael.erdmann@snafu.de>
Date: Tue, 28 Jun 2005 21:53:14 +0200
*Subject: Announcing *** Release of AdaPlugin Version 0.1.0 **
Newsgroups: comp.lang.ada

Adaplugin allows you to easily build plug-ins with Ada 95 on a Linux box for GNAT.

The source code and binaries for Linux are available for download at:

http://sourceforge.net/projects/ascl

I am sorry in the current version the documentation is quite insufficient but this will change in the next two weeks.

[The objective of the ASCL project is:

* Creation of an Ada support library which ranges from light weight packages (e.g. linked lists) to heavy weight package like configuration management packages. The central idea of this project is not to develop every thing from scratch but to integrate what is already existing.

* In the long term the library should become a de facto standard for the Ada community because of its intensive use in the community.

* In the long term the library should be available together with some of the most popular Ada 95 compilers.

From the web page (http://ascl.sourceforge.net/) --su]

Database Source Name Parser

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Wed, 15 Jun 2005 14:38:59 +0200
Subject: Update of data source name parser
Newsgroups: comp.lang.ada

The DSN library can be used to parse URL like database connection strings. News:

* unified grammar
* internationalization
* Ada 200Y version added

Assume GMGPL. The sources and docs are available at http://home.arcor.de/bauhaus/Tools/dsn.html

[See also the same topic in AUJ 26-1 (Mar. 2005), pp.11-13. --su]

Scout - Ada utility for Google Earth & NASA World Wind

From: Tom Moran <tmoran@acm.org>
Date: Sat, 30 Jul 2005 15:28:00 -0500
Subject: Ada utility for Google Earth, NASA World Wind
Newsgroups: comp.lang.ada

I've posted at home.comcast.net/~tommoran4/scout091.zip a zip file with readme.txt, Zoo.spc, and Scout.exe I'll add the Ada source code after a bit of cleanup.

Scout aids finding locations and paths with Google Earth or NASA World Wind by interconverting latitude & longitude decimal or degrees, minutes, seconds, State Plane Coordinates, Metes and Bounds (only straight lines for now). Given a list of points in any of those forms, it will create the Google Earth or World Wind path files to draw a path from point to point. It also can read Google Earth lat/lon format and convert to World Wind format, or vice versa. The current version is 0.91 and I'm adding to it (and will post that code).

Suggestions welcome.

Ada-related Products

AdaCore Implements New Ada Standard

Leading Ada technology provider brings Ada 2005 to software developers

http://www.adacore.com/pressroom_22.php

NEW YORK, August 29, 2005

AdaCore today announced the first implementation of the upcoming new version of the Ada programming language. Known as Ada 2005 and anticipated for official international standardization under ISO next year, the new definition advances the state-of-the-art in programming language design while meeting the goal of compatibility with earlier versions of Ada.

"AdaCore is leading the way with Ada 2005," said Robert Dewar, president of AdaCore. "Many of the new features will make our customers' programming jobs simpler, and the language's integration of object-oriented and concurrency facilities is truly innovative. By officially including the Ravenscar tasking profile, Ada 2005 will help users write portable high-integrity programs. Of course, because of Ada 2005's high degree of upward compatibility, our customers can use our latest development tools not only for the new Ada language version, but also for Ada 95 and Ada 83."

AdaCore has implemented many of Ada 2005's enhancements, including Java-like interfaces, 32-bit character support, and new standard libraries. These are currently available in the GNAT Pro development environment, as well as in the GNAT edition for the GNAT Academic Program (GAP). Aimed at spreading the use of Ada for teaching and research, GAP is an AdaCore initiative within the academic community. Ada

2005 offers many advantages as a language in computer science education, and the GAP program makes it easier for universities to bring Ada into their curricula. Many of the new features in the revision are also available in the GNAT GPL 2005 Edition, intended for Free Software developers.

About Ada

Ada is a modern programming language designed for large, long-lived applications - and embedded systems in particular - where reliability and efficiency are essential. It was originally developed in the early 1980s (generally known as Ada 83) and then revised and enhanced in an upward compatible fashion in the mid 1990s under the auspices of the International Organization for Standardization (ISO). The resulting language, Ada 95, was the first internationally standardized Object-Oriented Language and is currently seeing significant usage worldwide in the high-integrity / high-performance domains, including commercial and military aircraft avionics, air traffic control, railroad systems, and medical devices. Ada also serves as an excellent teaching language for both introductory and advanced computer science courses, and has been the subject of significant university research, particularly in the area of real-time technologies.

About Ada 2005

Ada 2005 offers a number of new capabilities while meeting the requirement of compatibility with Ada 95. Reflecting advances in Object-Oriented technology over the past decade, the language's enhancements include a Java-like interface mechanism and a syntax for inheritance that prevents accidental overloading. The needs of the real-time and high-integrity communities have been addressed; support in these areas include new task control mechanisms and the definition of the Ravenscar tasking profile in the standard. Additionally, Ada 2005 expands the predefined library (including generic "container" packages that improve upon C++'s STL) and makes a number of improvements in the access type area.

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/customers.php 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 - KC-767A Maiden Flight

Mission Control System Team Member AdaCore Lauds KC-767A Maiden Flight

http://www.adacore.com/pressroom_21.php

PARIS, Paris Air Show, June 13, 2005

AdaCore joins Boeing, Smiths Aerospace, and Wind River Systems in celebrating the maiden flight of Boeing's advanced aerial refueling tanker, the Italian Air Force KC-767A, which took place on May 21, 2005. AdaCore serves as a key member of Smiths Aerospace's development team for the tanker's mission control system (MCS), which manages the aircraft's unique flight guidance, navigation, and communications capabilities. The Italian Air Force was Boeing's first KC-767 customer, ordering four of the world's newest and most advanced tankers.

The KC-767 MCS is the first avionics application in flight to use the Software Common Operating Environment (SCOE), which consists of a safety-critical-certified ARINC653 operating system from Wind River, the GNAT Pro for VxWorks 653 Ada compilation system from AdaCore, and infrastructure software developed by Smiths Aerospace.

"AdaCore was chosen as a key member of the SCOE development team after an extensive evaluation of several Ada compiler vendors," said Dudrey Smith, manager of support software at Smiths Aerospace. "Over the multi-year development, AdaCore has proven instrumental in the success of both the 'SCOE' and the 767 Tanker program."

AdaCore was specifically selected by the MCS development team for its superior technical expertise with both Ada compilation systems and with avionics application development environments. With fast compilation speed, quality code generation, and an extensive set of switches and pragmas (compilation control directives), AdaCore's GNAT Pro specifically addressed the KC-767's need for mission-critical robustness and flexibility.

"The close cooperation between AdaCore and Wind River that enabled the successful development of the Boeing KC-767 mission control system demonstrates the value real partnerships bring to the development of device software," said John Fanelli, vice president of products, Wind River. "With device software complexity rising exponentially, only dedicated partnerships that provide true technology integration and optimization of best-of-breed products will provide companies like Smiths Aerospace and Boeing with the technology to deliver their products faster, better, at lower cost and more reliably."

In addition to flight-management tasks, the KC-767 MCS takes in such data as flight control, fuel storage and fuel flow as well as refuelling orbit patterns, rendezvous points, and other mission-specific data. The KC-767 MCS consists of two flight-management computers (FMCs) connected to multipurpose control display units (MCDUs). The VME-based MCS computer is implemented using a dual, open architecture with an ARINC653-compliant partitioned operating system that enhances safety-critical redundancy and isolation through temporal, physical and bandwidth segregation (schedule, memory and I/O throughput).

"The partitioned-OS tanker project is a perfect illustration of how mission-critical aerospace systems must be architected to ensure safety-critical compliance and reliability," said Robert Dewar, president of AdaCore. "The successful first flight of the KC-767 is the culmination of an exceptional effort put forth by the entire development team."

AdaCore - GNAT Pro on Intel Itanium 2

AdaCore Implements Ada on Intel Itanium 2-based HP Integrity Servers Running HP-UX 11i and Linux

Eases transition for customers upgrading from other platforms

http://www.adacore.com/pressroom_23.php

NEW YORK, September 6, 2005

AdaCore announced today the availability of its GNAT Pro Ada development environment on Intel® Itanium® 2-based HP Integrity servers running either the HP-UX or Linux operating systems. With GNAT Pro, Ada users can take full advantage of the performance, security, and flexibility offered by the HP servers. And Ada users with applications running on the AlphaServer and PA-RISC platforms can look forward to a smooth transition to HP-UX 11i and Linux on HP's Integrity servers.

GNAT Pro is the first Ada development environment available on the Itanium 2 architecture. Its implementation on HP-UX 11i and Linux for HP Integrity servers was conducted by AdaCore under a contract from HP.

"AdaCore's implementation of GNAT Pro on Itanium-based platforms sends an important message to the Ada market,"

said Robert Dewar, president of AdaCore. "First, Ada is clearly a significant language for major computer makers, and second, AdaCore's commercial open source software approach makes the most sense for supporting Ada users."

"HP Integrity servers provide an agile, standards-based platform for HP-UX and Linux, as evidenced by the more than 5,000 applications now available on these systems," said Don Jenkins, vice president of marketing, Business Critical Servers, HP. "AdaCore's port of their GNAT Pro Ada development environment to HP Integrity servers provides customers even more choices in enterprise solutions, and accelerates their journey toward becoming an Adaptive Enterprise."

AdaCore's GNAT Pro for HP-UX 11i v2 and Linux (Red Hat® and SUSE®) is immediately available from AdaCore on HP's Integrity servers.

AdaCore - New GNAT Pro Toolsuite for ERC32

http://www.adacore.com/pressroom_24.php

Paris, September 15, 2005

GNAT Pro for ERC32, a flexible cross-compilation environment supporting the Ravenscar tasking profile on top of a bare ERC32 computer, is now available. It is designed for mission-critical real-time space applications, including those that have to meet safety standards.

Developed under ESA (European Space Agency) sponsorship, AdaCore targeted the development environment to ESA's standard processor for spacecraft on-board computer systems, the ERC32, which is a radiation-tolerant SPARC V7 processor. Available host platforms are x86 Linux and SPARC Solaris.

"ESA favours, beside proprietary solutions, the existence of open source technologies provided that they are properly maintained." Explains Morten Nielsen, ESA Technical Officer, "GNAT Pro implements this policy with a competitive high quality compilation environment and run times for the space SPARC based radiation hardened CPUs that are used in current and future ESA space programmes."

The static and simple tasking model defined by the Ravenscar profile allows a streamlined implementation of the Ada run-time library directly on top of bare computers. Its reduced complexity, together with its configurability, make it an excellent choice for mission-critical space applications in which certification or small size is needed. Unused code and data can be removed from the final executable, helping reducing the footprint and simplifying the certification process.

The developer can choose from several predefined run-time libraries, each corresponding to a particular set of run-time Ada features, or, even more flexibly, configure a tailored library reflecting exactly the set of features that are used.

Also as part of the ESA contract, an extensive verification and validation campaign was carried out to meet the ESA requirements for such products. AdaCore has developed a comprehensive test suite that checks compliance with the Ravenscar profile and correct behavior of specialized features (such as the last-chance exception handler mechanism) and supplemental tools (such as the debugger).

IPL (www.ipl.com) were also involved in the development providing their AdaTEST 95 tool targeting the ERC32 compiler.

"We are very pleased with this development which more than ever opens the space market to Ada, and to the use of a state-of-the-art software development environment for mission-critical applications." - Dr. Jose Ruiz, AdaCore Project Manager.

About ESA

The European Space Agency is Europe's gateway to space. Its mission is to shape the development of Europe's space capability and ensure that investment in space continues to deliver benefits to the citizens of Europe. ESA has 16 Member States. By coordinating the financial and intellectual resources of its members, it can undertake programmes and activities far beyond the scope of any single European country.

What does ESA do?

ESA's job is to draw up the European space programme and carry it through. The Agency's projects are designed to find out more about the Earth, its immediate space environment, the solar system and the Universe, as well as to develop satellite-based technologies and services, and to promote European industries. ESA also works closely with space organisations outside Europe to share the benefits of space with the whole of mankind.

Aonix - ObjectAda for Windows Update

From: Owner-Intel-ObjectAda <owner-intel-objectada@aonix.com>
Date: Thu, 26 May 2005 20:18:14 -0700
To: intel-objectada@aonix.com
Subject: Intel-OA: New ObjectAda Update (1102V722-U18)

A new update for the Aonix ObjectAda for Windows product, 1102V722-U18, is now available at http://www.aonix.com/ada_patches.html.

All general product updates for the 7.2.2 release are cumulative. In other words, corrections and enhancements in this update will include those from all previous updates. Please see the Release Notes for further details on the corrections made and installation instructions. The release notes can viewed at

ftp://ftp.aonix.com/pub/adats/outgoing/1102/7.2.2/U18/1102V722-U18.Release_Notes

The upcoming ObjectAda for Windows 8.0 release will, of course, also contain any corrections and enhancements in this update.

From: Owner-Intel-ObjectAda <owner-intel-objectada@aonix.com>
To: intel-objectada@aonix.com
Date: Wed, 14 Sep 2005 16:29:01 -0700
Subject: Intel-OA: New ObjectAda Update (1102V722-U19)

A new update for the Aonix ObjectAda for Windows product, 1102V722-U19, is now available at http://www.aonix.com/ada_patches.html.

All general product updates for the 7.2.2 release are cumulative. In other words, corrections and enhancements in this update will include those from all previous updates. 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/1102/7.2.2/U19/1102V722-U19.Release_Notes

The upcoming ObjectAda for Windows 8.2 release will contain all corrections and enhancements in this update.

[See also "ObjectAda Update" in AUJ 25-3 (Sep. 2004), p.125. --su]

Aonix - ObjectAda tools to be used for Joint Strike Fighter Project

Aversan Selects Aonix® Software for Joint Strike Fighter Project

http://www.aonix.com/pr_08.16.05.html

San Diego, USA, August 16, 2005

Aonix®, a provider of safety critical and mission critical software solutions, announces that Aversan, a Canadian company offering real-time embedded solutions, selected Aonix ObjectAda® RAVEN™ safety-critical Ada95 environment, certifiable for DO-178B Level A certification, and AdaCAST testing tools. Aonix products will be used to support Aversan's work on the F-35 (Joint Strike Fighter) PTMSC Controller software design and embedded test software design and verification. Aversan selected Aonix tools based on selection and specification of ObjectAda for safety critical software design to MIL-STD-490.

"DO-178B Level A certification requires the most stringent safety-critical requirements," noted Ted Sherlock,

Software Manager at Aversan. “When we evaluate a supplier’s technology, we are primarily concerned with the quality of the products, the responsiveness of the subcontractor, and whether the software is technically adequate to meet our needs. Aonix fulfilled all of these requirements”.

Aversan’s selection of Aonix tools for safety-critical software design continues Aonix’ long successful history of delivering certifiable applications to both commercial and government safety-critical projects in avionics, space, high speed rail, and nuclear industries. Aonix gained its solid reputation in the safety-critical field by designing tools that comply with market standards and has provided safety-critical solutions to a myriad of commercial and defense projects including International Space Station, Boeing 777, Rafale Multi-Role Combat Fighter, C130-J Hercules, Airbus A 330-340, and NH90 Helicopter.

ARTiSAN - ARTiSAN Studio Version 6.0

ARTiSAN launches ARTiSAN Studio Version 6.0

Adds Ergonomic Profiling, DODAF profile, Change tracking, new code generation options.

www.artisansw.com/press/2005/v6.0.asp

Cheltenham, UK — July 13, 2005 — ARTiSAN Software Tools, Inc. has today launched ARTiSAN Studio version 6.0 which introduces more new features and functionality than any previous ARTiSAN release. ARTiSAN Real-time Studio has been renamed ARTiSAN Studio in line with ARTiSAN’s strategy to build an industry-wide UML 2.0 and SysML compliant tool.

At the core of the new product is the concept of Ergonomic Profiling, the ability to build a compelling environment based on the needs of specific domains or applications of UML. Using Ergonomic Profiling, ARTiSAN Studio will take on new menus and explorer windows based on the UML profiles the user is working with, including new icons, item types, and diagrams. This level of extensibility is unmatched in the industry, and enables ARTiSAN or its customers to provide multiple environments tailored to real customer need.

Also available with the introduction of ARTiSAN Studio 6.0 is an industry-leading profile for modeling compliant Department of Defense Architectural Frameworks (DoDAF). The DoDAF Profile customizes Studio's easy-to-use interface to provide an out-of-the-box solution for creating DoDAF-compliant models. The DoDAF profile allows modelers to describe DoDAF-compliant architectures using DoDAF diagrams and graphical notation, based on industry-leading UML 2.0 modeling techniques and technology. The complete range of framework products can be modeled in a single multi-user repository, ensuring architectural consistency and completeness across the Operational (OV), Systems (SV), and Technical Views (TV). This means a common tool can be used across an organization for both DoDAF and non-DoDAF deliverables, dramatically improving productivity and flow-down into implementation.

A further key and unique feature of ARTiSAN Studio 6.0 is the inclusion of multi-user Change Tracking, extending ARTiSAN Studio's powerful client-server repository in support of a quality-assured process. Based on the understanding that models are complex pieces of interwoven data, changes are automatically tracked on each model element, rather than an artificial granularity imposed by a Configuration Management (CM) tool. As ARTiSAN's customers state, multi-user development improves productivity by enabling collaborative teams to build coherent and consistent models fast and easily, a significant process improvement for high-level systems and software engineering work. ARTiSAN Studio scales from single laptop to large multi-site/multi-user configurations with ease.

In addition to extending its proven systems engineering capabilities to include the modeling of architectural frameworks, ARTiSAN has increased support at the implementation end. ARTiSAN Studio now offers choices of code generation: On-demand Code Synchronization (OCS), in which a user chooses when code is generated from, or reversed into the model; or Automatic Code Synchronization (ACS), which provides instantaneous synchronization of model and code. ACS allows one to work simultaneously in both ARTiSAN Studio and any IDE of choice. In keeping with ARTiSAN’s policy of open architecture and extensibility, both code generators are template-based, allowing users to modify the format and content of the generated code. 'Out of the box', ACS and OCS support C, C++, Java, and optionally Ada 83, Ada 95 and SPARK.

Other improvements include a web publisher for generating models that can be browsed using Microsoft Internet Explorer, a Component Sharing Wizard for improving the workflow associated with sharing parts of models in other models, as well as a host of customer requested usability and workflow enhancements.

Jeremy Goulding, CEO of ARTiSAN Software stated, "With the release of Studio 6.0, ARTiSAN raises the bar in the UML and SysML tools arena. The powerful concept of Ergonomic Profiling makes UML accessible to domain specialists and project stakeholders and also provides a robust and consistent solution for DODAF modelers thanks to the underpinning of UML 2 and SysML. Several of the new feature introductions, such as change tracking, SysML support, and the DoDAF Profile, come naturally to a tool that had its origins in systems modeling and further show the value of the tool's already highly acclaimed multi-user repository."

About ARTiSAN Software Tools

ARTiSAN is the leading supplier of collaborative modeling tools for requirements analysis, specification, design and development of complex applications. The company provides standards-based, multi-user tool support from architectural frameworks through systems design to software implementation. ARTiSAN offers products, services and a process for systems and software modeling to accelerate the development of next-generation real-time systems while ensuring that they always meet requirements. ARTiSAN enables engineering teams to visualize, design and validate systems before building them, and simplifies implementation with code generation and software reuse. Winner of the 2005 SD Times 100 award in the Modeling category for bringing UML and SysML to real-time embedded systems, ARTiSAN offers the most advanced tools for complex applications development.

ARTiSAN Software Tools, Inc., founded in March 1997, is privately held with headquarters in Cheltenham, United Kingdom. The company has regional sales offices and distributors throughout the world. For more information, call +44 (0)1242 229300 internationally, or 1(888)511-7975 from the US; or visit www.artisansw.com

ARTiSAN - Software Development on Meteor missile program

ARTiSAN Studio Selected by MBDA for Software Development on Meteor missile program

www.artisansw.com/press/2005/MBDA.asp

Cheltenham, UK — September 6, 2005 — ARTiSAN Software Tools, a global leader for UML 2 and SysML-based, real-time systems and software modeling tools, announced today that ARTiSAN Studio has been selected by the Seeker Division of MBDA, the world’s leading missile systems manufacturer, as their modeling tool for software development on the Meteor missile.

Jointly owned by BAE Systems, EADS and Finmeccanica, MBDA is the prime contractor on the six-nation Meteor missile program. Meteor is a highly maneuverable, fast, Beyond Visual Range

(BVR) air-to-air weapon system capable of operation by night or day, in all weather and in dense electronic warfare environments. The Seeker software design on Meteor, the most complex part of the missile, required engineers to take existing C code, re-purpose it and generate new functionality in Ada 95.

According to MBDA: "We looked at tools from several vendors and were immediately attracted by ARTiSAN Studio's excellent support for real-time software engineering and the strong integration with other software tools. Other factors influencing the decision-making process were its support for code generation for both Ada and C and the ability to combine them within the same UML model. ARTiSAN's continuing commitment to Ada and UML development was also an important factor in the selection."

An additional factor in the selection of ARTiSAN Studio was the ability of the tools to bring together the electronics and software components of the project design. The Hardware Architecture Diagram provided an intuitive view of the Electronics Architecture enabling the hardware engineers to readily define the bus interfaces, memory maps and details of how the electronics interfaced to the software.

Jeremy Goulding, President and CEO of ARTiSAN commented, "MBDA was extremely attracted to ARTiSAN's support for multiple languages within a single model, and particularly our pedigree with and support for Ada. Our continued commitment to support an Ada oriented process with UML2 and SysML modeling tools is a direct result of coupling the need to add the latest standards-based functionality, together with addressing the long-term support needs of our customers on major defense programs."

About MBDA

MBDA operates in all of the major world markets, and is the only company in its sector able to design and produce missile systems for land-based, naval and airborne requirements. The group offers customers high technology solutions and has unrivalled capability in key technologies. MBDA Seeker Division is an 'in-house' supplier of key missile elements. Its role is to design, develop, produce, and integrate RF seekers and fuzes and to merge existing capabilities and skills creating a 'Center of Excellence' and international skill pool. MBDA is jointly owned by BAE SYSTEMS (37.5%), EADS (37.5%) and FINMECCANICA (25%). For more information about MBDA, visit www.mbda.net

DDC-I - Upgrading DACS' Microsoft Visual Studio Support

Phoenix Arizona, March 1, 2005 -- According to DDC-I Senior Software Engineer Richard Frost, the company's proven DACS product has always offered support for objects created by Microsoft Visual Studio. A conversion utility was used to "fix up" an object file for compatibility with the DACS linker.

Unfortunately, later releases of Visual Studio - including .Net - have added incompatibilities in the LIBC library, as well as to the initialization code for the Run Time System (RTS).

"Addressing the increasing issues experienced by a growing customer base using a combination of development tools hosted on Windows became a priority," explains Frost, who's Windows experience includes leading the rehosting program for DDC-I's TADS product line.

With the DACS 4.7.16 release, DDC-I now provides two versions of the DACS RTS. The default product is built with the standard DACS assembler and fully compatible with the standard target linker provided with DACS. An optional RTS, built with Microsoft Visual Studio, is available to every DACS customer.

Due to legal limitations, DDC-I is not able to ship the "link.exe" component of Visual Studio. Instead, a template file is provided which calls the linker and uses the Microsoft compiled version of the DACS RTS, eliminating incompatibilities and providing the customer an easy integration path for Microsoft Visual Studio and .Net.

About DDC-I

DDC-I 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 applications 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.

DDC-I - DDC-I Joins the Eclipse Foundation

Offers Integrated Solution for Safety Critical Software Developers

June 1, 2005 – Phoenix, AZ – DDC-I, a global leader in safety critical software tools for embedded applications, announced today they have joined the Eclipse Foundation, and will market their SCORE development tools for VxWorks under the Eclipse Integrated Development Environment.

DDC-I joins the Foundation as an Add-in Provider, and is in the process of creating Eclipse-based products optimized for Wind River's VxWorks RTOS. The final product, due to release in Q3, will be an integrated solution which means one GUI / one feel under Workbench, Wind River's Eclipse based development environment.

"The fact that our solution will truly be integrated is what sets us apart from our competitors," states David Mosley, DDC-I Engineering Manager and SCORE® Product Champion. "This new product will offer world-class SCORE compilers & a wide range of third party tools being integrated into Eclipse."

"The goal of Eclipse is to create a universal development platform enabling global enterprises to develop software more efficiently," explains Eclipse Foundation Executive Director Mike Milinkovich. "DDC-I brings over twenty years of experience with safety-critical embedded system software to the Eclipse platform. Their SCORE product's close integration with Eclipse Strategic Developer Wind River's Workbench and VxWorks RTOS offers embedded developers even greater flexibility."

Founded in November 2001 by industry heavyweights IBM, Borland, Red Hat and others -- including Oracle, SAP and Intel -- the Eclipse Foundation's open source development environment is gaining traction in the enterprise market and reaching into embedded systems, where a number of vendors are adopting Eclipse technology.

Dedicated to providing an industry-wide platform for the development of highly integrated tools, the independent Eclipse "ecosystem" is built atop royalty-free technology and universal tool integration, using a plug-in based framework making it easier to create, integrate and utilize software tools. By collaborating and exploiting core integration technology, tool producers like DDC-I and Wind River are leveraging platform reuse and concentrating on core competencies to create new development technology for safety-critical software programmers.

DDC-I - DDC-I Tools for Ongoing Bradley Upgrades

Bradley Fighting Vehicle: Precision Targeting On The Cutting Edge

Key Subcontractors Harness DDC-I Tools for Ongoing Bradley Upgrades

May 02, 2005 - Phoenix, AZ - Developed and enhanced over the last twenty-five years, the Bradley Fighting Vehicle is a mainstay of American ground forces, delivering personnel, firepower and

battlefield telemetry in challenging environments around the globe. Earning a reputation as one of the finest fighting vehicles in the world, its mobility, survivability and shoot-on-the-move capability are the result of continual enhancement to meet and exceed the requirements of the changing battlefield.

DDC-I's proven DACS development tools continue to contribute directly to Bradley development, most recently responsible for embedded software for the turret drive and electronic transmission control units in the newest Bradley A3. Estimated to be an order of magnitude more challenging than previous Bradley programs, the A3 upgrade involved development of hardware and software by a large number of subcontractors located around the world. Efforts were magnified by the estimated 1.5M lines of code necessary to make everything go.

Safety-critical embedded code generated with DACS spins the improved A3 turret design and keeps the Bradley's gears turning. The turret's 360- degree continuous traverse enhances automatic dual target tracking, automatic gun target adjustment, automatic sighting, hunter/killer and day-and-night vision capabilities. The GM-Allison hydro-mechanical automatic transmission harnesses a 600 hp Cummins diesel engine, with a top land speed of 38 mph (61 km/h), amphibious capability and a cruising range of 250 miles.

In addition to standard combat roles, variants of the Bradley serve many others, from air defense support (M6 Linebacker) and troop and cargo transport (M993 Carrier) to the medical AMEV/AMTV (Armored Medical Evac/Treatment Vehicle). The M4 Command and Control Vehicle (C2V) provide commander and staff with a protected environment at the pace of today's armored forces. Providing protected transport of an infantry squad on the battlefield and watching over firefights to support dismounted infantry, the Bradley can suppress and defeat enemy tanks, reconnaissance vehicles, infantry fighting vehicles, armored personnel carriers, bunkers, dismounted infantry and attack helicopters.

Scouting, essential fire support, and laser-sighting missions are a crucial capability on the emerging 21st century battlefield. Incorporating the latest improvements, taking lessons learned during recent deployment of vehicles in the Middle East, the Bradley is engaged in ongoing project development and DACS remains involved.

DDC-I - DDC-I Releases v.6.2.0 for TADS-960

Offers New License and Installation Options

August 1, 2005 – Phoenix, AZ – DDC-I, a global leader in safety critical software tools for embedded applications, announces today a new release for the TADS Ada Development System targeting i960. TADS-960 for Windows V.6.2.0 includes the introduction of the FLEXnet 10.1 licensing system along with Distributed TADS for Windows and unbundled tools.

"The upgrades made in this version of TADS offer our customers a product that's easier to use, from a smooth installation to support for multiple users," states Stephen Hunter, Senior Software Engineer and TADS Product Champion. "We are dedicated to providing our clients with quality tools and services as well as customizing solutions as needed for their individual requirements," Hunter concludes.

FLEXnet 10.1 supports all of the latest licensing models including triple redundant license servers, MAC address host ID's, hardware dongles, license borrowing, and a host of other licensing models tailored to the customer's needs.

Distributed TADS for Windows allows a TADS-960 Universe to be installed on one Universe server, and then to be shared by any number of client installations on other Windows machines. This will allow different users on different machines to easily share projects and libraries making software project management easier to implement.

The Windows installation procedures have been re-engineered to allow for easy re-licensing of the product. The various licensing models supported can easily be changed through the installation procedure without the need to uninstall or reinstall the product.

The Stand-Alone AdaScope (debugger), Link Retargeting Kit, and Runtime Source Kits are now all available as separate shippable packages. Each may be purchased and installed separately or as part of the main development system.

DDC-I - TADS Windows for M680x0 & Mil-Std-1750A

New Upgrade Release for TADS Windows Targeting M680x0 & Mil-Std-1750A - Now Available

Offers upgraded licensing, shared projects and unbundled tools.

August 29, 2005 – Phoenix, AZ – DDC-I, a global leader in safety critical software tools for embedded applications, announces today a new release for the TADS Ada Development System targeting Motorola 68K and Mil-Std-1750A processors.

"This release completes V.6.2.0 upgrades to the entire TADS Windows Development Suite (TADS-i960, TADS-1750A, and TADS-680x0), states Stephen Hunter, Senior Software Engineer. "The upgrades offer our customers a product that is easier to use, from a smooth installation to support for multiple users," Hunter concludes.

Features:
FLEXnet 10.1 supports all of the latest licensing models including triple redundant license servers, MAC address host ID's, hardware dongles, license borrowing, and a host of other licensing models tailored to the customer's needs.

Distributed TADS for Windows allows a TADS-680x0 or TADS-1750A Universe to be installed on one Universe server, and then to be shared by any number of client installations on other Windows machines. This will allow different users on different machines to easily share projects and libraries making software project management easier to implement.

The Windows installation procedures have been re-engineered to allow for easy re-licensing of the product. The various licensing models supported can easily be changed through the installation procedure without the need to uninstall or reinstall the product.

Unbundled Tools: For TADS-680x0, the Stand-Alone AdaScope (debugger), Link Retargeting Kit, Protocol Retargeting Kit and Runtime Source Kits are now all available as separate shippable packages. For TADS-1750A, The Math Package, Stand-Alone AdaScope (debugger), Emulator Support Kit (PDU), Link Retargeting Kit, Protocol Retargeting Kit, and Runtime Source Kits are also all available as separate shippable packages. Each may be purchased and installed separately or as part of the main development system.

DDC-I - SCORE® Gives State-of-the-Art Ada Support to VxWorks

As a standalone product, SCORE® (DDC-I's Safety Critical Object-oriented Real-time Embedded Software Development Environment) has been able to meet the needs of most of DDC-I's customers. It's ability to seamlessly integrate and debug code written in multiple languages (Ada 95, C, Embedded C++, and FORTRAN), together with a small-real time kernel has provided a stable platform on which applications are built. Having a small real-time kernel is a significant advantage of SCORE®, but at the same time it's smallness is a liability to customers wanting more device drivers, communication stacks, disk management, and other middleware functions.

Now SCORE® has been integrated with VxWorks (Wind River's leading Embedded Real-Time Operating System)

- you can get the proven SCORE® code generation ability plus all the rich features and middleware of the top-of-the-line RTOS. Not only has this integration been done on the target, but the development environments are integrated as well. No more launching one tool from another; the entire edit, build, and debug activities are all controlled from within Workbench - the new Wind River Integrated Development Environment (IDE).

ICS - BX/Ada V6

From: Mark <mhatch@ics.com>
Date: 10 Aug 2005 12:29:27 -0700
Subject: Announce: BX/Ada - Motif GUI builder for Ada w/Motif bindings
Newsgroups: comp.lang.ada,comp.windows.x.motif

ICS announces the release of BX/Ada V6 which augments BX PRO, the industry leading GUI builder, with the ability to generate Ada code. BX/Ada provides a complete solution for GUI developers using Ada. It includes all the features of BX PRO, plus a set of Motif/Ada bindings, support for most major Ada compilers and 15 add-on Motif widgets from EnhancementPak for developing top-quality user interfaces.

This release extends to Ada developers all the conveniences in creating user interfaces that BX PRO has provided C/C++ developers for years. Features such as WYSIWYG development, resource editors, styles, support for localization and internationalization, integration of third party widgets, and more are now available to Ada developers who will be able to generate clean, readable and portable Ada code with the push of a button.

For more information or an evaluation copy of BX/Ada, visit http://www.ics.com/products/motif/bxada

PegaSoft - BUSH AdaScript Business Shell

From: dan <dan@cogeco.ca>
Date: Wed, 31 Aug 2005 13:24:21 -0400
Subject: BUSH Business Shell announcement New Version 1.0.3
Newsgroups: comp.lang.ada

BUSH (Business Shell) combines the capabilities of BASH, PHP, GCC, and databases into a uniform design for rapidly building secure, reliable Web sites. Based on an ISO standard, it promotes code reuse: scripts and templates can be compiled with GCC or ported to JVM or .Net using third party tools with only minor changes. It can also replace BASH as an interactive command shell with SQL support, and is a general purpose scripting language. BUSH supports Linux, FreeBSD, PostgreSQL and MySQL.

Changes for version 1.0.3: New built-in GNAT-compatible directory_operations package. Serious memory leak problem with pragma prompt_script repaired that could cause an interactive session to crash.

The BUSH home page is at http://www.pegasoft.ca/bush.html

[See also same topic in AUJ 26-1 (Mar. 2005), p.17. --su]

Praxis HIS - Merge of Praxis High Integrity Systems and Aspect Assessment

The Merger of Praxis High Integrity Systems and Aspect Assessment

14 September 2005

UK based Praxis High Integrity Systems has announced its intention to merge with Aspect Assessment. The merger confirms the firm's position as the foremost world specialist in critical systems engineering. The new firm will continue to trade as Praxis High Integrity Systems.

The firm will provide high value services in such areas as Ultra Low Defect Software Development, Systems Safety, Systems Security, Advanced Systems Engineering Tools, Independent Safety Assessment and Notified Body Services. A good example of Praxis High Integrity Systems' capability is provided by its recent work for the US. National Security Agency where it developed a demonstrator biometrics system that was independently tested and zero software defects were identified. The Praxis approach to developing critical systems has been cited by the US National Cyber Security Partnership (NCSP) as one of only three approaches world-wide which is capable of developing the ultra low defect software needed to improve US Cyber Security. The US NCSP was formed in response to the White House National Strategy to Secure Cyber Space. Aspect Assessment focuses on the independent risk evaluation of critical systems – a powerful complement to the core services offered by Praxis. Recently, Aspect completed the safety and conformance approval of the Tilt Authorisation and Speed Supervision (TASS) system for Virgin's tilting Pendalino trains. Tilting trains are required on the UK West Coast Mainline between London and Glasgow in order to reduce journey times, and the TASS system ensures they tilt safely on specific sections of the railway line.

Keith Williams, Managing Director of Praxis High Integrity Systems, says: "Praxis High Integrity Systems has over twenty years of market-leading expertise, delivering very high integrity, ultra low defect systems for applications such as Air Traffic Control, Weapons Systems, Railway Signalling, Telecommunications and Secure Financial applications. "

"Now, with the added expertise of Aspect Assessment, we have the very strongest capability in safety and environmental engineering. It allows us to offer a world beating approach to Independent Safety Assessment, which is a service line we remain strongly committed to. It is a most formidable combination."

Praxis HIS - SPARK and SCADE Suite for Rapid Development

Praxis High Integrity Systems and Esterel Technologies Commit to Integrate SPARK and SCADE Suite for Rapid Development Compliant with Def. Stan. 00-56 Requirements

BATH (UK) and ELANCOURT (France) — June 15, 2005

Praxis High Integrity Systems—a world-leading company providing products and services for the engineering of high integrity systems—and Esterel Technologies—a world-leading supplier of model based-design, validation, and verification tools for safety-critical embedded software applications—today announced their collaboration to combine the SPARK toolset from Praxis HIS with the SCADE Suite toolset from Esterel Technologies.

This tool integration will combine SCADE Suite's safe design capture, simulation, formal design verification, with the widely used SPARK programming language and associated analysis tool, the SPARK Examiner. By combining the best in modelling, code generation and language-based analysis, this integration will effectively enable rapid development that is in full compliance with Defence Standard 00-56 objectives and other demanding standards.

"Given the rapidly growing market share of SCADE in both Europe and the United States, Praxis HIS is delighted to collaborate with Esterel Technologies on this tool integration. It will allow our joint customers to benefit from both Esterel Technologies and Praxis HIS experience in safety- and reliability- critical systems, in particular for those designing systems that must comply to Defence Standard 00-56" said Peter Amey, Praxis's Chief Technical Officer.

"As we are deploying SCADE towards same market and customers as Praxis HIS with SPARK, it was an obvious move, for the best interest of our respective customer base, to cooperate with Praxis and combine our technologies and skills better to address specifics of safety critical systems. In particular we are very pleased to be able to provide a consistent flow to UK military accounts, with a fully

compliant Def. Standard 00-56 design flow" said Eric Bantegnie, the Esterel Technologies' CEO.

The SPARK language and supporting tool set provide the most precise environment for the implementation and verification of high-integrity software. The SPARK Examiner allows vital safety and security properties of the implementation to be rigorously demonstrated and provides an industrial-strength technology for proving that all potential run-time errors have been eliminated.

SCADE Suite, a design environment for safety-critical embedded software applications, provides graphical specification capture and simulation, along with the capability to check safety properties at the model level. In addition, the SCADE KCG qualified code generator has achieved compliance with the highest safety standards, IEC 61508 and RTCA DO-178B, enabling rapid deployment of SCADE-generated designs.

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 enables 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.

About Praxis High Integrity Systems

Praxis High Integrity Systems has developed a global reputation in the fields of systems and requirements engineering, software development, safety assurance, information security and risk management and works with many of the leading aerospace companies and other organizations. The Company's roots are in the application of sound engineering principles to the development of high-integrity software systems whether safety-, business- or security-critical. Its unique approaches, tools and products have evolved from practical experience in the most effective approaches to developing such systems. The Company has now diversified into several new markets including financial services, telecommunications, utilities and automotive. For more information, please visit www.praxis-his.com.

Ada and CORBA

Development version of PolyORB

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 31 Aug 2005 01:22:23 -0700
Subject: Re: PolyORB, GCC, and Debian (was Re: Recompiling?)
Newsgroups: comp.lang.ada

Vadim Godunko wrote:

> Current development version of PolyORB require at least GCC 3.4.1. May be it will be work with GCC 4.0.2. :(

This is bad news, but it seems that PolyORB 1.2r can be compiled with GNAT Pro 3.16a1, so perhaps 3.15p may be able to compile it as well, with some patching.

I don't think that a package maintainer would be very interested in following the CVS HEAD of PolyORB, unless they were a PolyORB developer themselves.

> The DSA personality (replacement of GLADE) may work with GCC 4.0.2 if someone integrate several related patches from GCC HEAD. Even you are use PolyORB's DSA personality you need the gnatdist program from GLADE (the current development version of GLADE does have the correct version of this tool).

This sounds like a packager's nightmare. Personally, I have rather negative views on the way AdaCore does configuration management. They seem to go out of their way to make packager's jobs more difficult.

Note that the reason why I didn't package PolyORB is that it is a generic middleware; I felt that I had to provide either all personalities, or none at all. Providing no personality was easier :) Also, I don't use it myself, and I would have done a lousy job of supporting it for Debian users.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 31 Aug 2005 01:25:13 -0700
Subject: Re: PolyORB, GCC, and Debian (was Re: Recompiling?)
Newsgroups: comp.lang.ada

(Posted on behalf of Vadim Godunko)

Current stable version of PolyORB is 1.3a. I think AdaCore release 1.3r (publicy available version) in near future.

This version have many code cleanup and remove many GNAT 3.15p workarounds. Thus it require GNAT Pro 5 or GCC 3.4.

I think the right way is provide only stable personalities, for now (PolyORB 1.3):

- application: CORBA MOMA
- protocol: GIOP SOAP
- services: Event IR Naming Notification Time

The DSA personality still unstable up to 1.4. More whan, it require synchronization between it support inside GNAT compiler, GLADE's gnatdist and PolyORB. :(This is too complex task for provide it outside of AdaCore products. :(

From: Jerome Hugues <hugues@nephilim.enst.fr>
Date: Wed, 31 Aug 2005 09:55:54
Subject: Re: PolyORB, GCC, and Debian (was Re: Recompiling?)
Organization: ENST, France
Newsgroups: comp.lang.ada

Note we got some feedback of people playing with PolyORB, DSA and GCC 4.0.1, so things are possible, given enough resources are provided.

Note that providing a personality means providing a few fields to configure, and packaging the libraries that are built, there is no other black magic behind. The process might be similar to other packages. The libraries are loosely coupled, and the dependences are easy to track after some analysis.

You can join polyorb-users@ to discuss these issues if you're interested.

[See also "ACT - PolyORB 1.1r" in AUJ 25-3 (Sep 2004), p.124-125. --su]

Ada and GNU/Linux

GNAT and Fedora Core 3

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Date: Tue, 28 Jun 2005 19:56:04 GMT
Subject: Re: gnat and fedora these days ?
Newsgroups: comp.lang.ada

Reinert Korsnes wrote:

> Did anybody try "gnatmake" under an up to date (28 June 2005) Fedora Core 3?

Yes, and it works.

(To be honest, my system is *almost* up-to-date. I just now updated selinux-policy-targeted to version 1.17.30-3.13, but then I could hardly run anything at all because dynamic libraries couldn't be loaded, so I quickly reverted to 1.17.30-3.9. That's obviously not the same problem as yours.)

Have you tried it with GCC 4?

From: Reinert Korsnes <reinert.korsnes@chello.no>
Date: Wed, 29 Jun 2005 14:13:50 +0200
Subject: Re: gnat and fedora these days ?
Newsgroups: comp.lang.ada

>> Did anybody try "gnatmake" under an up to date (28 June 2005) Fedora Core 3 ?

> Yes, and it works.
(To be honest, my system is *almost* up-to-date. I just now updated selinux-policy-targeted to version 1.17.30-3.13, but then I could hardly run anything at all because dynamic libraries couldn't be loaded, so I quickly reverted to 1.17.30-3.9. That's obviously not the same problem as yours.)

Under kernel-2.6.11-1.27_FC3 and previous it *did work* for me also. The problem started yesterday after I did an "up2date". If I start with the previous kernel, then I again can use Ada on Fedora. So the problem seems to be associated with the very very last updates....

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Date: Wed, 29 Jun 2005 20:45:58 GMT
Subject: Re: gnat and fedora these days ?
Newsgroups: comp.lang.ada

> Under kernel-2.6.11-1.27_FC3 and previous it *did work* for me also.

Oh, the kernel! I forgot that I haven't rebooted after the latest kernel update. 2.6.11-1.27_FC3 is still running. Sorry.

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Date: Wed, 29 Jun 2005 23:27:42 GMT
Subject: Re: gnat and fedora these days ?
Newsgroups: comp.lang.ada

>> Under kernel-2.6.11-1.27_FC3 and previous it *did work* for me also.

> Oh, the kernel! I forgot that I haven't rebooted after the latest kernel update. 2.6.11-1.27_FC3 is still running. Sorry.

Under Linux 2.6.11-1.35_FC3 I get the same error as Reinert when I run Gnatmake on a tiny test program - Storage_Error at a-textio.ads:53:9.

This is line 53 of a-textio.ads:

```
type File_Mode is
 (In_File, Out_File,
  Append_File);
```

From: brian.b.mcguinness@lmco.com
Date: 29 Jun 2005 05:26:37 -0700
Subject: Re: gnat and fedora these days ?
Newsgroups: comp.lang.ada

I tried the same thing last night, with similar results.

Among other things, Fedora Core 3 kernel 2.6.11-1.35_FC3 breaks gcc (GCC) 3.4.3 20050227 (Red Hat 3.4.3-22.fc3), which I use to compile Ada programs, and the APLX 1.1 APL interpreter.

They seem to work OK with kernel 2.6.11-1.27_FC3.

I recommend keeping an older kernel version that is known to be good whenever upgrading to a new kernel, as this sort of thing happens from time to time.

From: Duncan Sands <baldrick@free.fr>
Date: Thu, 30 Jun 2005 08:43:13 +0200
Subject: Re: gnat and fedora these days ?
Newsgroups: comp.lang.ada

Maybe it's all that non-executable stack / stack randomisation stuff that I hear Red Hat use in their kernel?

Ada and Microsoft

AIDE - Ada Instant Development Environment for Windows

From: Stephane Riviere <stephane@rochebrune.org>
Date: Mon, 22 Aug 2005 15:07:55 +0200
Subject: Lettre d'information n°3 - AIDE - Ada Instant Development Environment
Organization: Rochebrune
Newsgroups: fr.comp.lang.ada

[Translated from French.] AIDE 1.03 is now available!

AIDE is an Ada development environment for Windows, which uses multi-platform libre-software utilities only for creating multi-platform applications as either libre or commercial purposes, in textual and/or graphical mode and also for use via Internet.

AIDE is intended for teaching and development. It can be used by multiple programmers concurrently and all of their specific environment and preferences will be separately and individually maintained.

AIDE permits to select among three development environment options: a general IDE based on Emacs (Glide); an Ada-specific graphical IDE (GPS); a simple command-line mode (MSys, Bash compatible).

AIDE is unique thanks to its licence, integration, absence of negative impact on the system:

- all the necessary utilities are libre software, pre-integrated and already configured;

- AIDE is immediately usable after a simple installation;

- no modification whatsoever to the system directories;

- no system variable created nor modified outside the inner scope of AIDE.

AIDE includes a production chain for complete and cohesive documentation, based on Texinfo, which generates from a single source all in-line help folders (HTML) and all user manuals (PDF).

AIDE uses the AdaCore Ada technology (http://www.adacore.com).

Main novelties

AIDE is installed by means of a multi-volume installation program.

Simplified packaging: AIDE itself and AIDE-SRC (the collection of all its sources).

Simplified configuration: one single directory for Glide, GPS and MSys.

New GPS version (2.1).

OpenGL was integrated in AIDE with numerous examples (all tested).

A Texinfo module has been added to the AdaDoc document generator, which permits to automatically include any specification within Texinfo documentation. The source comments may embed Texinfo instructions for improving presentation.

Important update of the user manual, which currently contains 110 pages.

Two new chapters were added. Translation to English has started.

Program examples and associated documentation were all updated. The Ada 95 reference manual was added. Numerous other updates included.

Integration of various utilities (Unix compatible safeguard multi-volume for Afio and CD-R burning with cdrecord and mkisofs).

Download from: http://stephane.rochebrune.org -> rubrique AIDE.

[See also same topic in AUJ 23-3 (Sep 2002), p.18 and "Ada Starter CD-ROM for MS-Windows" in AUJ 24-1 (Mar 2003), p.25. --su]

A# and the PocketPC

From: Rob Veenker <veenker@xs4all.nl>
Date: Sun, 04 Sep 2005 23:55:20 +0200
Subject: Re: SIGAda Workshop
Newsgroups: comp.lang.ada

> > Has anyone built an application with A#?

> Depending on what you call an application. I built an Hello World like A# program and run it on GNU/Linux Mono. Worked fine. I wanted to have it executed on a PocketPC platform, but I was never able to do so...

The .Net compact framework indeed has limitations that also affect the A# runtime. A special monitor class was added to allow tasking to work with A# as well. I have successfully built applications for the .Net compact framework using A# and run on a PocketPC device (including the emulated device that comes with the Visual Studio :-)

As I recall there was a problem with A# for the .Net compact framework regarding the reference to the right mscorlib for PocketPC.

[See also "A# for Mono" in AUJ 25-4 (Dec 2004), p.197. --su]

References to Publications

DDC-I Online News

[Excerpts from the table of contents. See elsewhere in this News section for selected items. -- su]

From: jc <jcus@ddci.com>
To: 27 May 2005 Online News US <jcus@ddci.com>
Date: Mon, 2 May 2005 14:36:45)
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News - Real-Time Industry Updates - May 2005, Volume 6, Number 5 - [http://www.ddci.com/news_vol6num5.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

This Month:

* DDC-I in Action
Bradley Fighting Vehicle: Precision Targeting On The Cutting Edge

* In the News
Things are Heating Up

* Partner Update
Wind River 2005 Worldwide User Conference

* Tech Talk
Ada Subunits and Efficiency in SCORE®

* Something to Think About
Positive Deviants

From: jc <jcus@ddci.com>
To: 28 June_July 2005 Online News US <jcus@ddci.com>
Date: Fri, 3 Jun 2005 0:50:15
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News - Real-Time Industry Updates - June/July 2005, Volume 6, Number 6 - [http://www.ddci.com/news_vol6num6.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

This Month:
* For Immediate Release - DDC-I Joins the Eclipse Foundation

Offers Integrated Solution for Safety Critical Software Developers

* In The News
What Consumes Most of an Embedded Developer's Time?

* Tech Talk
The Ada Business of Importing and Exporting

* Something to Think About
Leave 'em Laughing

From: jc <jcus@ddci.com>
To: 31 Aug_Sept 2005 Online News US <jcus@ddci.com>
Date: Tue, 2 Aug 2005 1:03:11
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News - Real-Time Industry Updates - August/September 2005, Volume 6, Number 7 - [http://www.ddci.com/news_vol6num7.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

This Month:
* DDC-I In Action - Aircraft Condition Analysis and Management System (ACAMS)

SCORE(R) from DDC-I Helping Develop "Health-Conscious" Planes

* For Immediate Release
TADS-960 v.6.2.0 Now Available

* In The News
Programming for Parallel Processors

* Something To Think About:
We are Happier When We're in Control

SPARK in IEEE Spectrum Magazine

Praxis High Integrity Systems Press Release

Leading technology publication praises Praxis High Integrity Systems

Global recognition for UK Company from IEEE Spectrum

Praxis High Integrity Systems (Praxis HIS), a world-leading company providing products and services for the engineering of high integrity systems, has received high praise and global recognition from IEEE Spectrum magazine, the flagship publication of the Institute of Electrical and Electronics Engineers (The IEEE), in its September issue.

In the article, called 'The Exterminators', contributing editor Philip E. Ross features the British firm, highlighting their approach to the development of ultra-reliable computer systems. The article showcases the MULTOS CA—an ultra-secure smartcard certification authority that was developed by Praxis HIS for MasterCard International.

Praxis HIS has developed a world-class reputation within the software world for producing near defect-free software. Using mathematically based techniques, known as formal methods, Praxis HIS builds software with a simple creed: that defects should be systematically tracked down and ruthlessly exterminated during all stages of a project.

Although Praxis HIS is tiny when compared to the software giants such as Microsoft, Oracle and SAP it has developed some highly complex and customized systems that need to be incredibly reliable. This includes air-traffic control systems and, more recently, a highly secure demonstration system for the NSA—the U.S. signals intelligence and cryptographic agency.

Commenting on Praxis HIS, John C. Knight, Editor-in-Chief of IEEE Transactions on Software Engineering and Professor of Computer Science at the University of Virginia, said, "They're very, very talented, with a very different approach." Praxis's founders, he says, believed building software wasn't as hard as people made it out to be, "They thought it isn't rocket science, just very careful engineering."

Peter Amey, Praxis Chief Technical Officer commented, "We're delighted that IEEE Spectrum has chosen to highlight the success of Praxis HIS. To be the focus of such a prestigious publication is a real coup for everyone connected with the company."

IEEE Spectrum, with more than 360,000 readers in over 150 countries, explores the development, applications, and implications of new technologies and trends in engineering and science, providing a forum for understanding, discussion, and leadership in these areas.

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 Inside

Indirect Information on Ada Usage

[Excerpts from and translations of job-ads and other postings illustrating Ada usage around the world. -- su]

Date: 15 Jul 2005 14:00:00

USA - Ada Software Design Engineer

Experienced Design Engineer needed for development of embedded real-time

software for electronics manufacturer. Permanent full-time position in a challenging R&D environment. Our client seeks candidates with very strong technical skills and those with engineering or comp. sci. degrees are preferred, though this is not mandatory.

Required Skills:

These include:

* Minimum of 5 years experience developing embedded real-time system software using Ada. Prefer those candidates who also have C and or C++ as well as assembly language experience

* Must have excellent code-documenting skills.

* Must have experience using oscilloscopes, in-circuit emulators, logic analyzers and real-time debuggers.

* Familiarity with performing formal testing/integration a plus.

* Safety critical software and system design skills preferred.

* Knowledge of hardware, ability to read hardware schematics and ability to perform minor hardware debugging a plus.

* Prefer applicant to have been through a full product life cycle from concept, requirements captured through design, implementation and testing phases.

Date: Fri, 10 Jun 2005 11:53:03 +0200

France - Development of Industrial and Embedded Distributed Real-Time Systems

[…] engineering company leader in industrial, scientific and technical applications now entering the aeronautic, transport, automotive, space and avionic domains, has openings for new projects. In the respective sector, your mission will be:

- the conception and realisation of distributed real-time systems for industrial on-board environments (Ada, C)

- the development of software components
- the conception and realisation of development support utilities
- the participation in the system-level integration and the preparation for deployment. […]

Date: 24 May 2005 10:46:26 -0700

USA - Ada Certification Tests

[We are] looking for experts to work on the development of an Ada 95 certification test. We provide skill-based tests that measure a person's core knowledge in a specific field. [...] We have 450 certification tests in several categories including IT, Management, Financial and Health Care. [...]

Date: Wed, 13 Jul 2005 18:29:36 +0200

VMS specialist.

PostFinance is a dynamic organization in the Swiss financial industry, offering services to over two million customers.

Your responsibility: for the "Customer Core" domain (central systems) we look for a Software Engineer to reinforce our team. You are jointly responsible for further development of the application process control in the context of other projects. You work independently in the entire development cycle (design, programming, module test, support) and are jointly responsible for the safe migration and upgrade of the application.

Your profile: you are a computer scientist or engineer (FH/ETH/UNI) or possess an equivalent training. You have several years of experience in realizing large computer projects, specialized knowledge in software engineering as well as very good OpenVMS experience.

Knowledge of the programming language Ada is an advantage. [...]

[...] The work place is Bern.

Date: 23 Aug 2005 10:01:22 -0700

Italy - Engineer for Ada on-board software unit testing

A software engineering company with a strong presence in the international markets and high growth profile looks for a software engineer to be responsible for the unit testing process of real-time, safety-critical, on-board software for avionics applications (helicopters). She/He must be able to work independently with minimal supervision.

Responsibilities

Software tests specification, planning and execution. Testing team coordination. Documentation of the test process and results.

Qualifications and Experience

Engineering academic degree or relevant professional experience; Must have excellent documentation skills and master of the English language. Very good technology skills across Ada software development Very good technology skills across application testing (including unit testing and test specification for real-time applications). Must have Knowledge in Ada 83 and/or Ada 95. Must have Knowledge in AUnit, Adatest or other Ada testing framework. Must have a minimum 2 years of work experience

Ada Jobs

From: Flavius Vespasianus
Date: Wed, 20 Jul 2005 01:29:10 GMT
Subject: Ada Jobs
Newsgroups: comp.lang.ada

I have always wanted to work on an Ada project. While working for a computer vendor, I provided consulting to some customers (now no longer in existence) using Ada. I've written Ada programs for learning purpose. However, most of my work has been in C++ for the past 15 years.

The reason for my interest is that I also do a lot of Object Pascal work and I find that in spite of my very neat and careful C++ coding, whenever I port to Object Pascal I discover a lot of errors in my C++ code. I'm interest in seeing on a large scale any Ada improved reliability.

The problem is I can never find any Ada consulting work, let alone any where they would take someone for a first real Ada project.

Is anyone really using Ada in Northern NJ?

From: Jeffrey Carter <jrcarter@acm.org>
Date: Wed, 20 Jul 2005 02:31:48 GMT
Subject: Re: Ada Jobs
Newsgroups: comp.lang.ada

I have certainly seen Ada jobs advertised in NJ (not sure if they're N or not). Right now there are 3 on dice.com. Be sure to search for 'ada "software engineer"' or you'll get every job with an Ada disclaimer.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Wed, 20 Jul 2005 02:46:58 GMT
Subject: Re: Ada Jobs
Newsgroups: comp.lang.ada

I have certainly seen Ada jobs advertised in NJ (not sure if they're N or not). Right now there are 3 on dice.com. Be sure to search for 'ada "software engineer"' or you'll get every job with an Ada disclaimer.

To be more precise, a search on Dice turns up 3 listings in NJ. I didn't look at the details. In practice, you'll see a lot of "C++, Ada a plus" listings. I don't know what that means, but it may mean they're injecting errors into perfectly good systems by rewriting them in C++. It should be a crime to take such a job. But 1/3 or more of the listings are usually real Ada jobs.

From: rcarnesiii
<reid_carnes@yahoo.com>
Date: Wed, 17 Aug 2005 14:42:43 -0400
Subject: Re: Ada Jobs
Newsgroups: comp.lang.ada

There are tons of Ada jobs in South Jersey, the main part of the air traffic control system is being re-written and they're doing it in probably 80% Ada, we're talking about several million lines of code at least !

From: Marin David Condic
<mcondic@acm.org>
Date: Thu, 21 Jul 2005 12:28:32 GMT
Subject: Re: Ada Jobs
Newsgroups: comp.lang.ada

> Only hope for more Ada jobs is to have more open source Ada projects start. This will make Ada become popular again, and will get more people

interested in it, and eventually this will carry over to commercial work.

I'd suggest that generating lots of open source Ada projects is a rather circuitous route to more Ada jobs. Its rather uncertain that there is any direct correlation between lines of open source code available in a language and jobs using that language. At best, that could take a long time before it had the desired effect.

Much more likely to create Ada jobs is to create and market some kind of product in which Ada is a factor. If a guy builds a Furby and programs it in Ada and gets a market going for Furbys, he hires more Ada programmers. Even if its an "internal product" - something used by your company in its critical processes - it creates jobs. The point is that if you have numerous people using some end product that they consider valuable, they come up with the money to further its development - and that makes jobs. It doesn't really matter if the product has open source code or extremely proprietary code. What matters is that it is a _Product_ that is _Useful_ and for which people will exchange _Money_. Otherwise, its all just interesting academics that will never produce a job for anyone.

Ada in Context

Ada 2005 Language Reference Manual

From: Randy Brukardt <randy@rrsoftware.com>
Subject: Re: Ada 2006 working documents
Date: Fri, 10 Jun 2005 14:12:00 -0500
Newsgroups: comp.lang.ada

Ludovic Brenta wrote:

> > A new version (draft 12) should be up in the next week; watch for it.

> I was reading draft 11 and noticed that it is technically an Amendment to ISO 8652:1995(E). Is it anticipated that ISO will publish a new Edition, i.e. ISO 8652:2006?

That's up to ISO, not us. I've asked that WG 9 consider making a recommendation, but WG 9 has not yet considered that request.

From: Ludovic Brenta <ludovic.brenta@insalien.org>
Date: Sat, 11 Jun 2005 11:16:18 +0200
Subject: Re: Ada 2006 working documents
Newsgroups: comp.lang.ada

[I'm forwarding a response I received from Dirk Craeynest:]

As you may remember, Ada-Europe is sponsoring the production of a s.c. "Consolidated Reference Manual", i.e. the work that Randy is doing to produce the RM and ARM drafts that you refer to.

In one of the letters I sent to the Ada-Belgium members, I wrote:

"The work on the next Ada language standard is progressing nicely and is on track. The Ada-Europe organization, of which you are an indirect member through Ada-Belgium, is supporting the standardization process in various ways, among others by working to assure the new standard will be freely available on-line and will also be published by Springer in the LNCS series, plus giving extensive coverage in the Ada User Journal of the vision, strategy and progress of the s.c. Ada 0Y language revision process. [...]"

What I can tell you as well, is that at the next meeting of WG9 (later this month in York), one item on the agenda is a motion to request ISO to publish this consolidated reference manual (in addition to the amendment itself). But that decision is of course up to ISO to take...

In any case, Ada-Europe is working to get the consolidated reference manual printed in Springer's Lecture Notes in Computer Science series, which will make the RM easily available in printed form world-wide.

Ada and Supercomputing

From: jtg <jtg77@poczta.onet.pl>
Date: Thu, 23 Jun 2005 10:03:09 +0200
Subject: Re: Ada-friendly MPI
Newsgroups: comp.lang.ada,comp.parallel.mpi

> I am looking at getting into a LAM-MPI system running Ada 95 on Linux. Does anyone have any learning / reference info suggestions? Books? PDFs? etc?

I was looking for such a solution for a long time until the obvious came to my mind: MPI calls can be executed from C code and everything else can be in Ada. You can write one-line function in C for every MPI call you want to use, then gather these functions in separate .c file and call them from Ada code. Simple, quick and portable (if you support the right .c file for each platform). What is more, you can use all the abundant learning/reference info intended for C users.

From: jtg <jtg77@poczta.onet.pl>
Date: Thu, 23 Jun 2005 15:08:46 +0200
Subject: Re: Ada-friendly MPI
Newsgroups: comp.lang.ada,comp.parallel.mpi

> Why not use the Distributed Annex (Annex-E) and the Ada's built-in parallelism support?

As far as I am concerned Ada Distributed Annex is non-standard. :-(All three supercomputers, that I had opportunity to use, supported MPI only or MPI/PVM. I had no clue how to use Ada built-in parallelism in these environments. But I am interested in the subject. Do you know about environments where Ada Distributed Annex is supported (and used)? Do you use it yourself?

From: jtg <jtg77@poczta.onet.pl>
Date: Thu, 23 Jun 2005 22:26:17 +0200
Subject: Re: Ada-friendly MPI
Newsgroups: comp.lang.ada,comp.parallel.mpi

> Depending on what you mean by standard. Ada 95 is an ISO standard and the distributed annex is part of it :)

So it's a pun: a standard that is not standard :-) (An ISO standard that is not standard feature in supercomputer environments)

> What are your supercomputers ? Is that a cluster ?

1. Cluster 15 x POWER2 RS/6000
2. Cluster 128 x PIII Xeon
3. I don't remember.

But all that was several years ago :-)

> I've used it myself for experimentation on Windows, GNU/Linux and Solaris. If your supercomputers are running GNU/Linux I don't see why this won't be working. What compiler are you using ?

In supercomputer environments many problems arise when you want to use Ada. There are very fast communication links between processors and MPI libraries are especially optimized for hardware. You should have Ada libraries for the hardware and I haven't seen any such library even mentioned. Another problem is infrastructure. You need libraries, you need running host processes at every node (for starting your subtasks), you need task spooling system etc. You need proper configuration. But everything is suited for MPI. For example: When you put your parallel task on queue, the spooler automatically starts subtasks on some nodes as they become available, establishes MPI communication between them and assigns MPI number to every subtask. Now, how to begin computations using Ada built-in parallelism?

I don't even know how to establish Ada-style communication between them. And all the problems vanish when you use simple C binding for MPI library...

From: jtg <jtg77@poczta.onet.pl>
Date: Fri, 24 Jun 2005 00:14:42 +0200
Subject: Re: Ada-friendly MPI
Newsgroups: comp.lang.ada,comp.parallel.mpi

> Have you done some testing ? I know some people using the Ada distributed annex on some embedded platform with something like hundred of nodes for HPC (neutronics) without problem.

No performance testing. I had even no idea how to run it.

> What do you call a parallel task ? A process a thread ?

Errr... I should have said: job.

> Are you talking about an executable (process) and using some batch language LSF/PBM ? If so I don't see the difference with Ada.

LSF/PBM

> I'm not talking about parallelism which is more like OpenMP (shared memory) than MPI (distributed memory). I'm talking about Ada distributed annex.

OK... If you write your program for MPI, you must run it with "mpirun" command in mpi-enabled environment. If you write your program for PVM, you must run it with "pvmrun" command in pvm-enabled environment. If you use Ada Distributed Annex, there must be a command that dispatches tasks and establishes communication between them, and the environment itself must be configured for Ada Distribution Annex. On 15-processor cluster I could pass a parameter, either MPI or PVM. Funny thing: PVM was available but not properly configured so you had to use MPI.

On 128-processor cluster all jobs were by default MPI tasks and I could not find a way to change it. But even if I did, I'm pretty sure the system was not configured for Ada Distributed Annex.

> A program is a set of partitions (executable), each partition will communicate with others using the Ada built-in Partition Communication Subsystem.

That's automatic only if you run it in the proper way and the environment is configured. For instance if you run MPI task with "pvmrun" or vice versa nothing will be done automatic.

How do you run Ada distributed programs? How are they dispatched to other nodes (processors/computers)? Is there a command like "adarun" (analogous to mpirun and pvmrun)?

> Then use the same simple binding for Ada. If the binding is simple you should be able to create the Ada counterpart without problem. Which OS are you using ?

Now I use Windows/Debian on my personal computer, both with Ada installed.

Previously:

15-node cluster - AIX

128-node cluster - AFAIR TurboLinux (no Ada installed, I had to install gnat in my home directory)

Now they have Debian on it.

From: Pascal Obry <pascal@obry.net>
Date: 24 Jun 2005 08:39:20 +0200
Subject: Re: Ada-friendly MPI
Newsgroups: comp.lang.ada,comp.parallel.mpi

Job dispatching on the different nodes are handled by GLADE (the implementation of Annex-E for GNAT). In fact, gnatdist (the equivalent to gnatmake for building distributed program) can generate a script or Ada main that will run/dispatch the different job/partitions on the different nodes.

Then the boot-partition (a module in one or more partition) is used to initialized the distributed program. As you see MPI is far more low-level than the Ada solution. And the good news is that you can run your Ada program in a single computer (built with gnatmake) or in a distributed environment (built with gnatdist) without changing a single line of Ada code.

Looks like magic, but it is not. It is in my opinion the right level of abstraction to concentrate on your problem instead of low-level stuff.

[...] GNU/Linux Debian is well supported by GNAT. GNAT is maybe already on your system as it is installed along with the other GCC's compilers.

Differences between Ada 83 and Ada 95

From: Chris Albertson <chrisalbertson90278@yahoo.com>
Date: Fri, 3 Jun 2005 13:32:47 -0700 (PDT)
Subject: Re: Ada compilers/ difference between 83 and 95
Newsgroups: comp.lang.ada

[...] I'm curious, will Ada 83 programs not compile under Ada 95?

If there is a problem where is it? What part of the language spec changed in an incompatible way? Yes I do have some old code that currently runs on a VAX and I'm like to get it running on a Duel Xeon system under Solaris 10. I assumed only minor work would be required. I think gnat has a switch to disallow the '95 syntax.

I would not want to use an old compiler, so much work has been done in recent years it would be a shame not to take advantage of it.

From: Jeff C <jcreem@yahoo.com>
Subject: Re: Ada compilers/ difference between 83 and 95 - The answer
Date: Fri, 03 Jun 2005 21:44:29 -0400
Newsgroups: comp.lang.ada

Chris Albertson wrote:

> Sorry about the last message had no text. Here it is again.

> I'm curious, will Ada 83 programs not compile under Ada 95?

Some programs or packages from Ada 83 will indeed not compile but again it is generally no more of a problem than switching compiler vendors.

See this link:

http://www.adaic.org/learn/tech/8395comp.html

Summary. Along with my experience

a) A few new reserved words. (Never ran into this one)
b) New form required for indefinite generic parameters (I hit this once)
c) Packages can not have a body if their spec does not require it (I hit this once. It was always a bad idea by the way_
d) Character now has 256 items instead of 128 (never hurt by this one)
e) Numeric_Error no longer unique. Now renames of constraint error (Never hurt by this one)

That is all the site listed. I believe I ran into an additional problem with an attribute that was renamed/removed on floats ('small?) which is not listed on that link.

Other issues you will run into (that will happen regardless of the Ada 83 v.s. Ada 95 thing)

1) Use of vendor supplied packages - Can be a big deal.
2) Use of vendor specific attributes or pragmas - can be a moderate deal
3) Different level of support for rep-specs - can be a big deal
4) Different approach for supporting package Machine_Code insertions
5) Code that relies on buggy behavior of the old compiler.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Sat, 04 Jun 2005 05:34:25 GMT
Subject: Re: Ada compilers/ difference between 83 and 95
Newsgroups: comp.lang.ada

I've compiled lots of Ada-83 code with Ada-95 compilers and never had a serious problem. The important thing is that the code was designed to be portable in the first place. If you have lots of compiler or platform dependencies in your code, it becomes harder.

From: Keith Thompson <kst-u@mib.org>
Date: Sat, 04 Jun 2005 00:10:21 GMT
Subject: Re: Ada compilers/ difference between 83 and 95
Newsgroups: comp.lang.ada

The biggest change is the new reserved words that Ada 83 programs might use as identifiers. (That's actually not a very common problem, as far as I know).

The best way to find out is to try compiling the code with an Ada 95 compiler and read the error messages.

From: Marin David Condic <mcondic@acm.org>
Date: Mon, 06 Jun 2005 12:24:56 GMT
Subject: Re: Ada compilers/ difference between 83 and 95
Newsgroups: comp.lang.ada

I've ported a LOT of VAX/VMS (DEC) Ada 83 to Gnat Ada 95 and never encountered anything that was a problem between the two language versions. There

may be some obscure corner-case rules that trigger incompatibilities but in practical use, I just never encountered any. I did get some warnings about using the package ASCII and maybe that's as close to an incompatibility as I've seen.

Mostly, I had some issues where different compiler writers had different views of what to implement. A few instances where DEC Ada handled unconstrained types differently from Gnat Ada and, of course, some vendor specific implementation details once in a while (like packages that may or may not be there because they are vendor supplied) but two observations: I don't recall anything that got past the compiler that ended up being an issue - so the compiler is your friend here. I don't recall anything that didn't run just fine when it finally linked. I *have* had problems when porting between two *different* platforms - mostly with byte-sex issues, but that has been minimal. (Ada 95 has some features to help you out making byte-sex issues portable as well, but I have not delved into them lately)

Porting even a large body of Ada code (if reasonably well written to be portable - no fair throwing in compiler-specific things in every unit & expecting zero effort) even across platforms has not been much of an issue in my experience. I've done a few hundred thousand lines in a couple of days & had it up and running - but it was *my* code so I was familiar with it and I have usually had the wisdom to isolate any compiler specifics. Even a stranger's code being moved across platforms and to a different language standard ought to be minimal fuss.

From: Pascal Obry <pascal@obry.net>
Date: 04 Jun 2005 10:38:09 +0200
Subject: Re: Ada compilers/ difference between 83 and 95 - The answer
Newsgroups: comp.lang.ada

[...] So as you see going from Ada 83 to Ada 95 is not that difficult. This was the design decision when designing Ada 95: It must be upward compatible. And in practice it is.

Compiler Binary Compatibility

From: Harald Korneliussen <vintermann@gmail.com>
Subject: Recompiling?
Date: 29 Aug 2005 07:00:15 -0700
Newsgroups: comp.lang.ada

I have played around with Ada for some time on Linux, but it seems every time I upgrade the compiler I have to recompile (or possibly re-link?) my programs.

It's a bit worrying that all my binaries stop working every time. Anyone know why this happens, and if anything can be done about it?

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 29 Aug 2005 08:26:56 -0700
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

Yes, every major release of GNAT, historically, breaks binary compatibility with previous releases. If your binary is linked with one version of libgnat, upgrading libgnat breaks your program.

The issue is the same with C++ compilers, BTW. For example, g++-3.3 and g++-3.4 are incompatible with each other.

The answer to this is to have a policy about when to switch compilers; one version of the compiler must be designated as "the system compiler" and used by all software that must be deployed together.

One example of such a policy for Ada and GNAT can be found here:

http://www.ada-france.org/debian/debian-ada-policy.html

Unfortunately, it looks like no other GNU/Linux distribution has a policy for Ada (they normally have one for C++).

From: Frank Piron <frank@konad.net>
Date: Tue, 30 Aug 2005 08:12:17 +0200
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

To get rid of the dependencies on libgnat you can build a static version libgnat.a and then link your program statically. I did so on Solaris 2.8.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 30 Aug 2005 05:58:31 -0700
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

> Is the [Debian Ada Policy] page correct in saying: "ASIS, Glade and Florist are not currently available for GCC 3.4" or is this now out of date?

The page is still correct, unfortunately. I would have updated it if events had warranted it.

> OK. I know this question gets asked a lot, but things seem to be in continual flux:

No, it is not; but you are correct that the question is being asked a lot. Wishful thinking alone does not change the answer, however, and only creates this illusion of "flux".

> Which free GNAT version would be best for meeting these criteria:
> 1) Linux, x86 architecture
> 2) with GLADE, PolyORB, ASIS, Florist
> 3) stable and robust for serious use
> The answer seems to be to use GNAT 3.4, but you have to compile everything yourself :(

No, the answer is GNAT 3.15p, and as a consequence you don't have to compile anything :) well, nobody has yet volunteered to package PolyORB for Debian, or any other distribution.

> What if I want x86_64 LP64 mode? Is this straightforward?

Then you are out of luck. With GCC 3.4 or 4.0, you can have x86_64 LP64 mode but not ASIS or GLADE.

> I'd much rather not spend the time and effort trying to build a complete tool chain and libraries :(Are we going to see a suite of Debian packages in the near future? I wish I had time to help out!

Sarge (the current Debian stable distribution) contains a full suite of Ada packages, united together by the Debian Policy for Ada. To my knowledge there are 48 binary packages produced from 22 sources. In addition, the packages gnat-3.3 and gnat-3.4 are provided for experimental use.

You can help out in several ways:

1) Try to persuade AdaCore to release a new "p" version of GNAT, based on a recent GCC, with ASIS and GLADE. This would trigger a change in compilers in Debian (see section 2.8 of the Debian Policy for Ada).
2) Package more software for Debian.

3) Experiment with Martin Krischik's ASIS for GCC 4.0 and AdaCore's CVS repository for GLADE, and package them for Debian unstable. Note that neither of these two projects have announced a "stable, robust for serious use" release, so this would be experimental. But Debian "unstable" is appropriate for this. Also, PolyORB might replace GLADE; you'd want to coordinate with the developers of PolyORB.

Note that etch has switched its default C and C++ compilers to GCC 4.0; all C++ libraries are being recompiled with g++-4.0. Also, gnat-4.0 is already in etch. If and when ASIS and GLADE are provided for it, I will declare it the new default Ada compiler as well, and recompile all Ada packages with it, even on the newly supported architecture, amd64.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 30 Aug 2005 06:09:03 -0700
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

In case you don't know, "Etch" is the code-name for the next release of Debian. Etch is currently in testing. There are several major changes scheduled to take place before Etch becomes stable. One such change is the transition from gcc-3.3 to gcc-4.0 as the system C compiler, and g++-3.3 to g++-4.0 as the system C++ compiler. Another transition is from glibc 2.3.2 to 2.3.5, and a third is from XFree86 4.3 to X.org 6.8. The GCC transition is in progress now, the other two will take place later. Other scheduled transitions

include KDE (3.3 to 3.5) and GNOME (2.8 to 2.12). So, Etch is currently very much in a state of flux :)

As always with Debian, Etch will be released "when it is ready".

From: Alex R. Mosteo <alejandro@mosteo.com>
Date: Tue, 30 Aug 2005 15:23:06 +0200
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

I think [that a release of a new "p" version of GNAT] may be imminent, being really a repackaging of the GAP releases. I remember that Robert Dewar said in some other list that there's really no problem in people outside academic circles getting the GAP release. I've been looking for the exact post but I don't remember the mailing list and have not found it. If someone can point to it this would become clear and not a remembrance of mine (which may be imperfect :)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 1 Sep 2005 04:39:38 -0700
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

> > Are you trying to say that ASIS and GLADE work well with GCC 3.4.2?

> At least ASIS (gnat-asis.sourceforge.net) works well with gcc-3.4.x.

> > Any supporting evidence?

> I successfully used gnatelim, which is part of ASIS. AdaBrowse is a more complex ASIS application and works well, too.

Thanks. I'll change the system Ada compiler for Etch, then. It seems that gnat-3.4 is the best candidate, but unfortunately it would be at odds with the system C and C++ compilers, meaning that it would not be possible to mix Ada and C++ in the same program. If I choose gnat-4.0 as the system Ada compiler, C++ works but ASIS breaks, right?

Thoughts?

Note that I will start the transition only after Etch completes its glibc, g++, X.org, GNOME and KDE transitions, because of [1]. Note also that the GNAT transition will take time as all packages will need to be recompiled.

[1] http://lists.debian.org/debian-devel-announce/2005/08/msg00014.html

From: Georg Bauhaus <bauhaus@futureapps.de>
Date: Thu, 01 Sep 2005 14:37:48 +0200
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

Given that the year 200X is coming closer, I wonder whether the typical Debian Ada user (to be defined) will be more interested in ISO Ada tools, possibly including GtkAda, PolyORB and AWS (thinking of the SOA boom), than in ISO ASIS based add-ons? If Ada is used in a commercial setting, using Debian as a development platform, then the vendor/support will have a word with the customers as to whether to use Ada 200X in preference, I guess, thus GCC 4.x. Is this the case?

A# (hence Ada for Mono, adding value for the commercial Ada programmer), "Now compiles with GNAT GAP (Academic Edition) 1.0.0 (uses some Ada 2005 features)", that statement being made about the October 2004 version of A#. The latest release is from June this year. I take this to be a trend indicator towards GCC 4.

GNU Ada Environment Specification Support

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 31 Aug 2005 04:00:55 -0700
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

Harald Korneliussen wrote:

> I see. On a side note, it's not the only thing I have to fix when reinstalling. The projects I've made with autoconf/automake now refuse to work. Pretty sweet for a system designed to maintain compatibility, huh?

Yes, the autotools routinely break compatibility with previous releases of themselves. Isn't it ironic? Makefiles were invented to circumvent the deficiencies of C (namely the lack of separate compilation); the autotools were invented to circumvent the deficiencies of make (namely the lack of portability), and now the autotools are revealing their own deficiencies.

> There's another pretty basic question I want to ask, about the correct and portable way to install something like the Booch components as a shared library. The package is just a bunch of source files. I suppose I could copy them to every project that I use booch on, but that seems so unclean and wasteful.
> Any suggestions?

Package the Booch components for your distribution. Follow the GNU Ada Environment Specification [1]. Provide a GNAT project file, as explained in detail in [2].

[1] http://cert.uni-stuttgart.de/projects/ada/gnae.php

[2] http://www.ada-france.org/debian/debian-ada-policy.html

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: 1 Sep 2005 00:41:57 -0700
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

Simon Wright wrote:

> Looked at your [1]; it looks very "official" but it gives no context that I can see. What organisation is proposing it? what universe does it fit into? is it only intended for Debian? what about Windows? MacOS? what do AdaCore think? FSF? It refers to a GNAT Filesystem Hierarchy Standard proposal by Jürgen Pfeifer but gives no link? (perhaps this document _is_ the FHS?)

GNAE is primarily written for GNU/Linux, *BSD, and Unix platforms. The FHS (Filesystem Hierarchy Standard)[3] is a unifying standard for all GNU/Linux distributions. The organisation behind GNAE is Florian Weimer, who is perhaps listening. He took ideas from Jürgen Pfeifer, who led the Ada for Linux team [4] for a while.

[3] http://www.pathname.com/fhs/

[4] http://www.gnuada.org/alt.html

Debian now implements part of the GNAE. It implements the filesystem but not the "adainstall" and "adaconfig" commands, which are replaced with "apt-get" and GNAT project files, respectively. In fact, I would support a change in GNAE to mandate project files and drop "adaconfig".

I see that Martin Krischik offers RPM packages of GNADE and other things for SuSE and Red Hat, but I don't know if these packages comply with the GNAE.

I'm not aware of a MacOS X distribution of Ada packages, but if there is one, I hope it follows the GNAE too.

The only Ada distribution I know for Windows is AIDE, but it does not seem to comply with GNAE (but I haven't looked at 1.03).

I don't know what AdaCore or the FSF think of GNAE. Florian?

From: Stephane Riviere <stephane@rochebrune.org>
Date: Fri, 02 Sep 2005 10:02:59 +0200
Subject: Re: Recompiling?
Newsgroups: comp.lang.ada

> The only Ada distribution I know for Windows is AIDE, but it does not seem to comply with GNAE (but I haven't looked at 1.03).

The next (2.0) AIDE release could be GNAE compliant, as this new release is planned to be multi-platform (Windows and Debian GNU/Linux).

From the Windows side, the Unix root directory /usr could be changed for a root install directory chosen by the user... Everything behind could strictly follow GNAE recommendations.

[See also "Ada and the GNU Build System" in AUJ 25-4 (Dec 2004), p.203-204. --su]

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.

2005

October 02-05 — 25th IFIP WG 6.1 **International Conference on Formal Techniques for Networked and Distributed Systems** (FORTE'2005), Taipei, Taiwan. Co-located with ATVA'2005. Topics include: formal description techniques, embedded systems, tool supports, case studies on industrial projects, etc.

October 02-07 — 8th **International Conference on Model Driven Engineering Languages and Systems** (MoDELS'2005), Montego Bay, Jamaica. Formerly the UML series of conferences. Topics include: Model-driven development methodologies, approaches, and languages; Empirical studies of modeling and model-driven development; Tool support for any aspect of model-driven development or model use; Semantics of modeling languages; etc.

October 03-07 — 19th **Brazilian Symposium on Software Engineering** (SBES'2005), Uberlândia, Brazil. Topics include: Distributed Software Engineering; Generative Software Development; Multi-paradigm and Multi-language Modelling and Programming; Object-oriented Techniques; Software Engineering for Embedded and Real-time Software; Software Engineering Tools and Environments; Software Maintenance; Software Quality; Software Reuse; Software Safety and Reliability; Software Security; Software Verification, Validation and Inspection; etc.

October 04-07 — 3rd **International Symposium on Automated Technology for Verification and Analysis** (ATVA'2005), Taipei, Taiwan. Co-located with FORTE'2005.

October 13 — **FNRS Contact Group Meeting** on *The Theory and Practice of Software Verification*, Liège, Belgium. Topics include: model checking, abstraction methods, static analysis, verification tools, and modeling and specification formalisms; Applications and case studies; etc.

October 13-14 — **Workshop "Zuverlässigkeit in eingebetteten Systemen"**, Aachen, Germany. Organized by Gesellshaft für Informatik e.V. Fachgruppe "Ada", and Gesellshaft Mess- und Automatisierungstechnik Fachausschus 5.11 "Embedded Software". Topics include (in German): Programmiersprache Ada und Profile (Raven, SPARK), angeladene Hauptvorträge von Tucker Taft und Erhard Plödereder über den kommenden Standard Ada 2005, etc.

☺ October 13-14 — 3rd **Workshop on Object-oriented Modeling of Embedded Real-Time Systems** (OMER-3), Paderborn, Germany. Topics include: Architectures/frameworks for platform independent, reusable software components; Code-generation; Component interoperability; Formal verification at the model and code level; Software components as products; Software quality; Standards and guidelines; Respective trends in automotive software development; etc.

☺ October 16-20 — 20th Annual **ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2005), San Diego, California, USA. Sponsored by ACM SIGPLAN in cooperation with SIGSOFT.

- ☺ October 16 **Workshop on Synchronization and Concurrency in Object-Oriented Languages** (SCOOL'2005). Topics include: Compiler transformations, Concurrent data structure implementations, Expression of concurrency-related design intent, Languages and semantics, Memory models for concurrent object-oriented languages, Synchronization abstractions, etc.

☺ October 17-18 — **Architecture Analysis & Design Language (AADL) Workshop**, Massy, near Paris, France.

☺ October 18 — 2nd **International Workshop on Software Engineering Course Projects** (SECP'2005), Toronto, Canada.

October 19-21 — 17th **Nordic Workshop on Programming Theory** (NWPT'2005), Copenhagen, Denmark. Topics include: Program verification, Formal specification of programs, Real-Time and hybrid systems, Modeling of concurrency, Programming methods, Tools for program construction and verification, etc.

October 25-26 — **International Conference on Software Testing** (ICSTEST-E'2005), Bilbao, Spain. Topics include: Transportation and Safety-Critical Systems, Industry real experiences, Verification and Validation, Techniques for real time systems, Static and Dynamic analysis, Norms and standards, etc.

October 26-28 — 20th **International Symposium on Computer and Information Sciences** (ISCIS'2005), Istanbul, Turkey. Topics include: Parallel and Distributed Computing, Programming Languages and Algorithms, Software Engineering, etc.

October 27-28 — 6th **International Workshop on Advanced Parallel Processing Technologies** (APPT'2005), Hong Kong, China. Topics include: Middleware, Software Tools and Environments, Parallelizing Compilers, Software Engineering issues, Task Scheduling and Load Balancing, Fault tolerance and dependability, etc.

Oct. 30 – Nov. 03 — 24th **Digital Avionics Systems Conference** (DASC'2005), Washington D.C., USA. Theme: "Avionics in a Changing Market Place - Safe and Secure?" Topics include: Software Engineering: Development of large-scale, flight-critical software systems, including processes and formal methods for design, testing and certification; Lean Avionics: Application of continuous improvement principles/practices (lean, six sigma, TQM, CMM, CMMI) to the design, development and sustainment of mission critical avionics systems; Flight Critical Systems: Methods, techniques, and tools for the design, verification, integration, validation, and certification of complex and highly integrated flight critical systems; etc.

☺ Oct. 31 – Nov. 04 — 7th **International Symposium on Distributed Objects and Applications** (DOA'2005), Agia Napa, Cyprus. Topics include: Application case studies of distribution technologies; Design patterns for distributed systems; Distribution technologies for embedded systems; Interoperability between object systems and complementary technologies; Real-time solutions for distributed objects; Scalability for distributed objects and object middleware; Security for distributed object systems; Specification and enforcement of Quality of Service; Technologies for reliability and fault-tolerance; etc.

November 01-04 — 4th **International Symposium on Formal Methods for Components and Objects** (FMCO'2005), Amsterdam, the Netherlands.

November 01-04 — 7th **International Conference on Formal Engineering Methods** (ICFEM'2005), Manchester, UK. Topics include: all aspects of formal engineering methods, from theoretical work that promises various benefits, to application to real production systems.

October 31 — 3rd **International Workshop on Software Verification and Validation** (SVV'2005). Topics include: Tools, environments and case studies for large scale software verification; Techniques to validate system software (such as compilers) as well as assembly code/Java bytecode; Proof techniques for verifying specific classes of software (such as object-oriented programs); Integration of formal verification into software development projects; etc.

☺ November 02-05 — 3rd **International Symposium on Parallel and Distributed Processing and Applications** (ISPA'2005), Nanjing, China. Topics include: Parallel/distributed system architectures; Tools and environments for software development; Parallel/distributed algorithms; Parallel compilers; Parallel programming languages; Distributed systems; Reliability, fault-tolerance, and security; Parallel/distributed applications; etc.

November 08-11 — 16th **IEEE International Symposium on Software Reliability Engineering** (ISSRE'2005), Chicago, Illinois, USA. Theme: "Developing High Reliability for Ubiquitous Mobile Applications". Topics include: Software safety analysis, Formal reliability assurance methods, Software testing and verification, Empirical reliability studies, Reliability measurement, Tools and automation, Fault-tolerant and robust software, Security testing, Software certification, End-to-end dependability, etc.

November 08-11 — 12th **Working Conference on Reverse Engineering** (WCRE'2005), Pittsburgh, PA, USA. Theme: "Recovering and Reclaiming Architecture". Topics include: Software architecture recovery; Program

transformation and refactoring; Object and aspect identification; Preprocessing, parsing and fact extraction; Reverse engineering tool support; Program slicing; Redocumenting legacy systems; Program analysis; Reengineering patterns; etc.

November 09-11 **European Software Process Improvement and Innovation Conference** (EuroSPI'2005), Budapest, Hungary.

♦ Nov. 13-17 2005 **ACM SIGAda Annual International Conference** (SIGAda'2005), Atlanta, Georgia, USA. Sponsored by ACM SIGAda; in cooperation with SIGAPP, SIGCAS, SIGCSE, SIGPLAN, SIGSOFT, and Ada-Europe. Topics include: safety and high integrity issues, real-time and embedded applications, Ada & software engineering education, Ada in other environments such as XML and .NET, Ada and other languages, metrics, standards, analysis, testing, validation, and quality assurance, etc.

November 17-18 **XP Day Benelux** 2005, Rotterdam, The Netherlands.

Nov. 28 – Dec. 02 **ACM/IFIP/USENIX International Middleware Conference** (Middleware'2005), Grenoble, France.

Nov. 29 – Dec. 01 18th **International Conference on Software & Systems Engineering and their Applications** (ICSSEA'2005), Paris, France.

Nov. 29 – Dec. 02 5th **International Conference on Integrated Formal Methods** (IFM'2005), Eindhoven, The Netherlands. Deadline for submissions: October 17 (doctoral symposium).

☺ Dec. 05-08 6th **International Conference on Parallel and Distributed Computing, Applications, and Techniques** (PDCAT'2005), Dalian, China. Topics include: Formal methods and programming Languages, Parallelizing compilers, Component-based and OO Technology, Tools and environments for software development, etc.

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

December 12-14 9th **International Conference on Principles of Distributed Systems** (OPODIS'2005), Pisa, Italy. Topics include: communication and synchronization protocols; embedded systems; fault-tolerance, reliability, availability; real-time systems; security issues in distributed computing and systems; etc.

December 12-14 11th **International Symposium Pacific Rim Dependable Computing** (PRDC'2005), Changsha, Hunan, China. Topics include: Software and hardware reliability, testing, verification and validation; Dependability measurement, modeling and evaluation; Safety-critical systems and software; Tools for design and evaluation of dependable systems; Dependability issues in distributed and parallel systems; Dependability issues in real-time systems; etc.

December 15-17 12th **Asia-Pacific Software Engineering Conference** (APSEC'2005), Taipei, Taiwan. Topics include: Software Formal Methods, Software Process Improvement, Cost Estimation, Risk Management, Quality Management, Object-Oriented Technology, etc.

December 18-21 12th **IEEE International Conference on High Performance Computing** (HiPC'2005), Goa, India. Topics include: Scientific/Engineering Applications, System Design for High Reliability, Parallel and Distributed Computing, Heterogeneous Computing, Embedded Applications and Systems, Parallel Languages and Programming Environments, Load Balancing and Scheduling, etc.

2006

January 04-07 *Software Technology Track* of the 39th **Hawaii International Conference on System Sciences** (HICSS-39), Kauai, Haway, USA. Includes mini-tracks on: Strategic Software Engineering; Adaptive and Evolvable Software Systems; etc.

January 09-10 **ACM SIGPLAN 2006 Symposium on Partial Evaluation and Program Manipulation** (PEPM'2006), Charleston, South Carolina, USA. Deadline for submissions: October 7, 2005.

January 11-13 33rd **Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages** (POPL'2006), Charleston, South Carolina, USA. Topics include: fundamental principles and

important innovations in the design, definition, analysis, transformation, implementation and verification of programming languages, programming systems, and programming abstractions.

January 14 — 2006 **International Workshop on Foundations and Developments of Object-Oriented Languages** (FOOL/WOOD'2006), Charleston, South Carolina, USA. Following POPL'2006. Topics include: language semantics, type systems, program analysis and verification, concurrent and distributed languages, language-based security, etc.

February 13-17 — 5th **International Conference on COTS-Based Software Systems** (ICCBSS'2006), Orlando, Florida, USA. Theme: "Pushing the COTS Envelope".

March 06-10 — 3rd **International Conference on High Performance Scientific Computing** (HPSC'2006), Hanoi, Vietnam. Topics include: parallel computing (architectures, tools, environments, ...), software development, etc.

☺ March 13-15 — **International Symposium on Secure Software Engineering** (ISSSE'2006), McLean, VA, USA (near Washington, DC). Topics include: Formal specification, designs, policies, and proofs; Coding practices; Static analysis and other automated support; Processes for producing secure software; Certification and accreditation; Relationships among software correctness, reliability, safety, and security; Lessons learned; Technology transfer; etc.

March 20-24 — 5th **International Conference on Aspect-Oriented Software Development** (AOSD'2006), Bonn, Germany. Deadline for submissions: October 14, 2005 (workshops, tutorials, demos, exhibits, panels), January 23, 2006 (student extravaganza).

March 25-26 — ETAPS2006 - 5th **Workshop on Software Composition** (SC'2006), Vienna, Austria. Deadline for submissions: December 2, 2005.

March 27-29 — ETAPS2006 - 9th **International Conference on Fundamental Approaches to Software Engineering** (FASE'2006), Vienna, Austria. Topics include: Implementation concepts and technologies (distributed and embedded applications), Software evolution (refactoring, reverse and re-engineering, etc.), Software quality (validation and verification of software using theorem proving, testing, analysis, metrics, etc.), Application of formal methods to software development, etc. Deadline for submissions: October 7, 2005 (abstracts), October 14, 2005 (full papers).

☺ April 04-07 — 12th **IEEE Real-Time and Embedded Technology and Applications Symposium** (RTAS'2006), San Jose, CA. 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. Deadline for submissions: October 7, 2005 (full papers), January 16, 2006 (work in progress papers).

☺ 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. Deadline for submissions: January 9, 2006.

☺ 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. Deadline for submissions: October 8, 2005 (abstracts), October 15, (papers).

April 19 — 19th **Conference on Software Engineering Education and Training** (CSEET'2006) Oahu, Hawaii, USA. Deadline for submissions: October 15, 2005 (research and experience papers, tutorials, panels, and educational materials).

☺ 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. Deadline for submissions: October 13, 2005 (position papers, papers, panels).

April 23-27 — 21st **ACM Symposium on Applied Computing** (SAC'2006), Dijon, France. Includes tracks on: Software Engineering, etc.

☺ April 23-27 **Track on Programming Languages** (PL'2005). 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 25-29 20th **IEEE International Parallel and Distributed Processing Symposium** (IPDPS'2005), 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. Deadline for submissions: October 7, 2005.

☺ 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. Deadline for submissions: November 8, 2005.

May 01-04 **Systems and Software Technology Conference** (SSTC'2006), Salt Lake City, Utah, USA.

☺ May 20-28 28th **International Conference on Software Engineering** (ICSE'2006), Shanghai, China. Deadline for submissions: October 6, 2005 (workshops, tutorials), October 30, 2005 (experience, Far East experience, SE: achievements/challenges, demonstrations, emerging results), December 5, 2005 (doctoral symposium).

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. Deadline for submissions: October 10, 2005 (abstracts), December 1, 2005 (full papers).

May 28-31 6th **International Conference on Computational Science** (ICCS'2006), Reading, UK. Deadline for submissions: November 1, 2005 (workshops), December 2, 2005 (full papers).

♦ June 05-09 11th **International Conference on Reliable Software Technologies - Ada-Europe'2006**, Porto, Portugal. Sponsored by Ada-Europe, in cooperation with ACM SIGAda (approval pending). Deadline for submissions: October 30, 2005 (papers, tutorials, workshops).

June 26-28 11th Annual **Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2006), Bologna, Italy.

☺ July 03-07 20th **European Conference on Object-Oriented Programming** (ECOOP'2006), Nantes, France.

☺ 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. Deadline for paper submissions: February 1, 2006.

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

2007

☺ 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).

Call for Papers

11th International Conference on Reliable Software Technologies – Ada-Europe 2006

5-9 June 2006, Porto, Portugal

http://www.ada-europe.org/conference2006.html

Conference Chair

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

Program Co-Chairs

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

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

Tutorial Chair

Jorge Real
U. P. Valencia, Spain
jorge@disca.upv.es

Exhibition Chair

José Ruiz
AdaCore, France
ruiz@adacore.com

Publicity Chair

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

Local Chair

Sandra Almeida
Polytechnic Institute of Porto, Portugal
salmeida@dei.isep.ipp.pt

Ada-Europe Conference Liaison

Laurent Pautet
Telecom Paris, France
pautet@enst.fr

In cooperation with

SIGAda
(approval pending)

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.

Schedule

30 October 2005	Submission of papers, workshop/tutorial proposals
20 January 2006	Notification to authors
20 February 2006	Camera-ready papers required
5-9 June 2006	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, 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

Program Committee

(*preliminary*)

Alonso Alejandro, Universidad Politécnica de Madrid, Spain
Asplund Lars, Mälardalens Högskola, Sweden
Barnes Janet, Praxis Critical Systems, UK
Bernat Guillem, University of York, UK
Blieberger Johann, Technische Universität Wien, Austria
Brosgol Ben, AdaCore, USA
Burgstaller Bernd, University of Sidney, Australia
Burns Alan, University of York, UK
Cederling Ulf, Vaxjo University, Sweden
Craeynest Dirk, Aubay Belgium & K.U.Leuven, Belgium
Crespo Alfons, Universidad Politécnica de Valencia, Spain
Devillers Raymond, Université Libre de Bruxelles, Belgium
González Harbour Michael, Universidad de Cantabria, Spain
Gutiérrez José Javier, Universidad de Cantabria, Spain
Hately Andrew, Eurocontrol, Hungary
Hommel Günter, Technischen Univesität Berlin, Germany
Kauer Stefan, EADS Dornier, Germany
Keller Hubert, Institut für Angewandte Informatik, Germany
Kermarrec Yvon, ENST Bretagne, France
Kienzle Jörg, McGill University, Canada
Kordon Fabrice, Université Pierre & Marie Curie, France
LLamosi Albert, Universitat de les Illes Balears, Spain
Mazzanti Franco, Istituto di Scienza e Tecnologie dell'Informazione, Italy
McCormick John, University of Northern Iowa, USA
Michell Stephen, Maurya Software, Canada
Miranda Javier, Universidad Las Palmas de Gran Canaria, Spain
Pautet Laurent, Telecom Paris, France
Pinho Luís Miguel, Polytechnic Institute of Porto, Portugal
Plödereder Erhard, Universität Stuttgart, Germany
de la Puente Juan A., Universidad Politécnica de Madrid, Spain
Real Jorge, Universidad Politécnica de Valencia, Spain
Romanovsky Alexander, University of Newcastle upon Tyne, UK
Rosen Jean-Pierre, Adalog, France
Ruiz José, AdaCore, France
Schonberg Edmond, New York University & AdaCore, USA
Tokar Joyce, Pyrrhus Software, USA
Vardanega Tullio, Università di Padova, Italy
Wellings Andy, University of York, UK
Winkler Jürgen, Friedrich-Schiller-Universität, Germany

Submissions

Authors are invited to submit original contributions. Paper submissions shall be in English, should be complete and should not exceed 20 double-spaced pages in length. Authors should submit their work via the Web submission system accessible from the conference Home page. The preferred format for submission is PDF. Postscript can also be accepted, as long as it was generated selecting the "optimize for portability" option in the used printer driver. Submissions by other means and formats will not be accepted. If you do not have easy access to the Internet, or you do not have an appropriate Web browser, please contact the Program Co-Chair Luís Miguel Pinho, whose address details are on the flip side of this call as well as on the conference Home page.

Proceedings

The authors of accepted papers shall prepare their camera-ready submissions in full conformance with the LNCS style, not exceeding 12 pages and strictly by February 20, 2006. For format and style guidelines authors should refer to: http://www.springer.de/comp/lncs/authors.html. Failure to comply will prevent the paper from appearing in the conference proceedings. The conference proceedings including all accepted papers will be published in the Lecture Notes in Computer Science (LNCS) series by Springer Verlag, which will be available at the start of the conference.

Awards

Ada-Europe will offer honorary awards for the best paper and the best presentation, which will be presented during the banquet and at the close of the conference respectively.

Call for Tutorials

Tutorials should address subjects that fall within the thrust of the conference and may be proposed as either half- or a 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.

Call for Workshops

Workshops on themes within the conference scope may be arranged to discuss matters of immediate technical interest as well as to foster action on longer-term technical objectives. Proposals may be submitted for half- or full-day workshops, to be scheduled on either ends of the main conference. Workshop proposals should be submitted by e-mail to the Conference Chair Luís Miguel Pinho The workshop organiser shall also commit to preparing proceedings for timely publication in the Ada User Journal.

Exhibition

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.

Reduced Fees for Students

A small number of grants are available for students who will (co-)author and present papers at the conference. A reduction of 25% will be made to the conference fee. Contact the Conference Chair Luís Miguel Pinho for details.

Rationale for Ada 2005: 4 Tasking and Real-Time

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 paper describes various improvements in the tasking and real-time areas for Ada 2005.

There are only a few changes to the core tasking model itself. One major extension, however, is the ability to combine the interface feature described in an earlier paper with the tasking model; this draws together the object-oriented and tasking models of Ada which previously were disjoint aspects of the language.

There are also many additional predefined packages in the Real-Time Systems annex concerning matters such as scheduling and timing; these form the major topic of this paper.

Keywords: rationale, Ada 2005.

1 Overview of Changes

The WG9 guidance document [1] identifies real-time systems as an important area. It says

"The main purpose of the Amendment is to address identified problems in Ada that are interfering with Ada's usage or adoption, especially in its major application areas (such as high-reliability, long-lived real-time and/or embedded applications and very large complex systems). The resulting changes may range from relatively minor, to more substantial."

It then identifies the inclusion of the Ravenscar profile [2] (for predictable real-time) as a worthwhile addition and then asks the ARG to pay particular attention to

> Improvements that will maintain or improve Ada's advantages, especially in those user domains where safety and criticality are prime concerns. Within this area it cites as high priority, improvements in the real-time features and improvements in the high integrity features.

Ada 2005 does indeed make many improvements in the real-time area and includes the Ravenscar profile as specifically mentioned. The following Ada issues cover the relevant changes and are described in detail in this paper:

249 Ravenscar profile for high-integrity systems

265 Partition elaboration policy for high-integrity systems

266 Task termination procedure

297 Timing events

298 Non-preemptive dispatching

305 New pragma and restrictions for real-time systems

307 Execution-time clocks

321 Definition of dispatching policies

327 Dynamic ceiling priorities

345 Protected and task interfaces

347 Title of Annex H

354 Group execution-time budgets

355 Priority dispatching including Round Robin

357 Earliest Deadline First scheduling

386 Further functions returning time-span values

394 Redundant Restrictions identifiers and Ravenscar

397 Conformance and overriding for procedures and entries

399 Single tasks and protected objects with interfaces

421 Sequential activation and attachment

These changes can be grouped as follows.

First there is the introduction of a mechanism for monitoring task termination (266).

A major innovation in the core language is the introduction of synchronized interfaces which provide a high degree of unification between the object-oriented and real-time aspects of Ada (345, 397, 399).

There is of course the introduction of the Ravenscar profile (249) plus associated restrictions (305, 394) in the Real-Time Systems annex (D).

There are major improvement to the scheduling and task dispatching mechanisms with the addition of further standard policies (298, 321, 327, 355, 357). These are also in Annex D.

A number of timing mechanisms are now provided. These concern stand-alone timers, timers for monitoring the CPU time of a single task, and timers for controlling the budgeting of time for groups of tasks (297, 307, 354, 386). Again these are in Annex D.

Finally, more control is provided over partition elaboration which is very relevant to real-time high-integrity systems (265, 421). This is in Annex H which is now entitled High-Integrity Systems (347).

Note that further operations for the manipulation of time in child packages of Calendar (351) will be discussed with the predefined library in a later paper.

2 Task termination

In the Introduction we mentioned the problem of how tasks can have a silent death in Ada 95. This happens if a task raises an exception which is not handled by the task itself. Tasks may also terminate because of going abnormal as well as terminating normally. The detection of task termination and its causes can be monitored in Ada 2005 by the package Ada.Task_Termination whose specification is essentially

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Exceptions; use Ada.Exceptions;
package Ada.Task_Termination is
  pragma Preelaborable(Task_Termination);

  type Cause_Of_Termination is
            (Normal, Abnormal, Unhandled_Exception);

  type Termination_Handler is access protected
      procedure(Cause: in Cause_Of_Termination;
            T: in Task_Id; X: in Exception_Occurrence);

  procedure Set_Dependents_Fallback_Handler
                    (Handler: in Termination_Handler);
  function Current_Task_Fallback_Handler
                          return Termination_Handler;

  procedure Set_Specific_Handler(T: in Task_Id;
                    Handler: in Termination_Handler);
  function Specific_Handler(T: in Task_Id)
                          return Termination_Handler;

end Ada.Task_Termination;
```

(Note that the above includes use clauses in order to simplify the presentation; the actual package does not have use clauses. We will use a similar approach for the other predefined packages described in this paper.)

The general idea is that we can associate a protected procedure with a task. The protected procedure is then invoked when the task terminates with an indication of the reason passed via its parameters. The protected procedure is identified by using the type Termination_Handler which is an access type referring to a protected procedure.

The association can be done in two ways. Thus (as in the Introduction) we might declare a protected object Grim_Reaper

```
protected Grim_Reaper is
  procedure Last_Gasp(C: Cause_Of_Termination;
                T: Task_Id; X: Exception_Occurrence);
end Grim_Reaper;
```

which contains the protected procedure Last_Gasp. Note that the parameters of Last_Gasp match those of the access type Termination_Handler.

We can then nominate Last_Gasp as the protected procedure to be called when the specific task T dies by

```
Set_Specific_Handler(T'Identity,
                    Grim_Reaper.Last_Gasp'Access);
```

Alternatively we can nominate Last_Gasp as the protected procedure to be called when any of the tasks dependent on the current task becomes terminated by writing

```
Set_Dependents_Fallback_Handler
                    (Grim_Reaper.Last_Gasp'Access);
```

Note that a task is not dependent upon itself and so this does not set a handler for the current task.

Thus a task can have two handlers. A fallback handler and a specific handler and either or both of these can be null. When a task terminates (that is after any finalization but just before it vanishes), the specific handler is invoked if it is not null. If the specific handler is null, then the fallback handler is invoked unless it too is null. If both are null then no handler is invoked.

The body of protected procedure Last_Gasp might then output various diagnostic messages

```
procedure Last_Gasp(C: Cause_Of_Termination;
                T: Task_Id; X: Exception_Occurrence) is
begin
  case C is
    when Normal => null;
    when Abnormal =>
      Put("Something nasty happened to task ");
      Put_Line(Image(T));
    when Unhandled_Exception =>
      Put("Unhandled exception occurred in task ");
      Put_Line(Image(T));
      Put(Exception_Information(X));
  end case;
end Last_Gasp;
```

There are three possible reasons for termination, it could be normal, abnormal (caused by abort), or because of propagation of an unhandled exception. In the last case the parameter X gives details of the exception occurrence whereas in the other cases X has the value Null_Occurrence.

Initially both specific and fallback handlers are null for all tasks. However, note that if a fallback handler has been set for all dependent tasks of T then the handler will also apply to any task subsequently created by T or one of its descendants. Thus a task can be born with a fallback handler already in place.

If a new handler is set then it replaces any existing handler of the appropriate kind. Calling either setting procedure with null for the handler naturally sets the appropriate handler to null.

The current handlers can be found by calling the functions Current_Task_Fallback_Handler or Specific_Handler; they return null if the handler is null.

It is important to realise that the fallback handlers for the tasks dependent on T need not all be the same since one of the dependent tasks of T might set a different handler for its own dependent tasks. Thus the fallback handlers for a tree

of tasks can be different in various subtrees. This structure is reflected by the fact that the determination of the current fallback handler of a task is in fact done by searching recursively the tasks on which it depends.

Note that we cannot directly interrogate the fallback handler of a specific task but only that of the current task. Moreover, if a task sets a fallback handler for its dependents and then enquires of its own fallback handler it will not in general get the same answer because it is not one of its own dependents.

It is important to understand the situation regarding the environment task. This unnamed task is the task that elaborates the library units and then calls the main subprogram. Remember that library tasks (that is tasks declared at library level) are activated by the environment task before it calls the main subprogram.

Suppose the main subprogram calls the setting procedures as follows

```
procedure Main is

  protected RIP is
    protected procedure One( ... );
    protected procedure Two( ... );
  end;
  ...
begin
  Set_Dependents_Fallback_Handler(RIP.One'Access);
  Set_Specific_Handler(Current_Task, RIP.Two'Access);
  ...
end Main;
```

The specific handler for the environment task is then set to Two (because Current_Task is the environment task at this point) but the fallback handler for the environment task is null. On the other hand the fallback handler for all other tasks in the program including any library tasks is set to One. Note that it is not possible to set the fallback handler for the environment task.

The astute reader will note that there is actually a race condition here since a library task might have terminated before the handler gets set. We could overcome this by setting the handler as part of the elaboration code thus

```
package Start_Up is
  pragma Elaborate_Body;
end;

with Ada.Task_Termination; use Ada.Task_Termination;
package body Start_Up is
begin
  Set_Dependents_Fallback_Handler(RIP.One'Access);
end Start_Up;

with Start_Up;
pragma Elaborate(Start_Up);
package Library_Tasks is
  ...                          -- declare library tasks here
end;
```

Note how the use of pragmas Elaborate_Body and Elaborate ensures that things get done in the correct order.

Some minor points are that if we try to set the specific handler for a task that has already terminated then Tasking_Error is raised. And if we try to set the specific handler for the null task, that is call Set_Specific_Handler with parameter T equal to Null_Task_Id, then Program_Error is raised. These exceptions are also raised by calls of the function Specific_Handler in similar circumstances.

3 Synchronized interfaces

We now turn to the most important improvement to the core tasking features introduced by Ada 2005. This concerns the coupling of object oriented and real-time features through inheritance.

Recall from the paper on the object oriented model that we can declare an interface thus

```
type Int is interface;
```

An interface is essentially an abstract tagged type that cannot have any components but can have abstract operations and null procedures. We can then derive other interfaces and tagged types by inheritance such as

```
type Another_Int is interface and Int1 and Int2;

type T is new Int1 and Int2;

type TT is new T and Int3 and Int4;
```

Remember that a tagged type can be derived from at most one other normal tagged type but can also be derived from several interfaces. In the list, the first is called the parent (it can be a normal tagged type or an interface) and any others (which can only be interfaces) are called progenitors.

Ada 2005 also introduces further categories of interfaces, namely synchronized, protected, and task interfaces. A synchronized interface can be implemented by either a task or protected type; a protected interface can only be implemented by a protected type and a task interface can only be implemented by a task type.

A nonlimited interface can only be implemented by a nonlimited type. However, an explicitly marked limited interface can be implemented by any tagged type (limited or not) or by a protected or task type. Remember that task and protected types are inherently limited. Note that we use the term limited interface to refer collectively to interfaces marked limited, synchronized, task or protected and we use explicitly limited to refer to those actually marked as limited.

So we can write

```
type LI is limited interface;       -- similarly type LI2

type SI is synchronized interface;

type TI is task interface;

type PI is protected interface;
```

and we can of course provide operations which must be abstract or null. (Remember that **synchronized** is a new reserved word.)

We can compose these interfaces provided that no conflict arises. The following are all permitted:

```
type TI2 is task interface and LI and TI;

type LI3 is limited interface and LI and LI2;

type TI3 is task interface and LI and LI2;

type SI2 is synchronized interface and LI and SI;
```

The rule is simply that we can compose two or more interfaces provided that we do not mix task and protected interfaces and the resulting interface must be not earlier in the hierarchy: limited, synchronized, task/protected than any of the ancestor interfaces.

We can derive a real task type or protected type from one or more of the appropriate interfaces

```
task type TT is new TI with
  ...              -- and here we give entries as usual
end TT;
```

or

```
protected type PT is new LI and SI with
  ...
end PT;
```

Unlike tagged record types we cannot derive a task or protected type from another task or protected type as well. So the derivation hierarchy can only be one level deep once we declare an actual task or protected type.

The operations of these various interfaces are declared in the usual way and an interface composed of several interfaces has the operations of all of them with the same rules regarding duplication and overriding of an abstract operation by a null one and so on as for normal tagged types.

When we declare an actual task or protected type then we must implement all of the operations of the interfaces concerned. This can be done in two ways, either by declaring an entry or protected operation in the specification of the task or protected object or by declaring a distinct subprogram in the same list of declarations (but not both). Of course, if an operation is null then it can be inherited or overridden as usual.

Thus the interface

```
package Pkg is
  type TI is task interface;
  procedure P(X: in TI) is abstract;
  procedure Q(X: in TI; I: in Integer) is null;
end Pkg;
```

could be implemented by

```
package PT1 is
  task type TT1 is new TI with
    entry P;       -- P and Q implemented by entries
    entry Q(I: in Integer);
  end TT1;
end PT1;
```

or by

```
package PT2 is
  task type TT2 is new TI with
    entry P;       -- P implemented by an entry
  end TT2;
                   -- Q implemented by a procedure
  procedure Q(X: in TT2; I: in Integer);
end PT2;
```

or even by

```
package PT3 is
  task type TT3 is new TI with end;
                   -- P implemented by a procedure
                   -- Q inherited as a null procedure
  procedure P(X: in TT3);
end PT3;
```

In this last case there are no entries and so we have the juxtaposition **with end** which is somewhat similar to the juxtaposition **is end** that occurs with generic packages used as signatures.

Observe how the first parameter which denotes the task is omitted if it is implemented by an entry. This echoes the new prefixed notation for calling operations of tagged types in general. Remember that rather than writing

```
Op(X, Y, Z, ...);
```

we can write

```
X.Op(Y, Z, ...);
```

provided certain conditions hold such as that X is of a tagged type and that Op is a primitive operation of that type.

In order for the implementation of an interface operation by an entry of a task type or a protected operation of a protected type to be possible some fairly obvious conditions must be satisfied.

In all cases the first parameter of the interface operation must be of the task type or protected type (it may be an access parameter).

In addition, in the case of a protected type, the first parameter of an operation implemented by a protected procedure or entry must have mode **out** or **in out** (and in the case of an access parameter it must be an access to variable parameter).

If the operation does not fit these rules then it has to be implemented as a subprogram. An important example is that a function has to be implemented as a function in the case of a task type because there is no such thing as a function entry. However, a function can often be directly implemented as a protected function in the case of a protected type.

Entries and protected operations which implement inherited operations may be in the visible part or private part of the task or protected type in the same way as for tagged record types.

It may seem rather odd that an operation can be implemented by a subprogram that is not part of the task or protected type itself – it seems as if it might not be task safe in some way. But a common paradigm is where an operation as an abstraction has to be implemented by two or more entry calls. An example occurs in some implementations of the classic readers and writers problem as we shall see later.

Of course a task or protected type which implements an interface can have additional entries and operations as well just as a derived tagged type can have more operations than its parent.

The overriding indicators **overriding** and **not overriding** can be applied to entries as well as to procedures. Thus the package PT2 above could be written as

```
package PT2 is
  task type TT2 is new TI with
    overriding          -- P implemented by an entry
    entry P;
  end TT2;

  overriding            -- Q implemented by procedure
  procedure Q(X: in TT2; I: in Integer);
end PT2;
```

We will now explore a simple readers and writers example in order to illustrate various points. We start with the following interface

```
package RWP is
  type RW is limited interface;
  procedure Write(Obj: out RW; X: in Item) is abstract;
  procedure Read(Obj: in RW; X: out Item) is abstract;
end RWP;
```

The intention here is that the interface describes the abstraction of providing an encapsulation of a hidden location and a means of writing a value (of some type Item) to it and reading a value from it – very trivial.

We could implement this in a nonsynchronized manner thus

```
type Simple_RW is new RW with
  record
    V: Item;
  end record;

overriding
procedure Write(Obj: out Simple_RW; X: in Item);

overriding
procedure Read(Obj: in Simple_RW; X: out Item);

...

procedure Write(Obj: out Simple_RW; X: in Item) is
begin
  Obj.V := X;
end Write;

procedure Read(Obj: in Simple_RW; X: out Item) is
begin
  X := Obj.V;
end Read;
```

This implementation is of course not task safe (task safe is sometimes referred to as thread-safe). If a task calls Write and the type Item is a composite type and the writing task is interrupted part of the way through writing, then a task which calls Read might get a curious result consisting of part of the new value and part of the old value.

For illustration we could derive a synchronized interface

```
type Sync_RW is synchronized interface and RW;
```

This interface can only be implemented by a task or protected type. For a protected type we might have

```
protected type Prot_RW is new Sync_RW with
  overriding
  procedure Write(X: in Item);
  overriding
  procedure Read(X: out Item);
private
  V: Item;
end;

protected body Prot_RW is
  procedure Write(X: in Item) is
  begin
    V := X;
  end Write;

  procedure Read(X: out Item) is
  begin
    X := V;
  end Read;
end Prot_RW;
```

Again observe how the first parameter of the interface operations is omitted when they are implemented by protected operations.

This implementation is perfectly task safe. However, one of the characteristics of the readers and writers example is that it is quite safe to allow multiple readers since they cannot interfere with each other. But the type Prot_RW does not allow multiple readers because protected procedures can only be executed by one task at a time.

Now consider

```
protected type Multi_Prot_RW is new Sync_RW with
  overriding
  procedure Write(X: in Item);
  not overriding
  function Read return Item;
private
  V: Item;
end;
```

```
   overriding
   procedure Read(Obj: in Multi_Prot_RW; X: out Item);

   ...
   protected body Multi_Prot_RW is
     procedure Write(X: in Item) is
     begin
       V := X;
     end Write;

     function Read return Item is
     begin
       return V;
     end Read;
   end Multi_Prot_RW;

   procedure Read(Obj: in Multi_Prot_RW; X: out Item) is
   begin
     X := Obj.Read;
   end Read;
```

In this implementation the procedure Read is implemented by a procedure outside the protected type and this procedure then calls the function Read within the protected type. This allows multiple readers because one of the characteristics of protected functions is that multiple execution is permitted (but of course calls of the protected procedure Write are locked out while any calls of the protected function are in progress). The structure is emphasized by the use of overriding indicators.

A simple tasking implementation might be as follows

```
   task type Task_RW is new Sync_RW with
     overriding
     entry Write(X: in Item);
     overriding
     entry Read(X: out Item);
   end;

   task body Task_RW is
     V: Item;
   begin
     loop
       select
         accept Write(X: in Item) do
           V := X;
         end Write;
       or
         accept Read(X: out Item) do
           X := V;
         end Read;
       or
         terminate;
       end select;
     end loop;
   end Task_RW;
```

Finally, here is a tasking implementation which allows multiple readers and ensures that an initial value is set by only allowing a call of Write first. It is based on an example in that textbook [3].

```
   task type Multi_Task_RW(V: access Item) is
                                   new Sync_RW with
     overriding
     entry Write(X: in Item);
     not overriding
     entry Start;
     not overriding
     entry Stop;
   end;

   overriding
   procedure Read(Obj: in Multi_Task_RW; X: out Item);

   ...

   task body Multi_Task_RW is
     Readers: Integer := 0;
   begin
     accept Write(X: in Item) do
       V.all := X;
     end Write;
     loop
       select
         when Write'Count = 0 =>
         accept Start;
         Readers := Readers + 1;
       or
         accept Stop;
         Readers := Readers – 1;

       or
         when Readers = 0 =>
         accept Write(X: in Item) do
           V.all := X;
         end Write;
       or
         terminate;
       end select;
     end loop;
   end Multi_Task_RW;

   overriding
   procedure Read(Obj: in Multi_Task_RW; X: out Item) is
   begin
     Obj.Start;
     X := Obj.V.all;
     Obj.Stop;
   end Read;
```

In this case the data being protected is accessed via the access discriminant of the task. It is structured this way so that the procedure Read can read the data directly. Note also that the procedure Read (which is the implementation of the procedure Read of the interface) calls two entries of the task.

It should be observed that this last example is by way of illustration only. As is well known, the Count attribute used in tasks (as opposed to protected objects) can be misleading if tasks are aborted or if entry calls are timed out. Moreover, it would be gruesomely slow.

So we have seen that a limited interface such as RW might be implemented by a normal tagged type (plus its various

operations) and by a protected type and also by a task type. We could then dispatch to the operations of any of these according to the tag of the type concerned. Observe that task and protected types are now other forms of tagged types and so we have to be careful to say tagged record type (or informally, normal tagged type) where appropriate.

In the above example, the types Simple_RW, Prot_RW, Multi_Prot_RW, Task_RW and Multi_Task_RW all implement the interface RW.

So we might have

```
RW_Ptr: access RW'Class := ...

...
RW_Ptr.Write(An_Item);          -- dispatches
```

and according to the value in RW_Ptr this might call the appropriate entry or procedure of an object of any of the types implementing the interface RW.

However if we have

```
Sync_RW_Ptr: access Sync_RW'Class := ...
```

then we know that any implementation of the synchronized interface Sync_RW will be task safe because it can only be implemented by a task or protected type. So the dispatching call

```
Sync_RW_Ptr.Write(An_Item);    -- task safe dispatching
```

will be task safe.

An interesting point is that because a dispatching call might be to an entry or to a procedure we now permit what appear to be procedure calls in timed entry calls if they might dispatch to an entry.

So we could have

```
select
   RW_Ptr.Read(An_Item);          -- dispatches
or
   delay Seconds(10);
end select;
```

Of course it might dispatch to the procedure Read if the type concerned turns out to be Simple_RW in which case a time out could not occur. But if it dispatched to the entry Read of the type Task_RW then it could time out.

On the other hand we are not allowed to use a timed call if it is statically known to be a procedure. So

```
A_Simple_Object: Simple_RW;
...
select
   A_Simple_Object.Read(An_Item);        -- illegal
or
   delay Seconds(10);
end select;
```

is not permitted.

A note of caution is in order. Remember that the time out is to when the call gets accepted. If it dispatches to Multi_Task_RW.Read then time out never happens because the Read itself is a procedure and gets called at once. However, behind the scenes it calls two entries and so could take a long time. But if we called the two entries directly with timed calls then we would get a time out if there were a lethargic writer in progress. So the wrapper distorts the abstraction. In a sense this is not much worse than the problem we have anyway that a time out is to when a call is accepted and not to when it returns – it could hardly be otherwise.

The same rules apply to conditional entry calls and also to asynchronous select statements where the triggering statement can be a dispatching call.

In a similar way we also permit timed calls on entries renamed as procedures. But note that we do not allow timed calls on generic formal subprograms even though they might be implemented as entries.

Another important point to note is that we can as usual assume the common properties of the class concerned. Thus in the case of a task interface we know that it must be implemented by a task and so the operations such as **abort** and the attributes Identity, Callable and so on can be applied. If we know that an interface is synchronized then we do know that it has to be implemented by a task or a protected type and so is task safe.

Typically an interface is implemented by a task or protected type but it can also be implemented by a singleton task or protected object despite the fact that singletons have no type name. Thus we might have

```
protected An_RW is new Sync_RW with
   procedure Write(X: in Item);
   procedure Read(X: out Item);
end;
```

with the obvious body. However we could not declare a single protected object similar to the type Multi_Prot_RW above. This is because we need a type name in order to declare the overriding procedure Read outside the protected object. So singleton implementations are possible provided that the interface can be implemented directly by the task or protected object without external subprograms.

Here is another example

```
type Map is protected interface;
procedure Put(M: Map; K: Key; V: Value) is abstract;
```

can be implemented by

```
protected A_Map is new Map with
   procedure Put(K: Key; V: Value);
   ...
end A_Map;
```

There is a fairly obvious rule about private types and synchronized interfaces. Both partial and full view must be synchronized or not. Thus if we wrote

```
type SI is synchronized interface;
type T is new SI with private;
```

then the full type T has to be a task type or protected type or possibly a synchronized, protected or task interface.

We conclude this discussion on interfaces by saying a few words about the use of the word limited. (Much of this has already been explained in the paper on the object oriented model but it is worth repeating in the context of concurrent types.) We always explicitly insert limited, synchronized, task, or protected in the case of a limited interface in order to avoid confusion. So to derive a new explicitly limited interface from an existing limited interface LI we write

```
type LI2 is limited interface and LI;
```

whereas in the case of normal types we can write

```
type LT is limited ...
type LT2 is new LT and LI with ...        -- LT2 is limited
```

then LT2 is limited by the normal derivation rules. Types take their limitedness from their parent (the first one in the list) and it does not have to be given explicitly on type derivation – although it can be in Ada 2005 thus

```
type LT2 is limited new LT and LI with ...
```

Remember the important rule that all descendants of a nonlimited interface have to be nonlimited because otherwise limited types could end up with an assignment operation.

This means that we cannot write

```
type NLI is interface;                    -- nonlimited
type LI is limited interface;             -- limited
task type TT is new NLI and LI with ...   -- illegal
```

This is illegal because the interface NLI in the declaration of the task type TT is not limited.

4 The Ravenscar profile

The purpose of the Ravenscar profile is to restrict the use of many tasking facilities so that the effect of the program is predictable. The profile was defined by the International Real-Time Ada Workshops which met twice at the remote village of Ravenscar on the coast of Yorkshire in North-East England. A general description of the principles and use of the profile in high integrity systems will be found in an ISO/IEC Technical Report [2] and so we shall not cover that material here.

Here is a historical interlude. It is reputed that the hotel in which the workshops were held was originally built as a retreat for King George III to keep a mistress. Another odd rumour is that he ordered all the natural trees to be removed and replaced by metallic ones whose metal leaves clattered in the wind. It also seems that Henry Bolingbroke landed at Ravenscar in July 1399 on his way to take the throne as Henry IV. Ravenscar is mentioned several times by Shakespeare in Act II of King Richard II; it is spelt Ravenspurg which is slightly confusing – maybe we need the ability to rename profile identifiers.

A profile is a mode of operation and is specified by the pragma Profile which defines the particular profile to be used. The syntax is

```
pragma Profile(profile_identifier
                    [ , profile_argument_associations]);
```

where profile_argument_associations is simply a list of pragma argument associations separated by commas.

Thus to ensure that a program conforms to the Ravenscar profile we write

```
pragma Profile(Ravenscar);
```

The general idea is that a profile is equivalent to a set of configuration pragmas.

In the case of Ravenscar the pragma is equivalent to the joint effect of the following pragmas

```
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
pragma Locking_Policy(Ceiling_Locking);
pragma Detect_Blocking;

pragma Restrictions(
      No_Abort_Statements,
      No_Dynamic_Attachment,
      No_Dynamic_Priorities,
      No_Implicit_Heap_Allocations,
      No_Local_Protected_Objects,
      No_Local_Timing_Events,
      No_Protected_Type_Allocators,
      No_Relative_Delay,
      No_Requeue_Statements,
      No_Select_Statements,
      No_Specific_Termination_Handlers,
      No_Task_Allocators,
      No_Task_Hierarchy,
      No_Task_Termination,
      Simple_Barriers,
      Max_Entry_Queue_Length => 1,
      Max_Protected_Entries => 1,
      Max_Task_Entries => 0,
      No_Dependence =>
                  Ada.Asynchronous_Task_Control,
      No_Dependence => Ada.Calendar,
      No_Dependence =>
                  Ada.Execution_Time.Group_Budget,
      No_Dependence => Ada.Execution_Time.Timers,
      No_Dependence => Ada.Task_Attributes);
```

The pragma Detect_Blocking plus many of the Restrictions identifiers are new to Ada 2005. These will now be described.

The pragma Detect_Blocking, as its name implies, ensures that the implementation will detect a potentially blocking operation in a protected operation and raise Program_Error. Without this pragma the implementation is not required to detect blocking and so tasks might be locked out for an unbounded time and the program might even deadlock.

The identifier No_Dynamic_Attachment means that there are no calls of the operations in the package Ada.Interrupts.

The identifier No_Dynamic_Priorities means that there is no dependence on the package Ada.Priorities as well as no uses of the attribute Priority (this is a new attribute for protected objects as explained at the end of this section).

Note that the rules are that you cannot read as well as not write the priorities – this applies to both the procedure for reading task priorities and reading the attribute for protected objects.

The identifier No_Local_Protected_Objects means that protected objects can only be declared at library level and the identifier No_Protected_Type_Allocators means that there are no allocators for protected objects or objects containing components of protected types.

The identifier No_Local_Timing_Events means that objects of the type Timing_Event in the package Ada.Real_Time. Timing_Events can only be declared at library level. This package is described in Section 6 below.

The identifiers No_Relative_Delay, No_Requeue_ Statements, and No_Select_Statements mean that there are no relative delay, requeue or select statements respectively.

The identifier No_Specific_Termination_Handlers means that there are no calls of the procedure Set_Specific_ Handler or the function Specific_Handler in the package Task_Termination and the identifier No_Task_Termination means that all tasks should run for ever. Note that we are permitted to set a fallback handler so that if any task does attempt to terminate then it will be detected.

The identifier Simple_Barriers means that the Boolean expression in a barrier of an entry of a protected object shall be either a static expression (such as True) or a Boolean component of the protected object itself.

The Restrictions identifier Max_Entry_Queue_Length sets a limit on the number of calls permitted on an entry queue. It is an important property of the Ravenscar profile that only one call is permitted at a time on an entry queue of a protected object.

The identifier No_Dependence is not specific to the Real-Time Systems annex and is properly described in the next paper. In essence it indicates that the program does not depend upon the given language defined package. In this case it means that a program conforming to the Ravenscar profile cannot use any of the packages Asynchronous_ Task_Control, Calendar, Execution_Time.Group_Budget, Execution_Time.Timers and Task_Attributes. Some of these packages are new and are described later in this paper.

Note that No_Dependence cannot be used for No_Dynamic_Attachment because that would prevent use of the child package Ada.Interrupts.Names.

All the other restrictions identifiers used by the Ravenscar profile were already defined in Ada 95. Note also that the identifier No_Asynchronous_Control has been moved to Annex J because it can now be replaced by the use of No_Dependence.

5 Scheduling and dispatching

Another area of increased flexibility in Ada 2005 is that of task dispatching policies. In Ada 95, the only predefined policy is FIFO_Within_Priorities although other policies are permitted. Ada 2005 provides further pragmas, policies and packages which facilitate many different mechanisms such as non-preemption within priorities, the familiar Round Robin using timeslicing, and the more recently acclaimed Earliest Deadline First (EDF) policy. Moreover it is possible to mix different policies according to priority level within a partition.

In order to accommodate these many changes, Section D.2 (Priority Scheduling) of the Reference Manual has been reorganized as follows

D.2.1 The Task Dispatching Model
D.2.2 Task Dispatching Pragmas
D.2.3 Preemptive Dispatching
D.2.4 Non-Preemptive Dispatching
D.2.5 Round Robin Dispatching
D.2.6 Earliest Deadline First Dispatching

Overall control is provided by two pragmas. They are

pragma Task_Dispatching_Policy(*policy*_identifier);

pragma Priority_Specific_Dispatching(*policy*_identifer, *first_priority*_expression, *last_priority*_expression);

The pragma Task_Dispatching_Policy, which already exists in Ada 95, applies the same policy throughout a whole partition. The pragma Priority_Specific_Dispatching, which is new in Ada 2005, can be used to set different policies for different ranges of priority levels.

The full set of predefined policies in Ada 2005 is

FIFO_Within_Priorities – This already exists in Ada 95. Within each priority level to which it applies tasks are dealt with on a first-in-first-out basis. Moreover, a task may preempt a task of a lower priority.

Non_Preemptive_FIFO_Within_Priorities – This is new in Ada 2005. Within each priority level to which it applies tasks run to completion or until they are blocked or execute a delay statement. A task cannot be preempted by one of higher priority. This sort of policy is widely used in high integrity applications.

Round_Robin_Within_Priorities – This is new in Ada 2005. Within each priority level to which it applies tasks are timesliced with an interval that can be specified. This is a very traditional policy widely used since the earliest days of concurrent programming.

EDF_Across_Priorities – This is new in Ada 2005. This provides Earliest Deadline First dispatching. The general idea is that within a range of priority levels, each task has a deadline and that with the earliest deadline is processed. This is a fashionable new policy and has mathematically provable advantages with respect to efficiency.

For further details of these policies consult the forthcoming book by Alan Burns and Andy Wellings [4].

These various policies are controlled by the package Ada.Dispatching plus two child packages. The root package has specification

```
package Ada.Dispatching is
   pragma Pure(Dispatching);
   Dispatching_Policy_Error: exception;
end Ada.Dispatching;
```

As can be seen this root package simply declares the exception Dispatching_Policy_Error which is used by the child packages.

The child package Round_Robin_Dispatching enables the setting of the time quanta for time slicing within one or more priority levels. Its specification is

```
with System;  use System;
with Ada.Real_Time;  use Ada.Real_Time;
package Ada.Dispatching.Round_Robin is
   Default_Quantum: constant Time_Span :=
                                    implementation-defined;
   procedure Set_Quantum(Pri: in Priority,
                                 Quantum: in Time_Span);
   procedure Set_Quantum(Low, High: in Priority;
                                 Quantum: in Time_Span);
   function Actual_Quantum(Pri: Priority)
                                          return Time_Span;
   function Is_Round_Robin(Pri: Priority)
                                            return Boolean;
end Ada.Dispatching.Round_Robin;
```

The procedures Set_Quantum enable the time quantum to be used for time slicing to be set for one or a range of priority levels. The default value is of course the constant Default_Quantum. The function Actual_Quantum enables us to find out the current value of the quantum being used for a particular priority level. Its identifier reflects the fact that the implementation may not be able to apply the exact actual value given in a call of Set_Quantum. The function Is_Round_Robin enables us to check whether the round robin policy has been applied to the given priority level. If we attempt to do something stupid such as set the quantum for a priority level to which the round robin policy does not apply then the exception Dispatching_Policy_Error is raised.

The other new policy concerns deadlines and is controlled by a new pragma Relative_Deadline and the child package Dispatching.EDF. The syntax of the pragma is

```
pragma Relative_Deadline(relative_deadline_expression);
```

The deadline of a task is a property similar to priority and both are used for scheduling. Every task has a priority of type Integer and every task has a deadline of type Ada.Real_Time.Time. Priorities can be set when a task is created by pragma Priority

```
task T is
   pragma Priority(P);
```

and deadlines can similarly be set by the pragma Relative_Deadline thus

```
task T is
   pragma Relative_Deadline(RD);
```

The expression RD has type Ada.Real_Time.Time_Span. Note carefully that the pragma sets the relative and not the absolute deadline. The initial absolute deadline of the task is

```
Ada.Real_Time.Clock + RD
```

where the call of Clock is made between task creation and the start of its activation.

Both pragmas Priority and Relative_Deadline can appear in the main subprogram and they then apply to the environment task. If they appear in any other subprogram then they are ignored. Both properties can also be set via a discriminant. In the case of priorities we can write

```
task type TT(P: Priority) is
   pragma Priority(P);
   ...
end;

High_Task: TT(13);
Low_Task: TT(7);
```

We cannot do the direct equivalent for deadlines because Time_Span is private and so not discrete. We have to use an access discriminant thus

```
task type TT(RD: access Timespan) is
   pragma Relative_Deadline(RD);
   ...
end;

One_Sec: aliased constant Time_Span := Seconds(1);
Ten_Mins: aliased constant Time_Span := Minutes(10);

Hot_Task: TT(One_Sec'Access);
Cool_Task: TT(Ten_Mins'Access);
```

Note incidentally that functions Seconds and Minutes have been added to the package Ada.Real_Time. Existing functions Nanoseconds, Microseconds and Milliseconds in Ada 95 enable the convenient specification of short real time intervals (values of type Time_Span). However, the specification of longer intervals such as four minutes meant writing something like Milliseconds(240_000) or perhaps 4*60*Milliseconds(1000). In view of the fact that EDF scheduling and timers (see Section 6) would be likely to require longer times the functions Seconds and Minutes are added in Ada 2005. There is no function Hours because the range of time spans is only guaranteed to be 3600 seconds anyway.

If a task is created and it does not have a pragma Priority then its initial priority is that of the task that created it. If a task does not have a pragma Relative_Deadline then its initial absolute deadline is the constant Default_Deadline in the package Ada.Dispatching.EDF; this constant has the value Ada.Real_Time.Time_Last (effectively the end of the universe).

Priorities can be dynamically manipulated by the subprograms in the package Ada.Dynamic_Priorities and deadlines can similarly be manipulated by the subprograms in the package Ada.Dispatching.EDF whose specification is

```
with Ada.Real_Time;  use Ada.Real_Time;
with Ada.Task_Identification; use Ada.Task_Identification;
package Ada.Dispatching.EDF is
  subtype Deadline is Ada.Real_Time.Time;
  Default_Deadline: constant Deadline := Time_Last;
  procedure Set_Deadline(D: in Deadline;
                         T: in Task_Id := Current_Task);
  procedure Delay_Until_And_Set_Deadline
                        (Delay_Until_Time: in Time;
                         Deadline_Offset: in Time_Span);
  function Get_Deadline(T: Task_Id := Current_Task)
                                          return Deadline;
end Ada.Dispatching.EDF;
```

The subtype Deadline is just declared as a handy abbreviation. The constant Default_Deadline is set to the end of the universe as already mentioned. The procedure Set_Deadline sets the deadline of the task concerned to the value of the parameter D. The long-winded Delay_Until_And_Set_Deadline delays the task concerned until the value of Delay_Until_Time and sets its deadline to be the interval Deadline_Offset from that time – this is useful for periodic tasks. The function Get_Deadline enables us to find the current deadline of a task.

It is important to note that this package can be used to set and retrieve deadlines for tasks whether or not they are subject to EDF dispatching. We could for example use an ATC on a deadline overrun (ACT = Asynchronous Transfer of Control using a select statement). Hence there is no function Is_EDF corresponding to Is_Round_Robin and calls of the subprograms in this package can never raise the exception Dispatching_Policy_Error.

If we attempt to apply one of the subprograms in this package to a task that has already terminated then Tasking_Error is raised. If the task parameter is Null_Task_Id then Program_Error is raised.

As mentioned earlier, a policy can be selected for a whole partition by for example

```
pragma Task_Dispatching_Policy
              (Round_Robin_Within_Priorities);
```

whereas in order to mix different policies across different priority levels we can write

```
pragma Priority_Specific_Dispatching
              (Round_Robin_Within_Priority, 1, 1);
pragma Priority_Specific_Dispatching
              (EDF_Across_Priorities, 2, 10);
pragma Priority_Specific_Dispatching
              (FIFO_Within_Priority, 11, 24);
```

This sets Round Robin at priority level 1, EDF at levels 2 to 10, and FIFO at levels 11 to 24.

Note that if we write

```
pragma Priority_Specific_Dispatching
              (EDF_Across_Priorities, 2, 5);
pragma Priority_Specific_Dispatching
              (EDF_Across_Priorities, 6, 10);
```

then this is not the same us

```
pragma Priority_Specific_Dispatching
              (EDF_Across_Priorities, 2, 10);
```

despite the fact that the two ranges in the first case are contiguous. This is because in the first case any task in the 6 to 10 range will take precedence over any task in the 2 to 5 range whatever the deadlines. If there is just one range then only the deadlines count in deciding which tasks are scheduled.

This is emphasized by the fact that the policy name uses Across rather than Within. For other policies such as Round_Robin_Within_Priority two contiguous ranges would be the same as a single range.

We conclude this section with a few words about ceiling priorities.

In Ada 95, the priority of a task can be changed but the ceiling priority of a protected object cannot be changed. It is permanently set when the object is created using the pragma Priority. This is often done using a discriminant so that at least different objects of a given protected type can have different priorities. Thus we might have

```
protected type PT(P: Priority) is
  pragma Priority(P);
  ...
end PT;

PO: PT(7);                -- ceiling priority is 7
```

The fact that the ceiling priority of a protected object is static can be a nuisance in many applications especially when the priority of tasks can be dynamic. A common workaround is to give a protected object a higher ceiling than needed in all circumstances (often called "the ceiling of ceilings"). This results in tasks having a higher active priority than necessary when accessing the protected object and this can interfere with the processing of other tasks in the system and thus upset overall schedulability. Moreover, it means that a task of high priority can access an object when it should not (if a task with a priority higher than the ceiling priority of a protected object attempts to access the object then Program_Error is raised – if the object has an inflated priority then this check will pass when it should not).

This difficulty is overcome in Ada 2005 by allowing protected objects to change their priority. This is done through the introduction of an attribute Priority which applies just to protected objects. It can only be accessed within the body of the protected object concerned.

As an example a protected object might have a procedure to change its ceiling priority by a given amount. This could be written as follows

```
protected type PT is
  procedure Change_Priority(Change: in Integer);
  ...
end;

protected body PT is
  procedure Change_Priority(Change: in Integer) is
  begin
    ...                        -- PT'Priority has old value here
    PT'Priority := PT'Priority + Change;
    ...                        -- PT'Priority has new value here
    ...
  end Change_Priority;
  ...
end PT;
```

Changing the ceiling priority is thus done while mutual exclusion is in force. Although the value of the attribute itself is changed immediately the assignment is made, the actual ceiling priority of the protected object is only changed when the protected operation (in this case the call of Change_Priority) is finished.

Note the unusual syntax. Here we permit an attribute as the destination of an assignment statement. This happens nowhere else in the language. Other forms of syntax were considered but this seemed the most expressive.

6 CPU clocks and timers

Ada 2005 introduces three different kinds of timers. Two are concerned with monitoring the CPU time of tasks – one applies to a single task and the other to groups of tasks. The third timer measures real time rather than execution time and can be used to trigger events at specific real times. We will look first at the CPU timers because that introduces more new concepts.

The execution time of one or more tasks can be monitored and controlled by the new package Ada.Execution_Time plus two child packages.

Ada.Execution_Time – this is the root package and enables the monitoring of execution time of individual tasks.

Ada.Execution_Time.Timers – this provides facilities for defining and enabling timers and for establishing a handler which is called by the run time system when the execution time of the task reaches a given value.

Ada.Execution_Time.Group_Budgets – this enables several tasks to share a budget and provides means whereby action can be taken when the budget expires.

The execution time of a task, or CPU time as it is commonly called, is the time spent by the system executing the task and services on its behalf. CPU times are represented by the private type CPU_Time. This type and various subprograms are declared in the root package Ada.Execution_Time whose specification is as follows (as before we have added some use clauses in order to ease the presentation)

```
with Ada.Task_Identification; use Ada.Task_Identification;
with Ada.Real_Time;  use Ada.Real_Time;
package Ada.Execution_Time is

  type CPU_Time is private;
  CPU_Time_First: constant CPU_Time;
  CPU_Time_Last: constant CPU_Time;
  CPU_Time_Unit: constant :=
                    implementation-defined-real-number;
  CPU_Tick: constant Time_Span;

  function Clock(T: Task_Id := Current_Task)
                                          return CPU_Time;
  function "+" (Left: CPU_Time; Right: Time_Span)
                                          return CPU_Time;
  function "+" (Left: Time_Span; Right: CPU_Time)
                                          return CPU_Time;
  function "–" (Left: CPU_Time; Right: Time_Span)
                                          return CPU_Time;

  function "–" (Left: CPU_Time; Right: CPU_Time)
                                         return Time_Span;

  function "<" (Left, Right: CPU_Time) return Boolean;
  function "<=" (Left, Right: CPU_Time) return Boolean;
  function ">" (Left, Right: CPU_Time) return Boolean;
  function ">=" (Left, Right: CPU_Time) return Boolean;

  procedure Split(T: in CPU_Time;
          SC: out Seconds_Count; TS: out Time_Span);
  function Time_Of(SC: Seconds_Count;
                   TS: Time_Span := Time_Span_Zero)
                                          return CPU_Time;

private
  ... -- not specified by the language
end Ada.Execution_Time;
```

The CPU time of a particular task is obtained by calling the function Clock with the task as parameter. It is set to zero at task creation.

The constants CPU_Time_First and CPU_Time_Last give the range of values of CPU_Time. CPU_Tick gives the average interval during which successive calls of Clock give the same value and thus is a measure of the accuracy whereas CPU_Time_Unit gives the unit of time measured in seconds. We are assured that CPU_Tick is no greater than one millisecond and that the range of values of CPU_Time is at least 50 years (provided always of course that the implementation can cope).

The various subprograms perform obvious operations on the type CPU_Time and the type Time_Span of the package Ada.Real_Time.

A value of type CPU_Time can be converted to a Seconds_Count plus residual Time_Span by the function Split which is similar to that in the package Ada.Real_Time. The function Time_Of similarly works in the opposite direction. Note the default value of Time_Span_Zero for the second parameter – this enables times of exact numbers of seconds to be given more conveniently thus

```
Four_Secs: CPU_Time := Time_Of(4);
```

In order to find out when a task reaches a particular CPU time we can use the facilities of the child package Ada.Execution_Time.Timers whose specification is

```
with System;  use System;
package Ada.Execution_Time.Timers is

   type Timer(T: not null access constant Task_Id) is
                                        tagged limited private;
   type Timer_Handler is access
                  protected procedure (TM: in out Timer);

   Min_Handler_Ceiling: constant Any_Priority :=
                                        implementation-defined;

   procedure Set_Handler(TM: in out Timer;
          In_Time: Time_Span; Handler: Timer_Handler);
   procedure Set_Handler(TM: in out Timer;
          At_Time: CPU_Time; Handler: Timer_Handler);
   function Current_Handler(TM: Timer)
                                        return Timer_Handler;
   procedure Cancel_Handler(TM: in out Timer;
                                    Cancelled: out Boolean);

   function Time_Remaining(TM: Timer)
                                            return Time_Span;

   Timer_Resource_Error: exception;

private
   ... -- not specified by the language
end Ada.Execution_Time.Timers;
```

The general idea is that we declare an object of type Timer whose discriminant identifies the task to be monitored – note the use of **not null** and **constant** in the discriminant. We also declare a protected procedure which takes the timer as its parameter and which performs the actions required when the CPU_Time of the task reaches some value. Thus to take some action (perhaps abort for example although that would be ruthless) when the CPU_Time of the task My_Task reaches 2.5 seconds we might first declare

```
My_Timer: Timer(My_Task'Identity'Access);
Time_Max: CPU_Time := Time_Of(2, Milliseconds(500));
```

and then

```
protected Control is
   procedure Alarm(TM: in out Timer);
end;
...
protected body Control is
   procedure Alarm(TM: in out Timer) is
   begin
      -- abort the task
      Abort_Task(TM.T.all);
   end Alarm;
end Control;
```

Finally we set the timer in motion by calling the procedure Set_Handler which takes the timer, the time value and (an access to) the protected procedure thus

```
Set_Handler(My_Timer, Time_Max, Control.Alarm'Access);
```

and then when the CPU time of the task reaches Time_Max, the protected procedure Control.Alarm is executed. Note how the timer object incorporates the information regarding the task concerned using an access discriminant T and that this is passed to the handler via its parameter TM.

Aborting the task is perhaps a little violent. Another possibility is simply to reduce its priority so that it is no longer troublesome, thus

```
-- cool that task
Set_Priority(Priority'First, TM.T.all);
```

Another version of Set_Handler enables the timer to be set for a given interval (of type Time_Span).

The handler associated with a timer can be found by calling the function Current_Handler. This returns null if the timer is not set in which case we say that the timer is clear.

When the timer expires, and just before calling the protected procedure, the timer is set to the clear state. One possible action of the handler, having perhaps made a note of the expiration of the timer, it to set the handler again or perhaps another handler. So we might have

```
protected body Control is
   procedure Alarm(TM: in out Timer) is
   begin
      Log_Overflow(TM);   -- note that timer had expired
      -- and then reset it for another 500 milliseconds
      Set_Handler(TM, Milliseconds(500), Kill'Access);
   end Alarm;

   procedure Kill(TM: in out Timer) is
   begin
      -- expired again so kill it
      Abort_Task(TM.T.all);
   end Kill;
end Control;
```

In this scenario we make a note of the fact that the task has overrun and then give it another 500 milliseconds but with the handler Control.Kill so that the second time is the last chance.

Setting the value of 500 milliseconds directly in the call is a bit crude. It might be better to parameterize the protected type thus

```
protected type Control(MS: Integer) is ...
...
My_Control: Control(500);
```

and then the call of Set_Handler in the protected procedure Alarm would be

```
Set_Handler(TM, Milliseconds(MS), Kill'Access);
```

Observe that overload resolution neatly distinguishes whether we are calling Set_Handler with an absolute time or a relative time.

The procedure Cancel_Handler can be used to clear a timer. The out parameter Cancelled is set to True if the timer was in fact set and False if it was clear. The function

Time_Remaining returns Time_Span_Zero if the timer is not set and otherwise the time remaining.

Note also the constant Min_Handler_Ceiling. This is the minimum ceiling priority that the protected procedure should have to ensure that ceiling violation cannot occur.

This timer facility might be implemented on top of a POSIX system. There might be a limit on the number of timers that can be supported and an attempt to exceed this limit will raise Timer_Resource_Error.

We conclude by summarizing the general principles. A timer can be set or clear. If it is set then it has an associated (non-null) handler which will be called after the appropriate time. The key subprograms are Set_Handler, Cancel_Handler and Current_Handler. The protected procedure has a parameter which identifies the event for which it has been called. The same protected procedure can be the handler for many events. The same general structure applies to other kinds of timers which will now be described.

In order to program various so-called aperiodic servers it is necessary for tasks to share a CPU budget.

This can be done using the child package Ada.Execution_Time.Group_Budgets whose specification is

```
with System;  use System;
package Ada.Execution_Time.Group_Budgets is

   type Group_Budget is tagged limited private;
   type Group_Budget_Handler is access
      protected procedure (GB: in out Group_Budget);

   type Task_Array is array (Positive range <>) of
                                                Task_Id;

   Min_Handler_Ceiling: constant Any_Priority :=
                                   implementation-defined;

   procedure Add_Task(GB: in out Group_Budget;
                                                  T: in Task_Id);
   procedure Remove_Task(GB: in out Group_Budget;
                                                  T: in Task_Id);
   function Is_Member(GB: Group_Budget; T: Task_Id)
                                                return Boolean;
   function Is_A_Group_Member(T: Task_Id)
                                                return Boolean;
   function Members(GB: Group_Budget)
                                              return Task_Array;

   procedure Replenish(GB: in out Group_Budget;
                                              To: in Time_Span);
   procedure Add(GB: in out Group_Budget;
                                           Interval: in Time_Span);

   function Budget_Has_Expired(GB: Group_Budget)
                                                return Boolean;
   function Budget_Remaining(GB: Group_Budget)
                                              return Time_Span;

   procedure Set_Handler(GB: in out Group_Budget;
                         Handler: in Group_Budget_Handler);
   function Current_Handler(GB: Group_Budget)
                                     return Group_Budget_Handler;
   procedure Cancel_Handler(GB: in out Group_Budget;
                                    Cancelled: out Boolean);

   Group_Budget_Error: exception;

private
   ... -- not specified by the language
end Ada.Execution_Time.Group_Budgets;
```

This has much in common with its sibling package Timers but there are a number of important differences.

The first difference is that we are here considering a CPU budget shared among several tasks. The type Group_Budget both identifies the group of tasks it covers and the size of the budget.

Various subprograms enable tasks in a group to be manipulated. The procedures Add_Task and Remove_Task add or remove a task. The function Is_Member identifies whether a task belongs to a specific group whereas Is_A_Group_Member identifies whether a task belongs to any group. A task cannot be a member of more than one group. An attempt to add a task to more than one group or remove it from the wrong group and so on raises Group_Budget_Error. Finally the function Members returns all the members of a group as an array.

The value of the budget (initially Time_Span_Zero) can be loaded by the procedure Replenish and increased by the procedure Add. Whenever a budget is non-zero it is counted down as the tasks in the group execute and so consume CPU time. Whenever a budget goes to Time_Span_Zero it is said to have become exhausted and is not reduced further. Note that Add with a negative argument can reduce a budget – it can even cause it to become exhausted but not make it negative.

The function Budget_Remaining simply returns the amount left and Budget_Has_Expired returns True if the budget is exhausted and so has value Time_Span_Zero.

Whenever a budget *becomes* exhausted (that is when the value transitions to zero) a hander is called if one has been set. A handler is a protected procedure as before and procedures Set_Handler, Cancel_Handler, and function Current_Handler are much as expected. But a major difference is that Set_Handler does not set the time value of the budget since that is done by Replenish and Add. The setting of the budget and the setting of the handler are decoupled in this package. Indeed a handler can be set even though the budget is exhausted and the budget can be counting down even though no handler is set. The reason for the different approach simply reflects the usage paradigm for the feature.

So we could set up a mechanism to monitor the CPU time usage of a group of three tasks TA, TB, and TC by first declaring an object of type Group_Budget, adding the three tasks to the group and then setting an appropriate handler. Finally we call Replenish which sets the counting mechanism going. So we might write

```
ABC: Group_Budget;
...
Add_Task(ABC, TA'Identity);
Add_Task(ABC, TB'Identity);
Add_Task(ABC, TC'Identity);

Set_Handler(ABC, Control.Monitor'Access);
Replenish(ABC, Seconds(10));
```

Remember that functions Seconds and Minutes have been added to the package Ada.Real_Time.

The protected procedure might be

```
protected body Control is
  procedure Monitor(GB: in out Group_Budget) is
  begin
    Log_Budget;
    Add(GB, Seconds(10));   -- add more time
  end Monitor;
end Control;
```

The procedure Monitor logs the fact that the budget was exhausted and then adds a further 10 seconds to it. Remember that the handler remains set all the time in the case of group budgets whereas in the case of the single task timers it automatically becomes cleared and has to be set again if required.

If a task terminates then it is removed from the group as part of the finalization process.

Note that again there is the constant Min_Handler_Ceiling.

The final kind of timer concerns real time rather than CPU time and so is provided by a child package of Ada.Real_Time whereas the timers we have seen so far were provided by child packages of Ada.Execution_Time. The specification of the package Ada.Real_Time.Timing_Events is

```
package Ada.Real_Time.Timing_Events is

  type Timing_Event is tagged limited private;
  type Timing_Event_Handler is access
    protected procedure (Event: in out Timing_Event);

  procedure Set_Handler(Event: in out Timing_Event;
        At_Time: Time; Handler: Timing_Event_Handler);

 procedure Set_Handler(Event: in out Timing_Event;
In_Time: Time_Span; Handler: Timing_Event_Handler);

  function Is_Handler_Set(Event: Timing_Event)
                                        return Boolean;

  function Current_Handler(Event: Timing_Event)
                              return Timing_Event_Handler;

  procedure Cancel_Handler(Event: in out Timing_Event;
                                 Cancelled: out Boolean);

  function Time_Of_Event(Event: Timing_Event)
                                            return Time;

private
  ... -- not specified by the language
end Ada.Real_Time.Timing_Events;
```

This package provides a very low level facility and does not involve Ada tasks at all. It has a very similar pattern to the package Execution_Time.Timers. A handler can be set by Set_Handler and again there are two versions one for a relative time and one for absolute time. There are also subprograms Current_Handler and Cancel_Handler. If no handler is set then Current_Handler returns null.

Set_Handler also specifies the protected procedure to be called when the time is reached. Times are of course specified using the type Real_Time rather than CPU_Time.

A minor difference is that this package has a function Time_Of_Event rather than Time_Remaining.

A simple example was given in the introductory paper. We repeat it here for convenience. The idea is that we wish to ring a pinger when our egg is boiled after four minutes. The protected procedure might be

```
protected body Egg is
  procedure Is_Done(Event: in out Timing_Event) is
  begin
    Ring_The_Pinger;
  end Is_Done;
end Egg;
```

and then

```
Egg_Done: Timing_Event;
Four_Min: Time_Span := Minutes(4);
...
Put_Egg_In_Water;
Set_Handler(Event => Egg_Done, In_Time => Four_Min,
                          Handler => Egg.Is_Done'Access);
-- now read newspaper whilst waiting for egg
```

This is unreliable because if we are interrupted between the calls of Put_Egg_In_Water and Set_Handler then the egg will be boiled for too long. We can overcome this by adding a further procedure to the protected object so that it becomes

```
protected Egg is
  procedure Boil(For_Time: in Time_Span);
  procedure Is_Done(Event: in out Timing_Event);
end Egg;

protected body Egg is

  Egg_Done: Timing_Event;

  procedure Boil (For_Time: in Time_Span) is
  begin
    Put_Egg_In_Water;
    Set_Handler(Egg_Done, For_Time, Is_Done'Access);
  end Boil;

  procedure Is_Done (Event: in out Timing_Event) is
  begin
    Ring_The_Pinger;
  end Is_Done;
end Egg;
```

This is much better. The timing mechanism is now completely encapsulated in the protected object and the

procedure Is_Done is no longer visible outside. So all we have to do is

```
Egg.Boil(Minutes(4));
-- now read newspaper whilst waiting for egg
```

Of course if the telephone rings as the pinger goes off and before we have a chance to eat the egg then it still gets overdone. One solution is to eat the egg within the protected procedure Is_Done as well. A gentleman would never let a telephone call disturb his breakfast.

One protected procedure could be used to respond to several events. In the case of the CPU timer the discriminant of the parameter identifies the task; in the case of the group and real-time timers, the parameter identifies the event.

If we want to use the same timer for several events then various techniques are possible. Note that the timers are limited so we cannot test for them directly. However, they are tagged and so can be extended. Moreover, we know that they are passed by reference and that the parameters are considered aliased.

Suppose we are boiling six eggs in one of those French breakfast things with a different coloured holder for each egg. We can write

```
type Colour is (Black, Blue, Red, Green, Yellow, Purple);

Eggs_Done: array (Colour) of aliased Timing_Event;
```

We can then set the handler for the egg in the red holder by something like

```
Set_Handler(Eggs_Done(Red), For_Time, Is_Done'Access);
```

and then the protected procedure might be

```
procedure Is_Done(E: in out Timing_Event) is
begin
  for C in Colour loop
    if E'Access = Eggs_Done(C)'Access then
      -- egg in holder colour C is ready
      ...
      return;
    end if;
  end loop;
  -- falls out of loop – unknown event!
  raise Not_An_Egg ;
end Is_Done;
```

Although this does work it is more than a little distasteful to compare access values in this way and moreover requires a loop to see which event occurred.

A much better approach is to use type extension and view conversions. First we extend the type Timing_Event to include additional information about the event (in this case the colour) so that we can identify the particular event from within the handler

```
type Egg_Event is new Timing_Event with
  record
    Event_Colour: Colour;
  end record;
```

We then declare an array of these extended events (they need not be aliased)

```
Eggs_Done: array (Colour) of Egg_Event;
```

We can now call Set_Handler for the egg in the red holder

```
Set_Handler(Eggs_Done(Red), For_Time, Is_Done'Access);
```

This is actually a call on the Set_Handler for the type Egg_Event inherited from Timing_Event. But it is the same code anyway.

Remember that values of tagged types are always passed by reference. This means that from within the procedure Is_Done we can recover the underlying type and so discover the information in the extension. This is done by using view conversions.

In fact we have to use two view conversions, first we convert to the class wide type Timing_Event'Class and then to the specific type Egg_Event. And then we can select the component Event_Colour. In fact we can do these operations in one statement thus

```
procedure Is_Done(E: in out Timing_Event) is
  C: constant Colour :=
       Egg_Event(Timing_Event'Class(E)).Event_Colour;
begin
  -- egg in holder colour C is ready
  ...
end Is_Done;
```

Note that there is a check on the conversion from the class wide type Timing_Event'Class to the specific type Egg_Event to ensure that the object passed as parameter is indeed of the type Egg_Event (or a further extension of it). If this fails then Tag_Error is raised. In order to avoid this possibility we can use a membership test. For example

```
procedure Is_Done(E: in out Timing_Event) is
  C: Colour;
begin
  if Timing_Event'Class(E) in Egg_Event then
    C :=
      Egg_Event(Timing_Event'Class(E)).Event_Colour;
    -- egg in holder colour C is ready
    ...
  else
    -- unknown event – not an egg event!
    raise Not_An_Egg;
  end if;
end Is_Done;
```

The membership test ensures that the event is of the specific type Egg_Event. We could avoid the double conversion to the class wide type by introducing an intermediate variable.

It is important to appreciate that no dispatching is involved in these operations at all – everything is static apart from the membership test.

Of course, it would have been a little more flexible if the various subprograms took a parameter of type Timing_Event'Class but this would have conflicted with the

Restrictions identifier No_Dispatch. Note that Ravenscar itself does not impose No_Dispatch but the restriction is in the High-Integrity annex and thus might be imposed on some high-integrity applications which might nevertheless wish to use timers in a simple manner.

A few minor points of difference between the timers are worth summarizing.

The two CPU timers have a constant Min_Handler_Ceiling. This prevents ceiling violation. It is not necessary for the real-time timer because the call of the protected procedure is treated like an interrupt and thus is at interrupt ceiling level.

The group budget timer and the real-time timer do not have an exception corresponding to Timer_Resource_Error for the single task CPU timer. As mentioned above, it is anticipated that the single timer might be implemented on top of a POSIX system in which case there might be a limit to the number of timers especially since each task could be using several timers. In the group case, a task can only be in one group so the number of group timers is necessarily less than the number of tasks and no limit is likely to be exceeded. In the real-time case the events are simply placed on the delay queue and no other resources are required anyway.

It should also be noted that the group timer could be used to monitor the execution time of a single task. However, a task can only be in one group and so only one timer could be applied to a task that way whereas, as just mentioned, the single CPU timer is quite different since a given task could have several timers set for it to expire at different times. Thus both kinds of timers have their own distinct usage patterns.

7 High-Integrity Systems Annex

There are a few changes to this annex. The most noticeable is that its title has been changed from Safety and Security to High-Integrity Systems. This reflects common practice in that high-integrity is now the accepted general term for systems such as safety-critical systems and security-critical systems.

There are some small changes to reflect the introduction of the Ravenscar profile. It is clarified that tasking is permitted in a high-integrity system provided that it is well controlled through, for example, the use of the Ravenscar profile.

A new pragma Partition_Elaboration_Policy is introduced. Its syntax is

```
pragma Partition_Elaboration_Policy(policy_identifier);
```

Two policy identifiers are predefined, namely, Concurrent and Sequential. The pragma is a configuration pragma and so applies throughout a partition. The default policy is Concurrent.

The normal behaviour in Ada when a program starts is that a task declared at library level is activated by the environment task and can begin to execute before all library level elaboration is completed and before the main subprogram is called by the environment task. Race conditions can arise especially when several library tasks are involved. Problems also arise with the attachment of interrupt handlers.

If the policy Sequential is specified then the rules are changed. The following things happen in sequence

- The elaboration of all library units takes place (this is done by the environment task) but library tasks are not activated (we say their activation is deferred). Similarly the attachment of interrupt handlers is deferred.
- The environment task then attaches the interrupts.
- The library tasks are then activated. While this is happening the environment task is suspended.
- Finally, the environment task then executes the main subprogram in parallel with the executing tasks.

Note that from the library tasks' point of view they go seamlessly from activation to execution. Moreover, they are assured that all library units will have been elaborated and all handlers attached before they execute.

If Sequential is specified then

```
pragma Restrictions(No_Task_Hierarchy);
```

must also be specified. This ensures that all tasks are at library level.

A final small point is that the Restrictions identifiers No_Unchecked_Conversion and No_Unchecked_Deallocation are now banished to Annex J because No_Dependence can be used instead.

References

[1] ISO/IEC JTC1/SC22/WG9 N412 (2002) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of the Amendment*.

[2] ISO/IEC TR 24718:2004 (2004) *Guide for the use of the Ada Ravenscar Profile in high integrity systems*. This is based on University of York Technical Report YCS-2003-348 (2003).

[3] J. G. P. Barnes (1998) *Programming in Ada 95*, 2nd ed., Addison-Wesley.

[4] A. Burns and A. Wellings (2006) *Concurrent and Real-Time Programming In Ada 2005*, Cambridge University Press.

Rationale for Ada 2005: 5 Exceptions, generics etc

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 paper describes various improvements in a number of general areas in Ada 2005.

There are some minor almost cosmetic improvements in the exceptions area which add to convenience rather than functionality. There are some important changes in the numerics area: one concerns mixing signed and unsigned integers and another concerns fixed point multiplication and division.

There are also a number of additional pragmas and Restrictions identifiers mostly of a safety-related nature.

Finally there are a number of improvements in the generics area such as better control of partial parameters of formal packages.

Keywords: rationale, Ada 2005.

1 Overview of changes

The areas mentioned in this paper are not specifically mentioned in the WG9 guidance document [1] other than under the request to remedy shortcomings and improve interfacing.

The following Ada Issues cover the relevant changes and are described in detail in this paper.

161 Preelaborable initialization

216 Unchecked unions – variants without discriminant

224 pragma Unsuppress

241 Testing for null occurrence

251 Abstract interfaces to provide multiple inheritance

257 Restrictions for implementation defined entities

260 Abstract formal subprograms & dispatching constructors

267 Fast float to integer conversion

286 Assert pragma

317 Partial parameter lists for formal packages

329 pragma No_Return – procedures that never return

340 Mod attribute

361 Raise with message

364 Fixed point multiply and divide

368 Restrictions for obsolescent features

381 New Restrictions identifier – No_Dependence

394 Redundant Restrictions identifiers and Ravenscar

398 Parameters of formal packages given at most once

400 Wide and wide-wide images

414 pragma No_Return for overriding procedures

417 Lower bound of functions in Ada.Exceptions etc

419 Limitedness of derived types

420 Resolution of universal operations in Standard

423 Renaming, null exclusion and formal objects

These changes can be grouped as follows.

First there are some minor changes to exception handling. There are neater means for testing for null occurrence and raising an exception with a message (241, 361) and also wide and wide-wide versions of some procedures (400, 417).

The numerics area has a number of small but important changes. They are the introduction of an attribute Mod to aid conversion between signed and unsigned integers (340); changes to the rules for fixed point multiplication and division which permit user-defined operations (364, 420); and an attribute Machine_Rounding which can be used to aid fast conversions from floating to integer types (267).

A number of new pragmas and Restrictions identifiers have been added. These generally make for more reliable programming. The pragmas are: Assert, No_Return, Preelaborable_Initialization, Unchecked_Union, and Unsuppress (161, 216, 224, 286, 329, 414). The restrictions identifiers are No_Dependence, No_Implementation_Pragmas, No_Implementation_Restrictions, and No_Obsolescent_Features (257, 368, 381). Note that there are also other new pragmas and new restrictions identifiers concerned with tasking as described in the previous paper. However, the introduction of No_Dependence means that the identifiers No_Asynchronous_Control, No_Unchecked_Conversion and No_Unchecked_Deallocation are now obsolescent (394).

Finally there are changes in generic units. There are changes in generic parameters which are consequences of changes in other areas such as the introduction of interfaces and dispatching constructors as described in the paper on the object oriented model (parts of 251 and 260); there are also changes to formal access and derived types (419, 423). Also, it is now possible to give just some parameters of a formal package in the generic formal part (317, 398).

2 Exceptions

There are two minor improvements in this area.

One concerns the detection of a null exception occurrence which might be useful in a routine for analysing a log of exceptions. This is tricky because although a constant Null_Occurrence is declared in the package Ada.Exceptions, the type Exception_Occurrence is limited and no equality is provided. So the obvious test cannot be performed.

We can however apply the function Exception_Identity to a value of the type Exception_Occurrence and this returns the corresponding Exception_Id. Thus we could check to see whether a particular occurrence X was caused by Program_Error by writing

```
if Exception_Identity(X) = Program_Error'Identity then
```

However, in Ada 95, applying Exception_Identity to the value Null_Occurrence raises Constraint_Error so we have to resort to a revolting trick such as declaring a function as follows

```
function Is_Null_Occurrence(X: Exception_Occurrence)
                                        return Boolean is
  Id: Exception_Id;
begin
  Id := Exception_Identity(X);
  return False;
exception
  when Constraint_Error => return True;
end Is_Null_Occurrence;
```

We can now write some general analysis routine as

```
procedure Process_Ex(X: in Exception_Occurrence) is
begin
  if Is_Null_Occurrence(X) then          -- OK in Ada 95
    -- process the case of a null occurrence
  else
    -- process proper occurrences
  end if;
end Process_Ex;
```

But the detection of Constraint_Error in Is_Null_Occurrence is clearly bad practice since it would be all too easy to mask some other error by mistake. Accordingly, in Ada 2005, the behaviour of Exception_Identity is changed to return Null_Id when applied to Null_Occurrence. So we can now dispense with the dodgy function Is_Null_Occurrence and just write

```
procedure Process_Ex(X: in Exception_Occurrence) is
begin
  if Exception_Identity(X) = Null_Id then   -- OK in 2005
    -- process the case of a null occurrence
  else
    -- process proper occurrences
  end if;
end Process_Ex;
```

Beware that, technically, we now have an incompatibility between Ada 95 and Ada 2005 since the nasty function Is_Null_Occurrence will always return False in Ada 2005.

Observe that Constraint_Error is also raised if any of the three functions Exception_Name, Exception_Message, or Exception_Information are applied to the value Null_Occurrence so the similar behaviour with Exception_Identity in Ada 95 is perhaps understandable at first sight. However, it is believed that it was not the intention of the language designers but got in by mistake. Actually the change described here was originally classified as a correction to Ada 95 but later reclassified as an amendment in order to draw more attention to it because of the potential incompatibility.

The other change in the exception area concerns the raise statement. It is now possible (optionally of course) to supply a message thus

```
raise An_Error with "A message";
```

This is purely for convenience and is identical to writing

```
Raise_Exception(An_Error'Identity, "A message");
```

There is no change to the form of raise statement without an exception which simply reraises an existing occurrence.

Note the difference between

```
raise An_Error;      -- message is implementation defined
```

and

```
raise An_Error with "";              -- message is null
```

In the first case a subsequent call of Exception_Message returns implementation defined information about the error whereas in the second case it simply returns the given message which in this example is a null string.

Some minor changes to the procedure Raise_Exception are mentioned in Section 4 below.

There are also additional functions in the package Ada.Exceptions to return the name of an exception as a Wide_String or Wide_Wide_String. They have identifiers Wide_Exception_Name and Wide_Wide_Exception_Name and are overloaded to take a parameter of type Exception_Id or Exception_Occurrence. The lower bound of the strings returned by these functions and by the existing functions Exception_Name, Exception_Message and Exception_Information is 1 (Ada 95 forgot to state this for the existing functions). The reader will recall that similar additional functions (and forgetfulness) in the package Ada.Tags were mentioned in the paper on the object oriented model.

3 Numerics

Although Ada 95 introduced unsigned integer types in the form of modular types, nevertheless, the strong typing rules of Ada have not made it easy to get unsigned and signed integers to work together. The following discussion using Ada 95 is based on that in AI-340.

Suppose we wish to implement a simulation of a typical machine which has addresses and offsets. We make it a generic

```
generic
  type Address_Type is mod <>;
  type Offset_Type is range <>;
  ...
package Simulator is
  function Calc_Address(Base_Add: Address_Type;
                Offset: Offset_Type) return Address_Type;
  ...
end Simulator;
```

Addresses are represented as unsigned integers (a modular type), whereas offsets are signed integers. The function Calc_Address aims to add an offset to a base address and return an address. The offset could be negative.

Naïvely we might hope to write

```
function Calc_Address(Base_Add: Address_Type;
            Offset: Offset_Type) return Address_Type is
begin
  return Base_Add + Offset;                    -- illegal
end Calc_Address;
```

but this is plainly illegal because Base_Add and Offset are of different types.

We can try a type conversion thus

```
return Base_Add + Address_Type(Offset);
```

or perhaps, since Address_Type might have a constraint,

```
return Base_Add + Address_Type'Base(Offset);
```

but in any case the conversion is doomed to raise Constraint_Error if Offset is negative.

We then try to be clever and write

```
return Base_Add + Address_Type'Base(Offset mod
                Offset_Type'Base(Address_Type'Modulus));
```

but this raises Constraint_Error if Address_Type'Modulus > Offset_Type'Base'Last which it often will be. To see this consider for example a 32-bit machine with

```
type Offset_Type is range –(2**31) .. 2**31–1;
type Address_Type is mod 2**32;
```

in which case Address_Type'Modulus is 2**32 which is greater than Offset_Type'Base'Last which is 2**31–1.

So we try an explicit test for a negative offset

```
if Offset >= 0 then
  return Base_Add + Address_Type'Base(Offset);
else
  return Base_Add - Address_Type'Base(–Offset);
end if;
```

But if Address_Type'Base'Last < Offset_Type'Last then this will raise Constraint_Error for some values of Offset. Unlikely perhaps but this is a generic and so ought to work for all possible pairs of types.

If we attempt to overcome this then we run into problems in trying to compare these two values since they are of different types and converting one to the other can raise the Constraint_Error problem once more. One solution is to use a bigger type to do the test but this may not exist in some implementations. We could of course handle the Constraint_Error and then patch up the answer. The ruthless programmer might even think of Unchecked_Conversion but this has its own problems. And so on – 'tis a wearisome tale.

The problem is neatly overcome in Ada 2005 by the introduction of a new functional attribute

```
function S'Mod(Arg: universal_integer) return S'Base;
```

S'Mod applies to any modular subtype S and returns

```
Arg mod S'Modulus
```

In other words it converts a *universal_integer* value to the modular type using the corresponding mathematical mod operation. We can then happily write

```
function Calc_Address(Base_Add: Address_Type;
            Offset: Offset_Type) return Address_Type is
begin
  return Base_Add + Address_Type'Mod(Offset);
end Calc_Address;
```

and this always works.

The next topic in the numerics area concerns rounding. One of the problems in the design of any programming language is getting the correct balance between performance and portability. This is particularly evident with numeric types where the computer has to implement only a crude approximation to the mathematician's integers and reals. The best performance is achieved by using types and operations that correspond exactly to the hardware. On the other hand, perfect portability requires using types with precisely identical characteristics on all implementations.

An interesting example of this problem arises with conversions from a floating point type to an integer type when the floating type value is midway between two integer values.

In Ada 83 the rounding in the midway case was not specified. This upset some people and so Ada 95 went the other way and decreed that such rounding was always away from zero. As well as this rule for conversion to integer types, Ada 95 also introduced a functional attribute to round a floating value. Thus for a subtype S of a floating point type T we have

```
function S'Rounding(X: T) return T;
```

This returns the nearest integral value and for midway values rounds away from zero.

Ada 95 also gives a bit more control for the benefit of the statistically minded by introducing

```
function S'Unbiased_Rounding(X: T) return T;
```

This returns the nearest integral value and for midway values rounds to the even value.

However, there are many applications where we don't care which value we get but would prefer the code to be fast. Implementers have reported problems with the elementary

functions where table look-up is used to select a particular polynomial expansion. Either polynomial will do just as well when at the midpoint of some range. However on some popular hardware such as the Pentium, doing the exact rounding required by Ada 95 just wastes time and the resulting function is perhaps 20% slower. This is serious in any comparison with C.

This problem is overcome in Ada 2005 by the introduction of a further attribute

```
function S'Machine_Rounding(X: T) return T;
```

This does not specify which of the adjacent integral values is returned if X lies midway. Note that it is not implementation defined but deliberately unspecified. This should discourage users from depending upon the behaviour on a particular implementation and thus writing non-portable code.

Zerophiles will be pleased to note that if S'Signed_Zeros is true and the answer is zero then it has the same sign as X.

It should be noted that Machine_Rounding, like the other rounding functions, returns a value of the floating point type and not perhaps universal integer as might be expected. So it will typically be used in a context such as

```
X: Some_Float;
Index: Integer;
...
Index := Integer(Some_Float'Machine_Rounding(X));
...                    -- now use Index for table look-up
```

Implementations are urged to detect this case in order to generate fast code.

The third improvement to the core language in the numerics area concerns fixed point arithmetic. This is a topic that concerns few people but those who do use it probably feel passionately about it.

The trouble with floating point is that it is rather machine dependent and of course integers are just integers. Many application areas have used some form of scaled integers for many decades and the Ada fixed point facility is important in certain applications where rigorous error analysis is desirable.

The model of fixed point was changed somewhat from Ada 83 to Ada 95. One change was that the concepts of model and safe numbers were replaced by a much simpler model just based on the multiples of the number *small*. Thus consider the type

```
Del: constant := 2.0**(–15);
type Frac is delta Del range –1.0 .. 1.0;
```

In Ada 83 small was defined to be the largest power of 2 not greater than Del, and in this case is indeed 2.0**(–15). But in Ada 95, small can be chosen by the implementation to be any power of 2 not greater than Del provided of course that the full range of values is covered. In both languages a representation clause can be used to specify small and it need not be a power of 2.

A more far reaching change introduced in Ada 95 concerns the introduction of operations on the type *universal_fixed* and type conversion.

A minor problem in Ada 83 was that explicit type conversion was required in places where it might have been considered quite unnecessary. Thus supposing we have variables F, G, H of the above type Frac, then in Ada 83 we could not write

```
H := F * G;                -- illegal in Ada 83
```

but had to use an explicit conversion

```
H := Frac(F * G);          -- legal in Ada 83
```

In Ada 83, multiplication was defined between any two fixed point types and produced a result of the type *universal_fixed* and an explicit conversion was then required to convert this to the type Frac.

This explicit conversion was considered to be a nuisance so the rule was changed in Ada 95 to say that multiplication was only defined between *universal_fixed* operands and delivered a *universal_fixed* result. Implicit conversions were then allowed for both operands and result provided the type resolution rules identified no ambiguity. So since the expected type was Frac and no other interpretation was possible, the implicit conversion was allowed and so in Ada 95 we can simply write

```
H := F * G;                -- legal in Ada 95
```

Similar rules apply to division in both Ada 83 and Ada 95.

Note however that

```
F := F * G * H;            -- illegal
```

is illegal in Ada 95 because of the existence of the pervasive type Duration defined in Standard. The intermediate result could be either Frac or Duration. So we have to add an explicit conversion somewhere.

One of the great things about Ada is the ability to define your own operations. And in Ada 83 many programmers wrote their own arithmetic operations for fixed point. These might be saturation operations in which the result is not allowed to overflow but just takes the extreme implemented value. Such operations often match the behaviour of some external device. So we might declare

```
function "*"(Left, Right: Frac) return Frac is
begin
  return Standard."*"(Left, Right);
exception
  when Constraint_Error =>
    if (Left>0.0 and Right>0.0) or
                          (Left<0.0 and Right<0.0) then
      return Frac'Last;
    else
      return Frac'First;
    end if;
end "*";
```

and similar functions for addition, subtraction, and division (taking due care over division by zero and so on). This

works fine in Ada 83 and all calculations can now use the new operations rather than the predefined ones in a natural manner.

Note however that

```
H := Frac(F * G);
```

is now ambiguous in Ada 83 since both our own new "*" and the predefined "*" are possible interpretations. However, if we simply write the more natural

```
H := F * G;
```

then there is no ambiguity. So we can program in Ada 83 without the explicit conversion.

However, in Ada 95 we run into a problem when we introduce our own operations since

```
H := F * G;
```

is ambiguous because both the predefined operation and our own operation are possible interpretations of "*" in this context. There is no cure for this in Ada 95 except for changing our own multiplying operations to be procedures with identifiers such as mul and div. This is a very tedious chore and prone to errors.

It has been reported that because of this difficulty many projects using fixed point have not moved from Ada 83 to Ada 95.

This problem is solved in Ada 2005 by changing the name resolution rules to forbid the use of the predefined multiplication (division) operation if there is a user-defined primitive multiplication (division) operation for either operand type unless there is an explicit conversion on the result or we write Standard."*" (or Standard."/").

This means that when there is no conversion as in

```
H := F * G;
```

then the predefined operation cannot apply if there is a primitive user-defined "*" for one of the operand types. So the ambiguity is resolved. Note that if there is a conversion then it is still ambiguous as in Ada 83.

If we absolutely need to have a conversion then we can always use a qualification as well or just instead. Thus we can write

```
F := Frac'(F * G) * H;
```

and this will unambiguously use our own operation.

On the other hand if we truly want to use the predefined operation then we can always write

```
H := Standard."*"(F, G);
```

Another example might be instructive. Suppose we declare three types TL, TA, TV representing lengths, areas, and volumes. We use centimetres as the basic unit with an accuracy of 0.1 cm together with corresponding consistent units and accuracies for areas and volumes. We might declare

```
type TL is delta 0.1 range –100.0 .. 100.0;
type TA is delta 0.01 range –10_000.0 .. 10_000.0;
type TV is delta 0.001 range –1000_000.0 .. 1000_000.0;
for TL'Small use TL'Delta;
for TA'Small use TA'Delta;
for TV'Small use TV'Delta;

function "*"(Left: TL; Right: TL) return TA;
function "*"(Left: TL; Right: TA) return TV;
function "*"(Left: TA Right: TL) return TV;
function "/"(Left: TV; Right: TL) return TA;
function "/"(Left: TV; Right: TA) return TL;
function "/"(Left: TA; Right: TL) return TL;

XL, YL: TL;
XA, YA: TA;
XV, YV: TV;
```

These types have an explicit small equal to their delta and are such that no scaling is required to implement the appropriate multiplication and division operations. This absence of scaling is not really relevant to the discussion below but simply illustrates why we might have several fixed point types and operations between them.

Note that all three types have primitive user-defined multiplication and division operations even though in the case of multiplication, TV only appears as a result type. Thus the predefined multiplication or division with any of these types as operands can only be considered if the result has a type conversion.

As a consequence the following are legal

```
XV := XL * XA;          -- OK, volume = length × area
XL := XV / XA;          -- OK, length = volume ÷ area
```

but the following are not because they do not match the user-defined operations

```
XV := XL * XL;          -- no, volume ≠ length × length
XV := XL / XA;          -- no, volume ≠ length ÷ area
XL := XL * XL;          -- no, length ≠ length × length
```

But if we insist on multiplying two lengths together then we can use an explicit conversion thus

```
XL := TL(XL * XL);      -- legal, predefined operation
```

and this uses the predefined operation.

If we need to multiply three lengths to get a volume without storing an intermediate area then we can write

```
XV := XL * XL * XL;
```

and this is unambiguous since there are no explicit conversions and so the only relevant operations are those we have declared.

It is interesting to compare this with the corresponding solution using floating point where we would need to make the unwanted predefined operations abstract as discussed in an earlier paper.

It is hoped that the reader has not found this discussion to be too protracted. Although fixed point is a somewhat specialized area, it is important to those who find it useful

and it is good to know that the problems with Ada 95 have been resolved.

There are a number of other improvements in the numerics area but these concern the Numerics annex and so will be discussed in a later paper.

4 Pragmas and Restrictions

Ada 2005 introduces a number of new pragmas and Restrictions identifiers. Many of these were described in the previous paper when discussing tasking and the Real-Time and High Integrity annexes. For convenience here is a complete list giving the annex if appropriate.

The new pragmas are

Assert	
Assertion_Policy	
Detect_Blocking	High-Integrity
No_Return	
Preelaborable_Initialization	
Profile	Real-Time
Relative_Deadline	Real-Time
Unchecked_Union	Interface
Unsuppress	

The new Restrictions identifiers are

Max_Entry_Queue_Length	Real-Time
No_Dependence	
No_Dynamic_Attachment	Real-Time
No_Implementation_Attributes	
No_Implementation_Pragmas	
No_Local_Protected_Objects	Real-Time
No_Obsolescent_Features	
No_Protected_Type_Allocators	Real-Time
No_Relative_Delay	Real-Time
No_Requeue_Statements	Real-Time
No_Select_Statements	Real-Time
No_Synchronous_Control	Real-Time
No_Task_Termination	Real-Time
Simple_Barriers	Real-Time

We will now discuss in detail the pragmas and Restrictions identifiers in the core language and so not discussed in the previous paper.

First there is the pragma Assert and the associated pragma Assertion_Policy. Their syntax is as follows

```
pragma Assert([Check =>] boolean_expression
              [, [Message =>] string_expression]);

pragma Assertion_Policy(policy_identifier);
```

The first parameter of Assert is thus a boolean expression and the second (and optional) parameter is a string. Remember that when we write Boolean we mean of the predefined type whereas boolean includes any type derived from Boolean as well.

The parameter of Assertion_Policy is an identifier which controls the behaviour of the pragma Assert. Two policies are defined by the language, namely, Check and Ignore. Further policies may be defined by the implementation.

There is also a package Ada.Assertions thus

```
package Ada.Assertions is
  pragma Pure(Assertions);

  Assertion_Error: exception;

  procedure Assert(Check: in Boolean);
  procedure Assert(Check: in Boolean;
                                    Message: in String);
end Ada.Assertions;
```

The pragma Assert can be used wherever a declaration or statement is allowed. Thus it might occur in a list of declarations such as

```
N: constant Integer := ... ;
pragma Assert(N > 1);
A: Real_Matrix(1 .. N, 1 .. N);
EV: Real_Vector(1 .. N);
```

and in a sequence of statements such as

```
pragma Assert(Transpose(A) = A, "A not symmetric");
EV := Eigenvalues(A);
```

If the policy set by Assertion_Policy is Check then the above pragmas are equivalent to

```
if not N > 1 then
  raise Assertion_Error;
end if;
```

and

```
if not Transpose(A) = A then
  raise Assertion_Error with "A not symmetric";
end if;
```

Remember from Section 2 that a raise statement without any explicit message is not the same as one with an explicit null message. In the former case a subsequent call of Exception_Message returns implementation defined information whereas in the latter case it returns a null string. This same behaviour thus occurs with the Assert pragma as well – providing no message is not the same as providing a null message.

If the policy set by Assertion_Policy is Ignore then the Assert pragma is ignored at execution time – but of course the syntax of the parameters is checked during compilation.

The two procedures Assert in the package Ada.Assertions have an identical effect to the corresponding Assert pragmas except that their behaviour does not depend upon the assertion policy. Thus the call

```
Assert(Some_Test);
```

is always equivalent to

```
if not Some_Test then
  raise Assertion_Error;
end if;
```

In other words we could define the behaviour of

```
pragma Assert(Some_Test);
```

as equivalent to

```
if policy_identifier = Check then
   Assert(Some_Test);          -- call of procedure Assert
end if;
```

Note again that there are two procedures Assert, one with and one without the message parameter. These correspond to raise statements with and without an explicit message.

The pragma Assertion_Policy is a configuration pragma and controls the behaviour of Assert throughout the units to which it applies. It is thus possible for different policies to be in effect in different parts of a partition.

An implementation could define other policies such as Assume which might mean that the compiler is free to do optimizations based on the assumption that the boolean expressions are true although there would be no code to check that they were true. Careless use of such a policy could lead to erroneous behaviour.

There was some concern that pragmas such as Assert might be misunderstood to imply that static analysis was being carried out. Thus in the SPARK language [2], the annotation

```
--# assert N /= 0
```

is indeed a static assertion and the appropriate tools can be used to verify this.

However, other languages such as Eiffel have used **assert** in a dynamic manner as now introduced into Ada 2005 and, moreover, many implementations of Ada have already provided a pragma Assert so it is expected that there will be no confusion with its incorporation into the standard.

Another pragma with a related flavour is No_Return. This can be applied to a procedure (not to a function) and asserts that the procedure never returns in the normal sense. Control can leave the procedure only by the propagation of an exception or it might loop forever (which is common among certain real-time programs). The syntax is

```
pragma No_Return(procedure_local_name
                                 {, procedure_local_name});
```

Thus we might have a procedure Fatal_Error which outputs some message and then propagates an exception which can be handled in the main subprogram. For example

```
procedure Fatal_Error(Msg: in String) is
   pragma No_Return(Fatal_Error);
begin
   Put_Line(Msg);
   ...                          -- other last wishes
   raise Death;
end Fatal_Error;
...
procedure Main is
   ...
   ...
   Put_Line("Program terminated successfully");
exception
   when Death =>
      Put_Line("Program terminated: known error");
   when others =>
      Put_Line("Program terminated: unknown error");
end Main;
```

There are two consequences of supplying a pragma No_Return.

- The implementation checks at compile time that the procedure concerned has no explicit return statements. There is also a check at run time that it does not attempt to run into the final end – Program_Error is raised if it does as in the case of running into the end of a function.
- The implementation is able to assume that calls of the procedure do not return and so various optimizations can be made.

We might then have a call of Fatal_Error as in

```
function Pop return Symbol is
begin
   if Top = 0 then
      Fatal_Error("Stack empty");   -- never returns
   elsif
      Top := Top – 1;
      return S(Top+1);
   end if;
end Pop;
```

If No_Return applies to Fatal_Error then the compiler should not compile a jump after the call of Fatal_Error and should not produce a warning that control might run into the final end of Pop.

The pragma No_Return now applies to the predefined procedure Raise_Exception. To enable this to be possible its behaviour with Null_Id has had to be changed. In Ada 95 writing

```
Raise_Exception(Null_Id, "Nothing");
```

does nothing at all (and so does return in that case) whereas in Ada 2005 it is defined to raise Constraint_Error and so now never returns.

We could restructure the procedure Fatal_Error to use Raise_Exception thus

```
procedure Fatal_Error(Msg: in String) is
   pragma No_Return(Fatal_Error);
begin
   ...                          -- other last wishes
   Raise_Exception(Death'Identity, Msg);
end Fatal_Error;
```

Since pragma No_Return applies to Fatal_Error it is important that we also know that Raise_Exception cannot return.

The exception handler for Death in the main subprogram can now use Exception_Message to print out the message.

Remember also from Section 2 above that we can now also write

```
raise Death with Msg;
```

rather than call Raise_Exception.

The pragma No_Return is a representation pragma. If a subprogram has no distinct specification then the pragma No_Return is placed inside the body (as shown above). If a subprogram has a distinct specification then the pragma must follow the specification in the same compilation or declarative region. Thus one pragma No_Return could apply to several subprograms declared in the same package specification.

It is important that dispatching works correctly with procedures that do not return. A non-returning dispatching procedure can only be overridden by a non-returning procedure and so the overriding procedure must also have pragma No_Return thus

```
type T is tagged ...
procedure P(X: T; ... );
pragma No_Return(P);
...
type TT is new T with ...
overriding
procedure P(X: TT; ... );
pragma No_Return(P);
```

The reverse is not true of course. A procedure that does return can be overridden by one that does not.

It is possible to give a pragma No_Return for an abstract procedure, but obviously not for a null procedure. A pragma No_Return can also be given for a generic procedure. It then applies to all instances.

The next new pragma is Preelaborable_Initialization. The syntax is

```
pragma Preelaborable_Initialization(direct_name);
```

This pragma concerns the categorization of library units and is related to pragmas such as Pure and Preelaborate. It is used with a private type and promises that the full type given by the parameter will indeed have preelaborable initialization. The details of its use will be explained in the next paper.

Another new pragma is Unchecked_Union. The syntax is

```
pragma Unchecked_Union(first_subtype_local_name);
```

The parameter has to denote an unconstrained discriminated record subtype with a variant part. The purpose of the pragma is to permit interfacing to unions in C. The following example was given in the Introduction

```
type Number(Kind: Precision) is
  record
    case Kind is
      when Single_Precision =>
        SP_Value: Long_Float;
      when Multiple_Precision =>
        MP_Value_Length: Integer;
        MP_Value_First: access Long_Float;
    end case;
  end record;

pragma Unchecked_Union(Number);
```

Specifying the pragma Unchecked_Union ensures the following

- The representation of the type does not allow space for any discriminants.
- There is an implicit suppression of Discriminant_Check.
- There is an implicit **pragma** Convention(C).

The above Ada text provides a mapping of the following C union

```
union {
  double spvalue;
  struct {
    int length;
    double* first;
    } mpvalue;
} number;
```

The general idea is that the C programmer has created a type which can be used to represent a floating point number in one of two ways according to the precision required. One way is just as a double length value (a single item) and the other way is as a number of items considered juxtaposed to create a multiple precision value. This latter is represented as a structure consisting of an integer giving the number of items followed by a pointer to the first of them. These two different forms are the two alternatives of the union.

In the Ada mapping the choice of precision is governed by the discriminant Kind which is of an enumeration type as follows

```
type Precision is (Single_Precision, Multiple_Precision);
```

In the single precision case the component SP_Value of type Long_Float maps onto the C component spvalue of type double.

The multiple precision case is somewhat troublesome. The Ada component MP_Value_Length maps onto the C component length and the Ada component MP_Value_First of type **access** Long_Float maps onto the C component first of type *double.

In our Ada program we can declare a variable thus

```
X: Number(Multiple_Precision);
```

and we then obtain a value in X by calling some C subprogram. We can then declare an array and map it onto the C sequence of double length values thus

```
A: array (1 .. X.MP_Value_Length) of Long_Float;
for A'Address use X.MP_Value_First.all'Address;
pragma Import(C, A);
```

The elements of A are now the required values. Note that we don't use an Ada array in the declaration of Number because there might be problems with dope information.

The Ada type can also have a non-variant part preceding the variant part and variant parts can be nested. It may have several discriminants.

When an object of an unchecked union type is created, values must be supplied for all its discriminants even though they are not stored. This ensures that appropriate default values can be supplied and that an aggregate contains the correct components. However, since the discriminants are not stored, they cannot be read. So we can write

```
X: Number := (Single_Precision, 45.6);
Y: Number(Single_Precision);
...
Y.SP_Value := 55.7;
```

The variable Y is said to have an inferable discriminant whereas X does not. Although it is clear that playing with unchecked unions is potentially dangerous, nevertheless Ada 2005 imposes certain rules that avoid some dangers. One rule is that predefined equality can only be used on operands with inferable discriminants; Program_Error is raised otherwise. So

```
if Y = 55.8 then          -- OK

if X = 45.5 then          -- raises Program_Error

if X = Y then             -- raises Program_Error
```

It is important to be aware that unchecked union types are introduced in Ada 2005 for the sole purpose of interfacing to C programs and not for living dangerously. Thus consider

```
type T(Flag: Boolean := False) is
  record
    case Flag is
      when False =>
        F1: Float := 0.0;
      when True =>
        F2: Integer := 0;
    end case;
  end record;
pragma Unchecked_Union(T);
```

The type T can masquerade as either type Integer or Float. But we should not use unchecked union types as an alternative to unchecked conversion. Thus consider

```
X: T;                     -- Float by default
Y: Integer := X.F2;       -- erroneous
```

The object X has discriminant False by default and thus has the value zero of type Integer. In the absence of the pragma Unchecked_Union, the attempt to read X.F2 would raise Constraint_Error because of the discriminant check. The use of Unchecked_Union suppresses the discriminant check and so the assignment will occur. However, the ARM clearly says (11.5(26)) that if a check is suppressed and the corresponding error situation arises then the program is erroneous.

However, assigning a Float value to an Integer object using Unchecked_Conversion is not erroneous providing certain conditions hold such as that Float'Size = Integer'Size.

The final pragma to be considered is Unsuppress. Its syntax is

```
pragma Unsuppress(identifier);
```

The identifier is that of a check or perhaps All_Checks. The pragma Unsuppress is essentially the opposite of the existing pragma Suppress and can be used in the same places with similar scoping rules.

Remember that pragma Suppress gives an implementation the permission to omit the checks but it does not require that the checks be omitted (they might be done by hardware). The pragma Unsuppress simply revokes this permission. One pragma can override the other in a nested manner. If both are given in the same region then they apply from the point where they are given and the later one thus overrides.

A likely scenario would be that Suppress applies to a large region of the program (perhaps all of it) and Unsuppress applies to a smaller region within. The reverse would also be possible but perhaps less likely.

Note that Unsuppress does not override the implicit Suppress of Discriminant_Check provided by the pragma Unchecked_Union just discussed.

A sensible application of Unsuppress would be in the fixed point operations mentioned in Section 3 thus

```
function "*"(Left, Right: Frac) return Frac is
  pragma Unsuppress(Overflow_Check);
begin
  return Standard."*"(Left, Right);
exception
  when Constraint_Error =>
    if (Left>0.0 and Right>0.0) or
                          (Left<0.0 and Right<0.0) then
      return Frac'Last;
    else
      return Frac'First;
    end if;
end "*";
```

The use of Unsuppress ensures that the overflow check is not suppressed even if there is a global Suppress for the whole program (or the user has switched checks off through the compiler command line). So Constraint_Error will be raised as necessary and the code will work correctly.

In Ada 95 the pragma Suppress has the syntax

```
pragma Suppress(identifier [ , [On =>] name]);  -- Ada 95
```

The second and optional parameter gives the name of the entity to which the permission applies. There was never any clear agreement on what this meant and implementations varied. Accordingly, in Ada 2005 the second parameter is banished to Annex J so that the syntax in the core language is similar to Unsuppress thus

```
pragma Suppress(identifier);                    -- Ada 2005
```

For symmetry, Annex J actually allows an obsolete On parameter for Unsuppress. It might seem curious that a feature should be born obsolescent.

A number of new Restrictions identifiers are added in Ada 2005. The first is No_Dependence whose syntax is

```
pragma Restrictions(No_Dependence => name);
```

This indicates that there is no dependence on a library unit with the given name.

The name might be that of a predefined unit but it could in fact be any unit. For example, it might be helpful to know that there is no dependence on a particular implementation-defined unit such as a package Superstring thus

```
pragma Restrictions(No_Dependence => Superstring);
```

Care needs to be taken to spell the name correctly; if we write Supperstring by mistake then the compiler will not be able to help us.

The introduction of No_Dependence means that the existing Restrictions identifier No_Asynchronous_Control is moved to Annex J since we can now write

```
pragma Restrictions(
  No_Dependence => Ada.Asynchronous_Task_Control);
```

Similarly, the identifiers No_Unchecked_Conversion and No_Unchecked_Deallocation are also moved to Annex J.

Note that the identifier No_Dynamic_Attachment which refers to the use of the subprograms in the package Ada.Interrupts cannot be treated in this way because of the child package Ada.Interrupts.Names. No dependence on Ada.Interrupts would exclude the use of the child package Names as well.

The restrictions identifier No_Dynamic_Priorities cannot be treated this way either for a rather different reason. In Ada 2005 this identifier is extended so that it also excludes the use of the attribute Priority and this would not be excluded by just saying no dependence on Ada.Dynamic_Priorities.

Two further Restrictions identifiers are introduced to encourage portability. We can write

```
pragma Restrictions(No_Implementation_Pragmas,
                    No_Implementation_Attributes);
```

These do not apply to the whole partition but only to the compilation or environment concerned. This helps us to ensure that implementation dependent areas of a program are identified.

The final new restrictions identifier similarly prevents us from inadvertently using features in Annex J thus

```
pragma Restrictions(No_Obsolescent_Features);
```

Again this does not apply to the whole partition but only to the compilation or environment concerned. (It is of course not itself defined in Annex J.)

The reader will recall that in Ada 83 the predefined packages had names such as Text_IO whereas in Ada 95 they are Ada.Text_IO and so on. In order to ease transition from Ada 83, a number of renamings were declared in Annex J such as

```
with Ada.Text_IO;
package Text_IO renames Ada.Text_IO;
```

A mild problem is that the user could write these renamings anyway and we do not want the No_Obsolescent_Features restriction to prevent this. Moreover, implementations might actually implement the renamings in Annex J by just compiling them and we don't want to force implementations to use some trickery to permit the user to do it but not the implementation. Accordingly, whether the No_Obsolescent_Features restriction applies to these renamings or not is implementation defined.

5 Generic units

There are a number of improvements in the area of generics many of which have already been outlined in earlier papers.

A first point concerns access types. The introduction of types that exclude null means that a formal access type parameter can take the form

```
generic
  ...
  type A is not null access T;
  ...
```

The actual type corresponding to A must then itself be an access type that excludes null. A similar rule applies in reverse – if the formal parameter excludes null then the actual parameter must also exclude null. If the two did not match in this respect then all sorts of difficulties could arise.

Similarly if the formal parameter is derived from an access type

```
generic
  ...
  type FA is new A;              -- A is an access type
  ...
```

then the actual type corresponding to FA must exclude null if A excludes null and vice versa. Half of this rule is automatically enforced since a type derived from a type that excludes null will automatically exclude null. But the reverse is not true as mentioned in an earlier paper when discussing access types. If A has the declaration

```
type A is access all Integer;      -- does not exclude null
```

then we can declare

```
type NA is new A;                  -- does not exclude null
type NNA is new not null A;        -- does exclude null
```

and then NA matches the formal parameter FA in the above generic but NNA does not.

There is also a change to formal derived types concerning limitedness. In line with the changes described in the paper on the object oriented model, the syntax now permits **limited** to be stated explicitly thus

```
generic
  type T is limited new LT;                    -- untagged
  type TT is limited new TLT with private;     -- tagged
```

However, this can be seen simply as a documentation aid since the actual types corresponding to T and TT must be derived from LT and TLT and so will be limited if LT and TLT are limited anyway.

Objects of anonymous access types are now also allowed as generic formal parameters so we can have

```
generic
  A: access T := null;
  AN: in out not null access T;
  F: access function (X: Float) return Float;
  FN: not null access function (X: Float) return Float;
```

If the subtype of the formal object excludes null (as in AN and FN) then the actual must also exclude null but not vice versa. This contrasts with the rule for formal access types discussed above in which case both the formal type and actual type have to exclude null or not. Note moreover that object parameters of anonymous access types can have mode **in out**.

If the subprogram profile itself has access parameters that exclude null as in

```
generic
  PN: access procedure (AN: not null access T);
```

then the actual subprogram must also have access parameters that exclude null and so on. The same rule applies to named formal subprogram parameters. If we have

```
generic
  with procedure P(AN: not null access T);
  with procedure Q(AN: access T);
```

then the actual corresponding to P must have a parameter that excludes null but the actual corresponding to Q might or might not. The rule is similar to renaming – "not null must never lie". Remember that the matching of object and subprogram generic parameters is defined in terms of renaming. Here is an example to illustrate why the asymmetry is important. Suppose we have

```
generic
  type T is private;
  with procedure P(Z: in T);
package G is
```

This can be matched by

```
type A is access ...;
procedure Q(Y: in not null A);
...
package NG is new G(T => A; P => Q);
```

Note that since the formal type T is not known to be an access type in the generic declaration, there is no mechanism for applying a null exclusion to it. Nevertheless there is no reason why the instantiation should not be permitted.

There are some other changes to existing named formal subprogram parameters. The reader will recall from the discussion on interfaces in an earlier paper that the concept of null procedures has been added in Ada 2005. A null procedure has no body but behaves as if it has a body comprising a null statement. It is now possible to use a null procedure as a possible form of default for a subprogram parameter. Thus there are now three possible forms of default as follows

```
with procedure P( ... ) is <>;              -- OK in 95
with procedure Q( ... ) is Some_Proc;       -- OK in 95
with procedure R( ... ) is null;            -- only in 2005
```

So if we have

```
generic
  type T is (<>);
  with procedure R(X: in Integer; Y: in out T) is null;
package PP ...
```

then an instantiation omitting the parameter for R such as

```
package NPP is new PP(T => Colour);
```

is equivalent to providing an actual procedure AR thus

```
procedure AR(X: in Integer; Y: in out Colour) is
begin
  null;
end AR;
```

Note that the profile of the actual procedure is conjured up to match the formal procedure.

Of course, there is no such thing as a null function and so null is not permitted as the default for a formal function.

A new kind of subprogram parameter was introduced in some detail when discussing object factory functions in the paper on the object oriented model. This is the abstract formal subprogram. The example given was the predefined generic function Generic_Dispatching_Constructor thus

```
generic
  type T (<>) is abstract tagged limited private;
  type Parameters (<>) is limited private;
  with function Constructor(Params: access Parameters)
                                    return T is abstract;
function Ada.Tags.Generic_Dispatching_Constructor
        (The_Tag: Tag; Params: access Parameters)
                                    return T'Class;
```

The formal function Constructor is an example of an abstract formal subprogram. Remember that the interpretation is that the actual function must be a dispatching operation of a tagged type uniquely identified by the profile of the formal function. The actual operation can be concrete or abstract. Formal abstract subprograms can of course be procedures as well as functions. It is important that there is exactly one controlling type in the profile.

Formal abstract subprograms can have defaults in much the same way that formal concrete subprograms can have defaults. We write

```
with procedure P(X: in out T) is abstract <>;
with function F return T is abstract Unit;
```

The first means of course that the default has to have identifier P and the second means that the default is some function Unit. It is not possible to give null as the default for an abstract parameter for various reasons. Defaults will probably be rarely used for abstract parameters.

The introduction of interfaces in Ada 2005 means that a new class of generic parameters is possible. Thus we might have

```
generic
  type F is interface;
```

The actual type could then be any interface. This is perhaps unlikely.

If we wanted to ensure that a formal interface had certain operations then we might first declare an interface A with the required operations

```
type A is interface;
procedure Op1(X: A; ... ) is abstract;
procedure N1(X: A; ... ) is null;
```

and then

```
generic
  type F is interface and A;
```

and then the actual interface must be descended from A and so have operations which match Op1 and N1.

A formal interface might specify several ancestors

```
generic
  type FAB is interface and A and B;
```

where A and B are themselves interfaces. And A and B or just some of them might themselves be further formal parameters as in

```
generic
  type A is interface;
  type FAB is interface and A and B;
```

These means that FAB must have both A and B as ancestors; it could of course have other ancestors as well.

The syntax for formal tagged types is also changed to take into account the possibility of interfaces. Thus we might have

```
generic
  type NT is new T and A and B with private;
```

in which case the actual type must be descended both from the tagged type T and the interfaces A and B. The parent type T itself might be an interface or a normal tagged type. Again some or all of T, A, and B might be earlier formal parameters. Also we can explicitly state **limited** in which case all of the ancestor types must also be limited.

An example of this sort of structure occurred when discussing printable geometric objects in the paper on the object oriented model. We had

```
generic
  type T is abstract tagged private;
package Make_Printable is
  type Printable_T is
      abstract new T and Printable with private;
  ...
end;
```

It might be that we have various interfaces all derived from Printable which serve different purposes (perhaps for different output devices, laser printer, card punch and so on). We would then want the generic package to take any of these interfaces thus

```
generic
  type T is abstract tagged private;
  type Any_Printable is interface and Printable;
package Make_Printable is
  type Printable_T is
      abstract new T and Any_Printable with private;
  ...
end;
```

A formal interface can also be marked as limited in which case the actual interface must also be limited and vice versa.

As discussed in the previous paper, interfaces can also be synchronized, task, or protected. Thus we might have

```
generic
  type T is task interface;
```

and then the actual interface must itself be a task interface. The correspondence must be exact. A formal synchronized interface can only be matched by an actual synchronized interface and so on. Remember from the discussion in the previous paper that a task interface can be composed from a synchronized interface. This flexibility does not extend to matching actual and formal generic parameters.

Another small change concerns object parameters of limited types. In Ada 95 the following is illegal

```
type LT is limited
  record
    A: Integer;
    B: Float;
  end record;                    -- a limited type

generic
  X: in LT;                      -- illegal in Ada 95
  ...
procedure P ...
```

It is illegal in Ada 95 because it is not possible to provide an actual parameter. This is because the parameter mechanism is one of initialization of the formal object parameter by the actual and this is treated as assignment and so is not permitted for limited types.

However, in Ada 2005, initialization of a limited object by an aggregate is allowed since the value is created *in situ* as discussed in an earlier paper. So an instantiation is possible thus

```
procedure Q is new P(X => (A => 1, B => 2.0), ... );
```

Remember that an initial value can also be provided by a function call and so the actual parameter could also be a function call returning a limited type.

The final improvement to the generic parameter mechanism concerns package parameters.

In Ada 95 package parameters take two forms. Given a generic package Q with formal parameters F1, F2, F3, then we can have

```
generic
  with package P is new Q(<>);
```

and then the actual package corresponding to the formal P can be any instantiation of Q. Alternatively

```
generic
  with package R is new Q(P1, P2, P3);
```

and then the actual package corresponding to R must be an instantiation of Q with the specified actual parameters P1, P2, P3.

As mentioned in the Introduction, a simple example of the use of these two forms occurs with the package Generic_Complex_Arrays which takes instantiations of Generic_Real_Arrays and Generic_Complex_Types which in turn both have the underlying floating type as their single parameter. It is vital that both packages use the same floating point type and this is assured by writing

```
generic
  with package Real_Arrays is
                          new Generic_Real_Arrays(<>);
  with package Complex_Types is
       new Generic_Complex_Types(Real_Arrays.Real);
package Generic_Complex_Arrays is ...
```

However, the mechanism does not work very well when several parameters are involved as will now be illustrated with some examples.

The first example concerns using the new container library which will be discussed in some detail in the next paper. There are generic packages such as

```
generic
  type Index_Type is range <>;
  type Element_Type is private:
  with function "=" (Left, Right: Element_Type )
                                  return Boolean is <>;
package Ada.Containers.Vectors is ...
```

and

```
generic
  type Key_Type is private;
  type Element_Type is private:
  with function Hash(Key: Key_Type) return Hash_Type;
  with function Equivalent_Keys(Left, Right: Key_Type)
                                  return Boolean;
  with function "=" (Left, Right: Element_Type )
                                  return Boolean is <>;
package Ada.Containers.Hashed_Maps is ...
```

We might wish to pass instantiations of both of these to some other package with the proviso that both were instantiated with the same Element_Type. Otherwise the parameters can be unrelated.

It would be natural to make the vector package the first parameter and give it the (<>) form. But we then find that in Ada 95 we have to repeat all the parameters other than Element_Type for the maps package. So we have

```
with ... ; use Ada.Containers;
generic
  with package V is new Vectors(<>);
  type Key_Type is private;
  with function Hash(Key: Key_Type) return Hash_Type;
  with function Equivalent_Keys(Left, Right: Key_Type)
                                  return Boolean;
  with function "=" (Left, Right: Element_Type )
                                  return Boolean is <>;
  with package HM is new Hashed_Maps(
                Key_Type => Key_Type,
                Element_Type => V.Element_Type,
                Hash => Hash,
                Equivalent_Keys => Equivalent_Keys,
                "=" => "=");
package HMV is ...
```

This is a nuisance since when we instantiate HMV we have to provide all the parameters required by Hashed_Maps even though we must already have instantiated it elsewhere in the program. Suppose that instantiation was

```
package My_Hashed_Map is new Hashed_Maps
                (My_Key, Integer, Hash_It, Equiv, "=");
```

and suppose also that we have instantiated Vectors

```
package My_Vectors is new Vectors(Index, Integer, "=");
```

Now when we come to instantiate HMV we have to write

```
package My_HMV is new HMV(My_Vectors,
      My_Key, Hash_It, Equiv, "=", My_Hashed_Maps);
```

This is very annoying. Not only do we have to repeat all the auxiliary parameters of Hashed_Maps but the situation regarding Vectors and Hashed_Maps is artificially made asymmetric. (Life would have been a bit easier if we had made Hashed_Maps the first package parameter but that just illustrates the asymmetry.) Of course we could more or less overcome the asymmetry by passing all the parameters of Vectors as well but then HMV would have even more parameters. This rather defeats the point of package parameters which were introduced into Ada 95 in order to avoid the huge parameter lists that had occurred in Ada 83.

Ada 2005 overcomes this problem by permitting just some of the actual parameters to be specified. Any omitted parameters are indicated using the <> notation thus

```
generic
  with package S is new Q(P1, F2 => <>, F3 => <>);
```

In this case the actual package corresponding to S can be any package which is an instantiation of Q where the first

actual parameter is P1 but the other two parameters are left unspecified. We can also abbreviate this to

```
generic
  with package S is new Q(P1, others => <>);
```

Note that the <> notation can only be used with named parameters and also that (<>) is now considered to be a shorthand for (**others** => <>).

As another example

```
generic
  with package S is
                    new Q(F1 => <>, F2 => P2, F3 => <>);
```

means that the actual package corresponding to S can be any package which is an instantiation of Q where the second actual parameter is P2 but the other two parameters are left unspecified. This can be abbreviated to

```
generic
  with package S is new Q(F2 => P2, others => <>);
```

Using this new notation, the package HMV can now simply be written as

```
with ... ; use Ada.Containers;
generic
  with package V is new Vectors(<>);
  with package HM is new Hashed_Maps
     (Element_Type => V.Element_Type, others => <>);
package HMV is ...
```

and our instantiation of HMV becomes simply

```
package My_HMV is
              new HMV(My_Vectors, My_Hashed_Maps);
```

Some variations on this example are obviously possible. For example it is likely that the instantiation of Hashed_Maps must use the same definition of equality for the type Element_Type as Vectors. We can ensure this by writing

```
with ... ; use Ada.Containers;
generic
  with package V is new Vectors(<>);
  with package HM is new Hashed_Maps
     (Element_Type => V.Element_Type, "=" => V."=",
                                        others => <>);
package HMV is ...
```

Other examples might arise in the numerics area. Suppose we have two independently written generic packages Do_This and Do_That which both have a floating point type parameter and several other parameters as well. For example

```
generic
  type Real is digits <>;
  Accuracy: in Real;
  type Index is range <>;
  Max_Trials: in Index;
package Do_This is ...
```

```
generic
  type Floating is digits <>;
  Bounds: in Floating;
  Iterations: in Integer;
  Repeat: in Boolean;
package Do_That is ...
```

(This is typical of much numerical stuff. Authors are cautious and unable to make firm decisions about many aspects of their algorithms and therefore pass the buck back to the user in the form of a turgid list of auxiliary parameters.)

We now wish to write a package Super_Solver which takes instantiations of both Do_This and Do_That with the requirement that the floating type used for the instantiation is the same in each case but otherwise the parameters are unrelated. In Ada 95 we are again forced to repeat one set of parameters thus

```
generic
  with package This is new Do_This(<>);
  S_Bounds: in This.Real;
  S_Iterations: in Integer;
  S_Repeat: in Boolean;
  with package That is new Do_That(This.Real,
                  S_Bounds, S_Iterations, S_Repeat);
package Super_Solver is ...
```

And when we come to instantiate Super_Solver we have to provide all the auxiliary parameters required by Do_That even though we must already have instantiated it elsewhere in the program. Suppose the instantiation was

```
package That_One is new Do_That(Float, 0.01, 7, False);
```

and suppose also that we have instantiated Do_This

```
package This_One is new Do_This( ... );
```

Now when we instantiate Super_Solver we have to write

```
package SS is
new Super_Solver(This_One, 0.01, 7, False, That_One);
```

Just as with HMV we have all these duplicated parameters and an artificial asymmetry between This and That.

In Ada 2005 the package Super_Solver can be written as

```
generic
  with package This is new Do_This(<>);
  with package That is new Do_That(This.Real,
                                   others => <>);
package Super_Solver is ...
```

and the instantiation of Super_Solver becomes simply

```
package SS is new Super_Solver(This_One, That_One);
```

Other examples occur with signature packages. Remember that a signature package is one without a specification. It can be used to ensure that a group of entities are related in the correct way and an instantiation can then be used to identify the group as a whole. A trivial example might be

```
generic
  type Index is (<>);
```

```
   type item is private;
   type Vec is array (Index range <>) of Item;
package General_Vector is end;
```

An instantiation of General_Vector just asserts that the three types concerned have the appropriate relationship. Thus we might have

```
type My_Array is array (Integer range <>) of Float;
```

and then

```
package Vector is
      new General_Vector(Integer, Float, My_Array);
```

The package General_Vector could then be used as a parameter of other packages thereby reducing the number of parameters.

Another example might be the signature of a package for manipulating sets. Thus

```
generic
   type Element is private;
   type Set is private;
   with function Empty return Set;
   with function Unit(E: Element) return Set;
   with function Union(S, T: Set) return Set;
   with function Intersection(S, T: Set) return Set;
   ...
package Set_Signature is end;
```

We might then have some other generic package which takes an instantiation of this set signature. However, it is likely that we would need to specify the type of the elements but possibly not the set type and certainly not all the operations. So typically we would have

```
generic
   type My_Element is private;
   with package Sets is new Set_Signature
            (Element => My_Element, others => <>);
```

An example of this technique occurred when considering the possibility of including a system of units facility within Ada 2005. Although it was considered not appropriate to include it, the use of signature packages was almost essential to make the mechanism usable. The interested reader should consult AI-324.

We conclude by noting a small change to the syntax of a subprogram instantiation in that an overriding indicator can be supplied as mentioned in Section 7 of the paper on the object oriented model. Thus (in appropriate circumstances) we can write

```
overriding
procedure This is new That( ... );
```

This means that the instantiation must be an overriding operation for some type.

References

[5] ISO/IEC JTC1/SC22/WG9 N412 (2002) *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of the Amendment*.

[2] J. G. P. Barnes (2003) *High Integrity Software – The SPARK Approach to Safety and Security*, Addison-Wesley.

Ada Bug Finder

Alan Marriott and Urs Maurer

White Elephant GmbH, Postfach 327, CH-8450 Andelfingen, Switzerland; email: ada@white-elephant.ch

Abstract

In the context of this paper, we consider bug patterns to be sections of code that whilst syntactically correct are unlikely to be what the author intended.

Everyone, even the most erudite programmers, make dumb mistakes, often as a result of a particularly inept piece of cut and paste editing or sometimes simply by typing the exact opposite of what was meant.

Our experience has shown that even the most blatantly incorrect code can make its way into production code!

In many situations compilers could have detected the bug patterns. However it seems that the current generation of Ada compilers is content if the programmer writes legal Ada syntax. Determining whether this code is meaningful or not seems to beyond their remit.

As a consequence we have written a bug finder tool which, using static code analysis, attempts to detect code that is either obviously incorrect, is in some way questionable or is so badly written that the tool itself cannot make sense of it and is therefore worthy of further analysis.

This paper describes the tool, the bug patterns it employs and an evaluation of the results of applying the tool over several large Ada code bases.

Keywords: utility, bug finder, Ada.

1 Introduction

In the autumn of 2004, we were fortunate to attend the Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) 2004 conference in Vancouver, BC, Canada. As part of the OOPSLA Onward! Track, David Hovemeyer and William Pugh presented their paper entitled "Finding Bugs is Easy [1]"

Their paper and the presentation they made at the conference have been the basis and inspiration for the work described in this paper. Hovemeyer & Pugh's work concentrated exclusively on Java. We merely extended and adapted the idea for Ada.

Their basic premise is that many simple and obvious bugs slip through testing and end up in production code and that with a little bit of effort these bugs can be automatically found.

It is their belief that bugs in production code are not normally found because either the user does not notice the symptom of the bug, has no means to report the bug to the developers or cannot reproduce the situation that caused the bug.

Their idea is that if you detected a bug in some code you are working on, you should examine how one could look for other occurrences of the same bug and then try to determine whether this search could be somehow automated.

If a pattern can be established by which the bug can be automatically discovered then this mechanism should be incorporated into some form of tool and the tool used to search actively for the bug in as much source code as possible.

This is to say not just in the current module or project, but also in other projects, libraries and as much open source that is available.

Because their paper concentrated on Java and Java specific problems most of the bug patterns described by Hovemeyer and Pugh were not applicable. Therefore part of our work has been to develop bug patterns that are specific to Ada.

2 The Ada Bug Finder Utility

Our Ada Bug Finder tool is an interactive Windows® based program written exclusively in Ada 95.

Although not open source, the executable is available for download from our web site www.white-elephant.ch. We would like actively to encourage everyone to try it out on all available Ada source code and to report back to us with statistics regarding how many bugs the tool found and how many of these were serious.

We would also be interested in any feedback concerning how the utility might be enhanced either by suggesting new bug patterns or citing occasions where an unnecessarily large number of false positives were incurred.

2.1 Overview

The Ada Bug Finder utility takes the name of a directory as its only input. When commanded the utility searches all the Ada files contained in the specified directory and all its subdirectories.

Ada package specifications and implementations are assumed to be in pairs of files, either

- Both files have the same filename but different file extensions. The package specification having the file extension ads and the implementation the extension adb

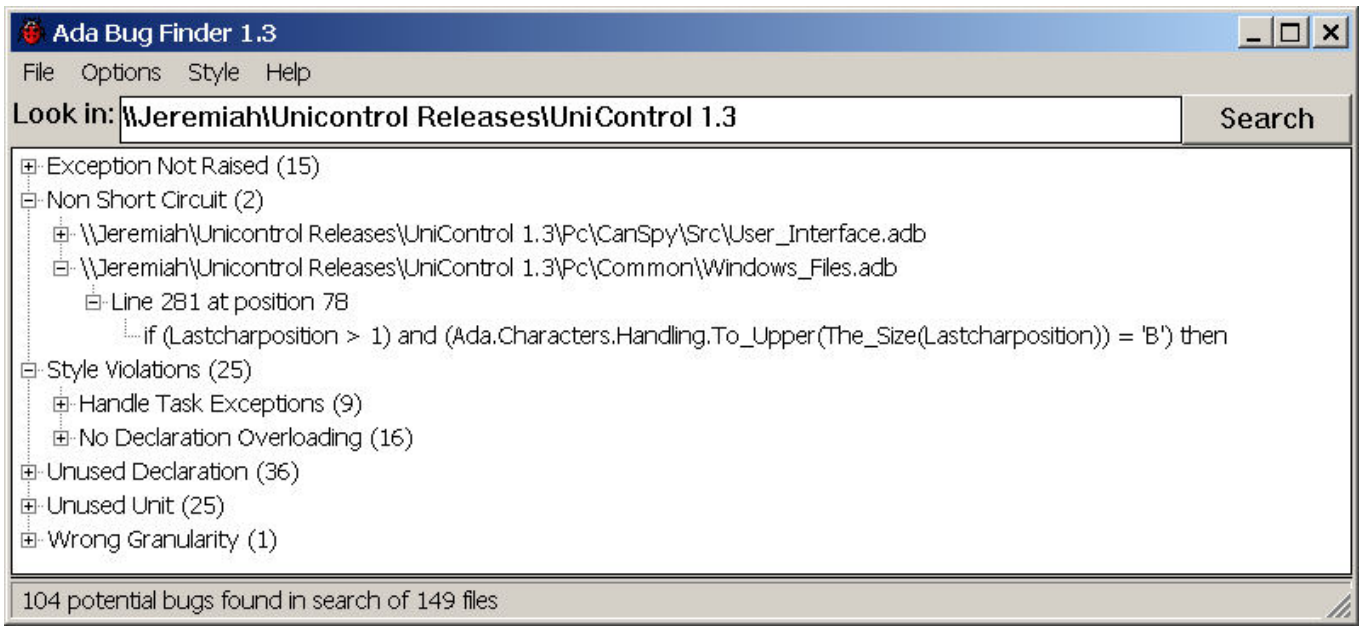

Figure 1 Screenshot of the Ada Bug Finder

For example: Test.ads and Test.adb

- Both files have the file extension ada and share the same filename up until the final character. An additional underscore signifies that the file contains the package specification.

  For example: Test_.ada and Test.ada

The program supports two options.

- Gnat extensions

  If enabled instructs the syntax parser to accept the Gnat implementation defined attributes.

- Preparation phase.

  If enabled causes the utility to process the files twice. The first pass gathers additional information that can be used to reduce the number of false positives at the expense of speed.

Results of the search are displayed in a tree view. These may be saved as either text or as a comma delimited file suitable for further processing by utilities such as Microsoft Excel.

2.2 Bugs vs. Style

Hovemeyer and Pugh consider the primary purpose of style rules is to make it easier for developers to understand each other's code and consequently they should not be included in a bug-finding tool.

Whilst we agree that this is true with respect to many style rules, we believe that some style rules have been introduced with the specific intention of prohibiting language features that are considered to encourage unsafe programming practices. Other style rules have been introduced to facilitate easier debugging.

In both these cases the enforcement of style rules could directly affect the software reliability. For this reason our utility also offers the optional detection of various style rule violations.

2.3 False Positives

Unfortunately, the utility doesn't always get it right! Occasionally the utility will highlight a segment of code as being a bug when, in fact, it is perfectly correct.

A goal of the utility is to reduce the number of these false positives to a minimum without making the pattern unnecessarily complex and unduly expensive to implement. There is a trade off, therefore, between complexity and the number of false positives the pattern might generate.

Within reason, we would rather have a few false positives than the utility missing an actual bug or making the pattern so complex that it becomes no longer reasonable to implement.

2.4 Code Marking

Given that there will be the occasional false positive, we decided that there should be a mechanism whereby the utility could be instructed to ignore a specific pattern on a particular line.

The mechanism we chose is to place a special comment at the end of the line on which the utility detects the bug. The special comment starts with a *greater than* symbol (>) followed by a two or three character bug abbreviation code terminated by a colon.

Example:

```
C_False : constant Integer := 0; --> UD: Completeness
```

In the above example the line is flagged against unused declarations (UD). The Ada Bug Finder utility will not report any unused declarations declared on this line.

3 Ada Bug Patterns

Version 1.3 of the Ada Bug Finder utility recognises eight Ada bug patterns.

3.1 Illogical Operator Rename (IOR)

In Ada 83, where there is no *use type* clause, operators are often renamed to avoid the use of prefixed notation in environments where the *use* clause is expressly forbidden.

Clumsy cut and paste editing might result in renaming an operator to be something totally different. The compiler allows this, although it is highly unlikely to be what the author intended.

Example:

```
function "<"(Left, Right : Xt.Widget)
            return Boolean renames Xt."=";
```

3.2 Code Not Reachable (CNR)

Statements after an unconditional raise, return or exit will never be executed.

Note: The Gnat compiler 3.15p checks for this pattern however both the Aonix and HP compilers do not.

Example:

```
procedure Cnr is
begin
  loop
    exit;
    Io.Put_Line ("Never written!");
  end loop;
  return;
  Io.Put_Line ("Will never get written!");
end Cnr;
```

3.3 Null Pointer (NP)

This pattern looks for occasions of a pointer being dereferenced whilst it is known to be null. Typically, this occurs in the body of an *if* statement that has previously tested the pointer explicitly for null.

Example:

```
if The_String = null then
  Io.Put_Line (The_String.all);
end if;
```

3.4 Non Short Circuit (NSC)

Essentially, testing for a condition and then, in the same expression, using the result of that condition normally requires that the programmer use the "and then" or the "or else" construct rather than simply "and" or "or".

Example:

```
Result := (The_String = null) or
          (The_String.all = "Hello");

Result := (Index <= Numbers'last) and
          (Numbers (Index) = 42);
```

3.5 Wrong Granularity (WG)

Ada's *'Size* attribute returns the size of the object in bits whereas storage allocation and most interfaces expect object sizes to be supplied in bytes.

Consequently it is very unusual for *'Size* to be used outside of an expression. These occurrences are likely to be bugs and therefore warrant further scrutiny.

Example:

```
Read (Buffer      => Buffer'address
      Max_Size    => Buffer'size
      Amount_Read => The_Size);
```

3.6 Unused Declarations

If something is declared but never used, it might simply be because it is not required. Its presence might cause the compiler more work, it might make the program bigger and it might possibly make the code less understandable.

Whilst all these symptoms are undoubtedly undesirable they are not actually bugs.

However another reason that a declared object may never be referenced is because something else is being referenced in place of it. These occurrences are bugs and it the aim of the next three patterns is to identify them.

3.6.1 Unused Declaration (UD)

A constant or variable is declared but never used. Note, however, that this might be deliberate. The initialization of controlled objects or the default initialization may have an effect that is actually required.

3.6.2 Exception Not Raised (ENR)

An exception is declared, perhaps even handled, but is never raised.

3.6.3 Unused Unit (UU)

A package is imported but never used, or a procedure, function or package is defined but neither exported nor used locally.

3.7 Syntax Error (SE)

This isn't really a bug pattern per se. Unfortunately, some of the code placed in open source libraries doesn't actually compile! Our utility reports syntax error if the code it is analysing appears to be invalid Ada.

4 Style Rule Checking

Our utility optionally checks six style rules. Each style rule can be individually enabled or disabled.

The paragraph references within parentheses refer to the Ada 95 Quality and Style Guide [2].

4.1 HTE - Handle Task Exceptions (6.3.4)

A task will terminate if an exception is raised within it, for which there is no exception handler. In such cases, the exception is not propagated outside of the task (unless it occurs during a rendezvous). The task simply dies with no notification to other tasks in the program. This makes debugging these tasks especially difficult and so we have

implemented a style rule that checks that every task has an exception handler at its outermost level that includes a *when others* statement.

4.2 NDO - No Declaration Overloading

Prohibits declarations that have the same name as a declaration currently in scope. We believe that it is poor programming style to occlude a declaration deliberately.

4.3 NGS - No Goto Statements (5.6.7)

Prohibits the use of the *goto* statement as this is considered an unstructured change in the control flow. In Ada, the label does not require an indicator of where the corresponding *goto* statements are. Many believe that this renders the code unreadable.

4.4 NPUC - No Package Use Clause (5.7.1)

Prohibits the use of the *use* clause and thereby forces external names to be always fully qualified. To provide visibility to operators use the *use type* clause.

4.5 NVIS - No Variable in Specification

Prohibits the declaration of variables in package specifications.

4.6 CNP – Code Not Portable

In Ada 83 identifiers may only contain ASCII alphanumeric characters. However some compilers fail to enforce this restriction. Although Ada-95 allows identifiers to be constructed from any alphanumeric from row 00 of the ISO 10646 BMP, effectively ISO 8859-1 (Latin-1), using characters outside of the ASCII character range may lead to portability problems.

4.7 Superfluous Code Mark

If an Ada Bug Finder code mark (>xx:) is used to suppress the reporting of a particular bug but the line in question doesn't actually produce the bug in question then something is probably wrong. It is bad style to suppress warnings unnecessarily.

5 Other Patterns (to be implemented)

5.1 Division by Zero

This pattern looks for the situation when an identifier is explicitly compared with zero and then used as the right operand of one of the operators /, *rem* and *mod*

Example:

```
if Index = 0 then
  Result := (42 / Index) > 10;
end if;
```

5.2 Raise after Assignment

Leaving a procedure abnormally nullifies any assignment to in-out or out parameters.

Example:

```
procedure Raa (The_Number : in out Natural) is
begin
 The_Number := The_Number + 1;
 raise Failed;
end Raa;
```

5.3 Redundant Comparison to null

If a null pointer check is made after code has already dereferenced the pointer, the comparison is redundant.

Either the comparison is made too late or is superfluous because the condition is known never to arise.

Example:

```
procedure Rcn is
begin
  Ada.Text_Io.Put_Line (The_String.all);
  if The_String /= null then
    Ada.Text_Io.Put_Line (The_String.all);
  end if;
end Rcn;
```

5.4 Symmetrical Comparison

If the left and right sides of a comparison are identical then this is probably a cut and paste error as it obviously makes no sense!

Example:

```
if Table (Index) = Table (Index) then
```

Sources	Files	CNR	ENR	IOR	NSC	NP	SE	UD	UU	WG	Style
UniControl 1.3	149		15		2			36	35	1	25
ILTIS 3622_12_36	4539	25	267	2	131	11		1672	317	23	2109
Aonix 7.2.2	828	2	18		4			196	23	5	1080
GCC 3.15p, Gps1.4	2976	1	55		4	1	8	255	236	3	14070
AI-302	147				1			1	1		240

Figure 2 – Bug Warnings

6 Evaluation

It was relatively easy to use our utility to search for bugs in the Ada source code we had available, however, evaluating the results is a time-consuming and subjective process.

Figures 2 and 3 summarise the results of our using the Ada Bug Finder version 1.4 on the following applications and libraries

- Soudronic AG, UniControl release 1.3
- Siemens AG, ILTIS PC release 3622_12_36
- Source code provided with the Aonix compiler version 7.2.2
- Gnat open source for GCC version 3.15p, Gps 1.4 and Xml
- Charles library & AI- 302

Figure 3 - Style Rule Violations

Sources	Total	CNP	HTE	NDO	NGS	NPUC	NVIS	SCM
UniControl 1.3	25		9	16				
ILTIS 3622_12_36	2109	24	22	1443		169	451	
Aonix 7.2.2	1080		16	20	2	475	567	
Gnat GCC 3.15p, Gps1.4	14070		17	314	462	11303	1974	
AI-302	240			2	3	233	2	

In Figure 2, the number of files that the utility analysed is provided in order to give some sort of idea as to the their relative sizes.

Unfortunately we have had neither the time nor the resources to make anything other than a cursory evaluation of the results.

However, we have been able to make the following observations:

1. The ILTIS application was the only Ada 83 code we analysed. This explains why it alone contained illogical operator renaming bugs.
2. We believe that the low number of CNR bugs within the Gnat code base can be attributed to it normally being compiled using the Gnat Ada Compiler which itself issues this type of warning.
3. The vast majority of reported bugs were harmless unused declarations of some sort. However, we believe that removing this clutter generally improved the readability of the code.

7 Conclusions

The utility has been instrumental in discovering several bugs that had made their way into production code.

Some of these bugs were so obscure that they would probably be very difficult to discover using traditional methods.

For example, the UniControl Wrong Granularity (WG) bug informed an API that a buffer was larger than it really was. The consequence of this was that occasionally, depending on what the function wanted to return, code would get overwritten and the application would crash.

Although written to search for bugs in existing code bases, we have discovered that the utility is also a useful development tool. Occasionally running the Bug Finder over newly developed code before it has been released or submitted into a library has detected several bugs that probably would have only been detected during testing.

Figure 4 - Bug Pattern Codes

Code	Description
CNR	Code Not Reachable
ENR	Exception Not Raised
IOR	Illogical Operator Rename
NSC	Non Short Circuit
NP	Null Pointer
SE	Syntax Error
UD	Unused Declaration
UU	Unused Unit
WG	Wrong Granularity
Style	Style Rule Violation

Figure 5 - Style Rule Codes

Code	Description
CNP	Code Not Portable
HTE	Handle Task Exceptions
NDO	No Declaration Overloading
NGS	No Goto Statements
NPUC	No Package Use Clause
NVIS	No Variable In Specification
SCM	Superfluous Code Mark

8 An alternative method

From start to finish, the Ada Bug Finder project, including testing and presentation, took 140 Man-hours of effort.

We were able to develop the utility within these constraints by reusing an Ada text parser that we had developed for a previous project.

However using static code analysis has severe limitations. The utility simply does not know enough about the semantics of the code it is analysing for it to detect some of the bug patterns we had hoped to implement.

An alternative method could be to use the ASIS compiler interface [3]. This is an open, published callable interface that gives access to semantic and syntactic information from an Ada environment.

Acknowledgments

Siemens Schweiz AG sponsored the development of the Ada Bug Finder

References

[1] David Hovemeyer and William Pugh, *Finding Bugs Is Easy*. Department of Computer Science, University of Maryland, College Park, Maryland 20742 USA {daveho, pugh}@cs.umd.edu

[2] Ausnit-Hood, Johnson, Pettit & Opdahl, *Ada 95 Quality and Style*. LNCS 1344, Springer-Verlag

[3] Ada Semantic Interface Specification (ASIS) JTC1/SC22 ISO Standard ISO/IEC 15291:1999

Ada Development for a Basic Train Control System for Regional Branch Lines

Burkhard Stadlmann

Upper Austrian University of Applied Sciences, Wels College of Engineering
Stelzhamerstr. 23, 4600 Wels, Austria; Tel: +43 7242 72811 3420; email:b.stadlmann@fh-wels.at

Abstract

This paper presents a basic train control system for regional branch lines which uses radio based operational train control. This new system has the goal of improving the safety of this operational train control. The system has been developed to be cost effective using standard hardware a UML-design, and Ada as the programming language.

Keywords: train control system, operational train control, UML, Ada.

1 Introduction

A new kind of train control system for branch lines has been developed at the Upper Austrian University of Applied Sciences in Wels in cooperation with the railway operator Stern & Hafferl and financially supported by the province of Upper Austria. From the start of the design process attention has been focused on low cost solutions and user friendly operational sequences. The software design is based on UML and mainly implemented in Ada. This paper presents the idea and the functionality of that system as well as the experiences that have been made with its implementation. It is a project which is comparably small and which is explicitly designed as a low cost solution with regard to hardware and software. In conjunction with the operational personnel, the system should improve the overall safety of this operational train control.

2 The background

There are numerous regional branch lines with very simple operational conditions with radio-based operational train control (in German: *Zugleitbetrieb mit Sprechfunk*). The single-track line traffic controller gives the movement authority to the train driver via radio phone. (e.g.: Train *x* has movement authority until *A*-stop where it will cross paths with train *y*). The movement authority will be marked in a graph of train movements

When using this kind of system, the level of safety is poor. The failure of one single person in the chain could be the cause of a bad accident. If the single-track line traffic controller gives two overlapping movement authorities, or if the train driver does not stop when required the failure might remain undetected until the trains crashed. With respect to Austria, there are tragic examples of those kinds of accidents. In 2002 there were accidents on the Danube river line between Krems and Grein (a mistake made by the train controller) as well as on the line in the Mur valley (a mistake of the train driver). [1]

The goal is to improve this dissatisfying situation. As many of these regional lines cannot afford to invest in traditional signalling equipment, a much cheaper solution is required. This combination of affordable solution and improving actual safety level is the difference to other systems like the German "*Funkfahrbetrieb*" which could not achieve its goal mainly due to economic reasons. [2]

3 Basic idea of the train control system

The newly developed train control system is based on the same operational principles by radio phone as before. But the operation receives computer-aided support with the addition of a radio data system for the communication between trains and central train controller. The train controller performs the same routines as before, but now he uses screen and mouse instead of paper and pencil. Train drivers obtain their movement authority from the screen instead hearing it via radio and writing it on a sheet of paper. To achieve the goal of a low cost system no redundant hardware systems are used and the software development process is not as complicated as for high safety systems, both of which have an adverse effect on costs and safety.

4 The components of the system

The basic system concept is presented in Figure 1 and consists of a central computer and an onboard-computer in each train. The communication between trains and central device is performed by a line specific data radio system.

4.1 The central station

The central computer runs under the operating system Windows 2000 and consists of a *core application*, implemented as an Ada-application, and a *GUI for visualisation*, which has been implemented as a Java-application. The communication between these two parts runs over TCP/IP. The hardware is a standard server with all necessary peripheral equipment like radio-controlled clock for precise timing etc. The central station is equipped with a GPS-Receiver which generates the correction data for the train location.

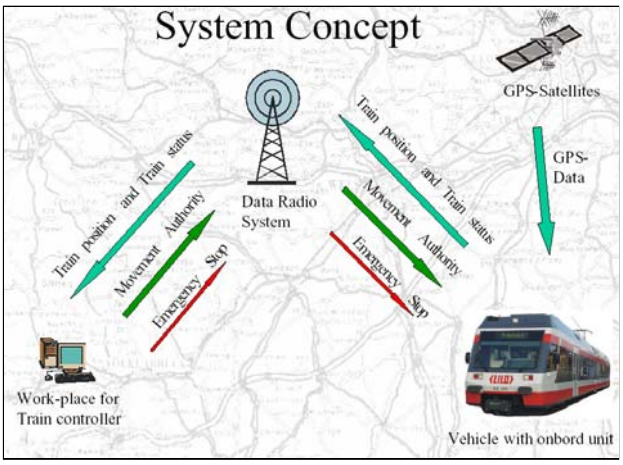

Figure 1 The system concept

The administration of the movement authorities and the communication between it and the trains is performed within the *core application*. The application prohibits the train controller entering two overlapping movement authorities. As a background safety measure a special task has been implemented within the core application which monitors the distance between trains. If this distance reaches a lower limit, the task will trigger an alarm in both trains to execute an emergency stop. The central computer also performs electronic protocolling of all user inputs and of all data traffic via radio.

Within the *visualisation part (GUI)* two different representations of the line are available. On the one hand there is a schematic representation of the line and its tracks (upper part of Figure 2), and on the other hand there is a scaled electronic train diagram including theoretical as well as actual train running (lower part of Figure 2). Both presentations are available simultaneously on screen at the central computer.

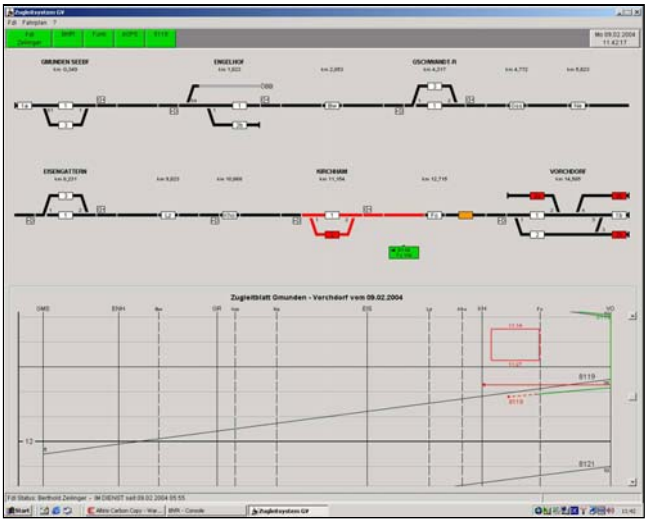

Figure 2 GUI of the central station

The visualisation of the movement authorities and the train locations are available in both representations, each in a different manner. Within the schematic line presentation the line changes from black to red and within the train diagram a red arrow marks the given movement authority.

4.2 The onboard units

The hardware of the board computer is a standard robust industrial PC including the appropriate I/O-modules and interfaces. This is a low cost solution and it makes sure that similar systems will be available on the market for a long time.

The software of the *board computer* is an Ada-multitasking-application under the operating system ETS and is responsible for the following actions:

- Train location based on dGPS-data and an incremental sensor at one axle,
- data communication between it and the central computer (periodical status messages, handling of all operational control messages including emergency brake message),
- visualisation and communication to the train driver (HMI),
- supervision of the correct execution of all movement authorities received,
- electronic protocolling of selected driver's actions,
- dynamic passenger information in the train,
- additional features for support and service.

5 System safety

All safety-relevant actions within the computer-based train control system require the explicit input of the operator or the train driver. The supervision of the system by the personnel is supported by the permanent visualisation of all relevant system states including an appropriate failure management system.

Safety of this system is based on numerous redundancies within the software as well as the already mentioned controlling inputs of the operating personal.

The software development process has been structured in analogy to most of the requirements of the EN 50128, however, in order to limit the software development costs the implemented software does not meet the requirements for the software safety level 4 as defined in the EN 50128 [3] (meaning the software is not as safe as is common for electronic signalling systems).

The design process began with a detailed description of the system requirements, themselves the result of collaborations between the railway company and software developers.

6 The software development process

The software development process is based on the principles of EN 50128 using the V-model (see Figure 3). Based on the system requirements a systematic UML-design was created. [4]

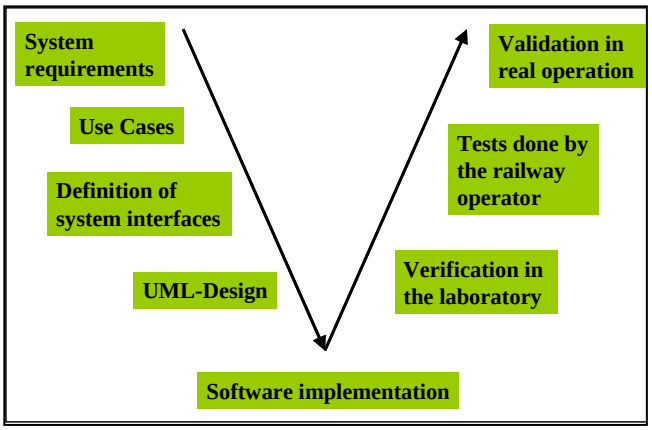

Figure 3 V-model

6.1 UML diagrams

Based on the system requirements there have been specified Use Cases for all operational input / output sequences. For each Use Case there have been worked out two different representations: On the one hand sequence diagrams in UML and on the other hand a textual description of what happens in natural language but with a formalized structure. Figure 4 shows the first part of the sequence diagram representing the sending of an arrival message from the train driver to the central computer.

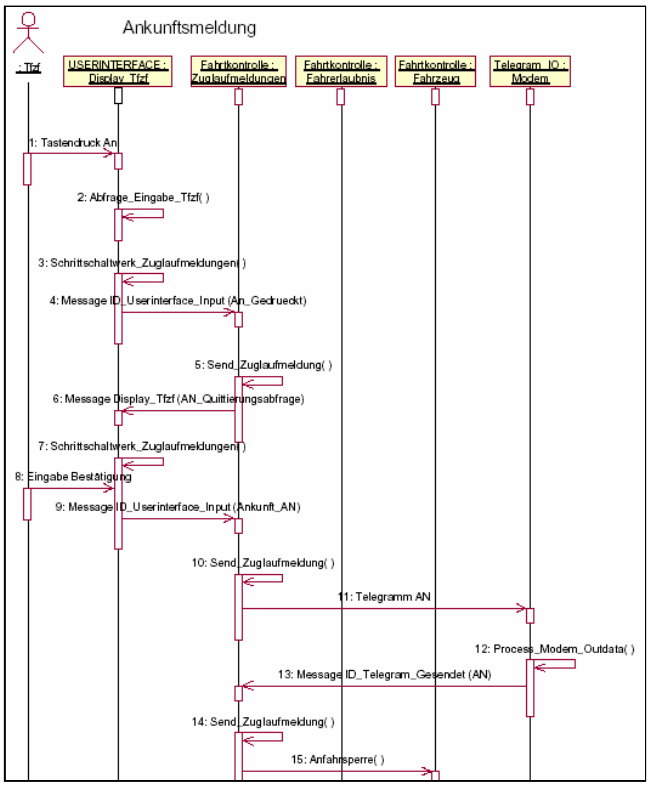

Figure 4 Sequence diagram for arrival message

The level of detail in the sequence diagrams has been chosen to meet the task structure as well as this rough object structure. This step of the development process made it possible to analyse and to improve the tasks of the software tasks and the kind of communication relations between the tasks and the rough object structure. The required messages could be specified and implemented very quickly using the visualisation of the communication relations within the sequence diagrams.

The software tasks have a cyclic behaviour. For designing and verification of these cyclic functions for all tasks *activity diagrams* have been designed. The example for this type of diagrams shows the calculation of the train location. Figure 5 shows the steps of activity during one calculation cycle. The train location is calculated using two different measurements (GPS-data and incremental encoder mounted at one axle). Combining these two measurement values during regular service and during error condition is a very complex calculation. Several calculation steps must be separated as much as possible. The logical path of data should be visible. An activity diagram creates the necessary clearness for the understanding.

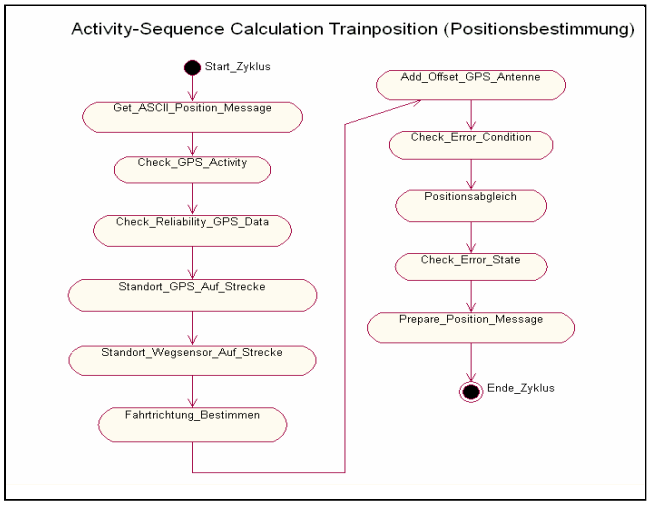

Figure 5 Activity diagram

State machines have been used to operate properly for the different states of each activity. For each state machine a *state diagram* has been designed. Figure 6 shows an example of a state diagram.

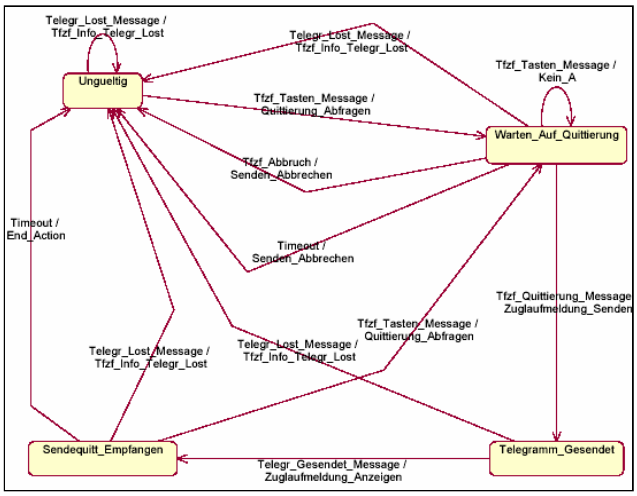

Figure 6 State diagram

In those state diagrams all states have been documented. Events and activities have been partly included. All events

and all activities have been documented if they belong to normal operational activities or normal failure states. Special failure states have not been included for the reason of better clarity.

The last step of the UML design was the design of the class diagrams for the whole project. All methods but not all attributes (due to the lack of clarity) have been documented within the class diagrams. The messages and their communication structure are a vital part of this message based system and they are documented within the class diagrams as well. Figure 7 shows an example.

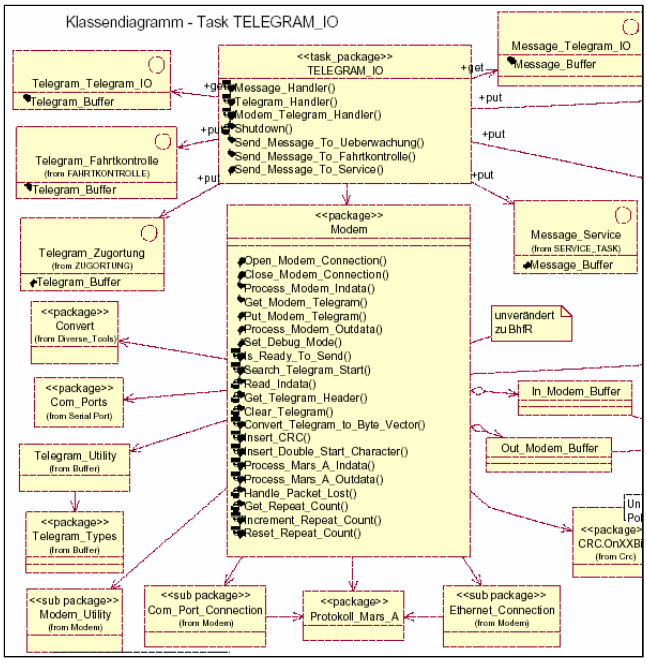

Figure 7 Class diagram

6.2 Implementation

As previously mentioned the application was implemented mainly in Ada and the visualisation using Java. Automatic code generation was not used. The decision to use Ada was influenced by the recommendations within the EN 50128 for safety critical software within railway systems. Java has been used for the visualisation because of the better support of graphical features. Ada turned out being very useful within this relatively small project and its small group of developers.

The state machines have been implemented using generic Ada-packages. The inter-task communication is implemented as a message handling system with an appropriate amount of buffers using generic protected message buffers.

One task of the board computer supervises the functionality of all the other tasks and checks all error messages, to ensure that they are not dangerous. Combined with a hardware watchdog this supervisory task can perform a system reset if necessary. This supervisor task is capable of starting six tasks each of them managing in turn the user interface, the driving management, the train location, the radio communication, the process input-output, and the service interface.

The design of the Ada part of the central computer is very similar to the design of the board computer. The task for collision supervision is separated as much as possible. A supervisory task controls the cyclic life messages of all other tasks. The Java-part communicates with the Ada part via TCP/IP.

Special features improving safety have been implemented to partly compensate for the lack of hardware redundancy. This includes using a CRC checksum for important state variables as well as for the memory image of the driver's display and cyclic integrity checks of the digital line atlas which is the data base for train location.

7 Conclusion

This paper presents a basic train control system for regional branch lines which uses radio based operational train controls. This new system has the goal of improving the safety of this operational train control. As regional branch lines often do not have enough money for traditional signalling equipment, such a train control system must be a low cost solution. Thus the presented system does not require any further lineside installation and it uses standard hardware for the central station, as well as for the onboard units. The communication between the trains and the central station is implemented by a line specific data radio system. System safety is achieved combining software and operational control done by the single-track line traffic controller and the engine driver. The train control system is an Ada and Java implementation based on a systematic UML-design.

Practical experience since January 2003 on the line Gmunden – Vorchdorf has improved the system and enables the implementation on further lines in Upper Austria.

References

[4] Eisenbahn Österreich (Jahrgang 2002), *Berichte über die Unfälle auf der Donauuferautobahn und im Murtal*, Minirex Verlag Luzern.

[5] J. Pachl (2005), *Entwicklung der Leit- und Sicherungstechnik für das System Bahn*, Eisenbahntechnische Rundschau, Jg 2005, Heft 3 – März pp 96, Eurailpress Tetzlaff-Hestra Hamburg.

[6] EN 50128, *Bahnanwendungen Telekommunikations-technik, Signaltechnik und Datenverarbeitungssysteme – Software für Eisenbahnsteuerungs- und Überwachungssysteme*, Ausgabe 2001-11.

[7] G. Booch,, The Unified Modelling Language User Guide, Addison Wesley Longman.

Ada-Europe 2005 Sponsors