ADA
USER
JOURNAL

Volume 31
Number 3
September 2010

Contents

Editorial Policy for Ada User Journal

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

Further details on our approach to these are given below.

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

News and Product Announcements

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

Reprinted Articles

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

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

Commentaries

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

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

Announcements and Reports

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

Reviews

Inclusion of any review in the Journal is at the discretion of the Editor.

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

Submission Guidelines

All material for publication should be sent to the Editor, preferably in electronic format. The Editor will only accept typed manuscripts by prior arrangement.

Prospective authors are encouraged to contact the Editor by email to determine the best format for submission. Contact details can be found near the front of each edition.

Example papers conforming to formatting requirements as well as some word processor templates are available from the editor. There is no limitation on the length of papers, though a paper longer than 10,000 words would be regarded as exceptional.

Editorial

Exactly one year ago, the September 2009 issue (Volume 30, Number 3) of the Ada User Journal published the Proceedings of the “Workshop on Vulnerabilities”, an event co-located with the Ada-Europe 2009 conference. This workshop was the beginning of a process intending to produce an Ada annex to the ISO Technical Report on Avoiding Programming Language Vulnerabilities through Language Selection and Use (ISO/IEC PDTR 24772.2); this process was concluded in a meeting at the Ada-Europe 2010 conference, this June in Valencia.

This issue of the Journal publishes the resulting document, co-edited by Alan Burns of the University of York, UK, and Joyce Tokar of Pyrrhus Software, USA, which was submitted to ISO/IEC JTC 1/SC 22/WG23, the working group responsible for the technical report. This document identifies the vulnerabilities that are applicable to Ada, and describes how these can be avoided or mitigated. In the scope of this process an annex for SPARK was also developed, which will be published in the next issue of the Journal.

Also in this issue, the reader will find the specification for the first student programming contest organized by Ada-Europe. This annual contest, under the name "The Ada Way", is an effort to promote the use of Ada by students and educators “in a form that is both fun and instructive”. This first edition opened this September, being the theme a software simulator for a football match; submissions will be accepted during April 2011.

As usual, the issue presents the news digest and calendar sections, providing the readers with a quick look to the world of Ada. The forthcoming events section provides the call for papers for “The Ada Connection”, next year in Edinburgh, Scotland, an event that Ada practitioners must attend, and that combines the Ada-Europe 2011 conference and the Ada-Conference UK 2011. The section also provides information about SIGAda 2010, which will take place this year in Fairfax, Virginia, USA, in the week of October 24 to 28. Finally, the Ada Gems section provides two gems on type-based security, by Yannick Moy of AdaCore.

Luís Miguel Pinho
Porto
September 2010
Email: lmp@isep.ipp.pt

Quarterly News Digest

Marco Panunzio

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

Contents

Ada-related Organizations

18th Award Ada-Spain

From: Ada-Spain website
Date: Wed, 1 Sep 2010 [retrieved —mp]
Subject: XVIII award Ada-Spain for the best academic project related to the Ada programming language
URL: http://adaspain.unican.es/ Premio_XVIII.pdf

The goal of this award is to foster the adoption of the Ada programming language in University teaching programs as well as in professional training programs, both as study subject and as language for the realization of work projects and investigations.

The details of the award and the criteria for the eligibility of works follow:

1. It is hereby established an award granted with a price of 750 Euro for the winner and a price of 450 Euro for the runner-up.
2. The work can be realised by a group of persons, not exceeding the number of three persons.
3. The submitted projects shall be part of the academic work of their authors, who should be studying in an officially recognised teaching center (academic or for professional training) during the realisation of the project. In particular, those works can be either projects assigned at the end of the study period or projects assigned during a course.
4. The deadline for the submission of the projects is February 15, 2011.
5. The submitted projects shall be sent via post courier to the following address:

   Ada-Spain
   Apartado Postal 50.403
   28080 Madrid

   An electronic copy of the requested documentation shall be sent via email to the Ada-Spain Secretary (Javier Gutiérrez, gutierjj@unican.es)
6. The accompanying documentation shall include:

   a) Personal data of the participants.

   b) A presentation letter of the professor or senior instructor that is supervising the submitted project.

   c) All the documentation that is considered necessary, relative to the submitted project.

   d) An executive summary of the project, in electronic format, not exceeding 10 pages.
7. Ada-Spain is not obliged to return the documentation submitted for the projects.
8. The evaluation committee will be composed by the members of the Board of Ada-Spain, and will notify its decision to the winning participants within three months from the submission deadline. The result of the contest will also be published in the web page of Ada-Spain. A summary of the winning projects will be published as well.
9. The evaluation committee reserves the right of not to award the price if no submitted project is deemed of sufficient quality.

[Translated from Spanish. For the original announcement (in Spanish), please refer to http://adaspain.unican.es/ Premio_XVIII.pdf —mp]

Ada-related Events

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

Ada-Europe Programming Contest

From: Dirk Craeynest <dirk@asgard.cs.kuleuven.be>
Date: Thu, 17 Jun 2010 00:56:49 +0200 CEST
Subject: Press Release - Ada-Europe launches Programming Contest
Newsgroups: comp.lang.ada,fr.comp.lang.ada, comp.lang.misc

FOR IMMEDIATE RELEASE

Ada-Europe Launches

Annual Student Programming Contest "The Ada Way"

VALENCIA, Spain (June 16, 2010) On the occasion of Ada-Europe 2010, the 15th annual Conference on Reliable Software Technologies, Ada-Europe, the international organization that promotes the knowledge and use of Ada in European academia, research and industry, launched an annual Student Programming Contest under the provisional title of "The Ada Way".

The contest will be a yearly competition among student teams, whereby each team must have a codename and a logo, a university affiliation, and the endorsement by an educator. The theme of this year's contest will be announced by September 1, and submissions will be accepted until April 30 of the following year. A Steering Committee composed of representatives of promoting institutions will oversee the organization of this contest.

Submissions will be marked by an Evaluation Committee composed of leading Ada experts, such as John Barnes (author of the famous Programming in Ada books), S. Tucker Taft (leader of the Ada 95 language revision), Ed Schonberg (co-author of the open-source GNAT Ada compiler and toolset), Joyce Tokar (convenor of the ISO working group on the Ada language standards), etc.

The winning team will be notified by May 31. The Steering Committee will offer the winners one free registration, accommodation, and airfare to the Ada-Europe conference, a slot in the conference program, publication space in the Ada User Journal, and visibility in other media. Additional prizes might be offered.

Ada-Europe wants the competition to be fun and educational. The theme of the contest will be determined by the Steering Committee, and shall be intellectually challenging and elating. Evaluation criteria shall include correctness, clarity and readability of the code, ingenuity, cuteness, and time and space efficiency.

Submissions shall include the source code and the User Manual. The code must run out-of-the-box when following the User Manual's instructions. The mplementation does not need to be 100% Ada, but of course the essence must be in Ada, and only the Ada code will be part of the evaluation. Tullio Vardanega, president of Ada-Europe, stated: "The winning submission must be a reference for good Ada programming, software design, and innovation."

About Ada-Europe

Ada-Europe is the international non-profit organization that promotes the knowledge and use of Ada into academia, research and industry in Europe. Current member organizations of Ada-Europe are: Ada-Belgium, Ada in Denmark, Ada-Deutschland, Ada-France, Ada-Spain, Ada in Sweden and Ada-Switzerland. Ada-Europe also includes and welcomes individual members from other European countries with no national organization, and has a total membership in the region of 300.

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

Press contact

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

(V9.1)

Call for Papers Ada Connection – Ada-Europe 2011

From: Steve Riddle <riddsteve@gmail.com>
Date: Wed, 21 Jul 2010 07:21:27 -0700 PDT
Subject: The Ada Connection - 20-24 June 2011, Edinburgh, UK
Newsgroups: comp.lang.ada

The Ada Connection = Ada Europe 2011 + Ada Conference UK 2011

Call for papers, tutorials and workshops is now available at http://conferences.ncl.ac.uk/adaconnection2011/

The Ada Connection combines the 16th International Conference on Reliable Software Technologies – Ada-Europe 2011 – with Ada Conference UK 2011. It will take place in Edinburgh, Scotland's capital city and the UK's most popular conference destination.

In traditional Ada-Europe style, the conference will span a full week, including a three-day technical program and vendor exhibition from Tuesday to Thursday, along with parallel tutorials and workshops on Monday and Friday. The Ada Connection will also encompass technical and vendor tracks under the banner of Ada Conference UK, which exists to promote awareness of Ada and to highlight the increased relevance of Ada in safety- and security-critical programming.

From: Dirk Craeynest <dirk@cs.kuleuven.ac.be>
Date: Sun, 29 Aug 2010 20:30:14
Subject: CfP 16th Conf. Reliable Software Technologies, Ada-Europe 2011
Newsgroups: comp.lang.ada,fr.comp.lang.ada, comp.lang.misc

CALL FOR PAPERS

The Ada Connection

16th International Conference on

Reliable Software Technologies - Ada-Europe 2011

+

Ada Conference UK 2011

20 - 24 June 2011, Edinburgh, UK

http://www.ada-europe.org/conference2011

Organized by Ada-Europe,

in cooperation with ACM SIGAda (approval pending)

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

[…]

Schedule

----

21 November 2010: Submission deadline for regular papers, tutorial and workshop proposals

08 January 2011: Submission of industrial presentation proposals

08 February 2011: Notification of acceptance to authors

08 March 2011: Camera-ready version of regular papers required

16 May 2011: Industrial presentations, tutorial and workshop material required

20-24 June 2011: Conference

[…]

[see also the Forthcoming Events section in this AUJ issue —mp]

Call for Papers SIGAda 2010

From: Michael Feldman <mfeldman@seas.gwu.edu>
Date: Fri, 11 Jun 2010 14:57:41 -0500
Subject: REMINDER: approaching deadline (June 25, 2010) for SIGAda 2010 submissions
Newsgroups: comp.lang.ada,comp.edu, comp.software-eng,comp.realtime

Hello,

This is a brief and gentle reminder of the approaching submission deadline -- June 25, 2010 -- for technical contributions to SIGAda 2010

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

This conference will take place October 24-28, 2010 in Fairfax, Virginia (Washington, DC area), at the Hyatt Hotel Fair Lakes.

We're soliciting Technical Articles, Extended Abstracts, Experience Reports, Panel Sessions, Workshops, and Tutorials on the Ada programming language and related technologies for developing, analyzing, and certifying reliable, safe, secure software.

We are especially interested in experience in integrating these concepts into the instructional process at all levels.

Please visit the conference website at

http://sigada.org/conf/sigada2010

for further details.

Please forgive us if you receive several copies of this message because you are on several mailing lists. Thank you very much for your time, and thank you in advance for your contribution!

Yours truly,

Michael Feldman

Professor Emeritus, Dept. of Computer Science

The George Washington University, Washington, DC

Registration and Publicity Chair, SIGAda 2010

Nominations for 2010 SIGAda awards

From: John McCormick <mccormick@cs.uni.edu>
Date: Tue, 13 Jul 2010 08:23:43 -0700 PDT
Subject: Nominations for 2010 SIGAda awards
Newsgroups: comp.lang.ada

Dear Members of the Ada Community:

On Thursday, 28 October 2010, the 2010 SIGAda Awards will be presented in a special morning plenary session at the SIGAda 2010 conference in Fairfax, Virginia. (See http://www.sigada.org/conf/sigada2010/ 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.sigada.org/exec/awards/awards.html#Previous 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.sigada.org/exec/awards/awards.html#Form.

Submit your nomination as an e-mail or e-mail attachment to SIGAda-Award@acm.org.

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 Monday September 13!

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

Thank you,

John McCormick

Past Chair ACM SIGAda

Ada-related Resources

Dhryston benchmark

From: Bill Findlay <findlaybill@blueyonder.co.uk>
Date: Sat, 24 Jul 2010 00:20:42 +0100
Subject: Dhrystone
Newsgroups: comp.lang.ada

Does anyone know where the original Ada source of the Dhrystone benchmark can be downloaded? So far I've only identified the ACM portal, to which I have no access.

From: BrianG <briang000@gmail.com>
Date: Fri, 23 Jul 2010 22:11:02 -0400
Subject: Re: Dhrystone
Newsgroups: comp.lang.ada

[…]

Did it exist in the PAL or ASE archives? I found this, but didn't see it there.

http://archive.adaic.com/ase/support/cardcatx/ad_index.htm

From: Georg Bauhaus <rm-host.bauhaus@maps.futureapps.de>
Date: Sat, 24 Jul 2010 09:09:47 +0200
Subject: Re: Dhrystone
Newsgroups: comp.lang.ada

[…]

The FTP archive at Ada Belgium (ftp.cs.kuleuven.ac.be) lists a zip in /pub/Ada-Belgium/ase/ase02_02/benchmrk/piwg/piwg_11/piwg

File A000091.Ada from the zip archive starts with these lines:

--A000091

[…]

-- *As published in Communications of ACM, October 1984 Vol 27 No 10*

(This time Google groups search has worked for me. One of the two messages in the result was written by someone with username stt and points to "the PIWG benchmarks" :)

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Sat, 24 Jul 2010 08:24:52 -0700 PDT
Subject: Re: Dhrystone
Newsgroups: comp.lang.ada

[…]

This most probably was Simon Tucker Taft

AdaLog's tools documentation in CHM format

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Tue, 17 Aug 2010 18:47:22 +0200
Subject: CHM version of AdaLog's tools documentation (for Windows users)
Newsgroups: comp.lang.ada

Hi all,

For Windows users: I've finished to set up a Windows version of the AdaLog's tools documentation, which you can get here:

http://www.les-ziboux.rasama.org/download/AdaLog.chm

The search facility is actually a bit limited for the reason given in the file. However, you will still enjoy the common tree-view facility (especially useful with AdaControl rules which are numerous) and the local favorites capability for quick access along with better integration with the look-and-feel of typical Windows applications.

Note: I did not setup topic IDs (which the Windows Help API requires) as this would be long and I do not know if this will really be used. If someone feel this is needed, just send me a mail, I will find the time to do it.

If you do not already know about AdaLog's tools, just visit:

http://adalog.pagesperso-orange.fr/compo1.htm

From: J-P. Rosen <rosen@adalog.fr>
Date: Tue, 17 Aug 2010 21:15:56 +0200
Subject: Re: CHM version of AdaLog's tools documentation (for Windows users)
Newsgroups: comp.lang.ada

[…]

Many thanks for doing that!

If you don't mind, I'd prefer to have three different help files for the three utilities, since the distributions are separate. I could then integrate them with the Windows distributions.

> If you do not already know about AdaLog's tools, just visit:

> http://adalog.pagesperso-orange.fr/compo1.htm

Please don't use this address (and remove it from any link you may have). This is the physical address that may change at any time. Use: http://www.adalog.fr/compo1.htm (in French) or http://www.adalog.fr/compo2.thm (in English)

Update of proof resources at SparkSure

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Mon, 23 Aug 2010 06:34:53 -0700 PDT
Subject: ANN: SparkSure proof resources now up-to-date
Newsgroups: comp.lang.ada

All the material on http://www.sparksure.com is up-to-date with the latest release of SPARK GPL 2010, GNAT GPL 2010, and GtkAda gpl-2.14.1.

- The tutorials for proof annotations and the Proof Checker
- The VC_View and PCHIF** tools
- The high integrity data structure examples.

Cheers,

Phil

** The version of the Proof Checker in the GPL 2010 release has an (apparently unintended) change in the way that it handles I/O - consequently is will not work with PCHIF. (The Proof Checker in the previous release - 8.1.1 - works correctly with PCHIF.)

New website ada.cx

From: Thomas Løcke
Date: Tue, 31 Aug 2010
Subject: ada.cx - A new Ada website!
URL: http://ada-dk.org/?page=news&news_id=176

Today, while chatting a bit on the #ada IRC channel, I stumbled on a new Ada website: ada.cx.

The website is still very new and under heavy development. Already though it sports some nice tools:

- An URL shortener

- A pastebin

- Planet Ada

- Get code

- Git hosting for Ada projects

The Planet Ada link is, in my humble opinion, very interesting. It is an aggregate to various Ada related blogs. Current writers include Dmitry Kazakov, Gustaf Thorslund and Tero Koskinen.

ada.cx is the brainchild of J. Kimball. You can get in touch with him on the #ada IRC channel, where he goes by the nick jkimball4.

Personally I wish ada.cx all the best. The more Ada websites, the better!

Ada-related Tools

Tables v1.10

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 27 Jun 2010 19:12:51 +0200
Subject: ANN: Tables 1.10
Newsgroups: comp.lang.ada

Tables are specialized maps with string keys. Additionally to normal map operations they support matching the longest key.

This version adds experimental Fedora and Debian packages.

[see also "Fuzzy sets for Ada" in AUJ 29-2 (Jun 2008), p.81 —mp]

Simple Components for Ada v3.9

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 11 Jul 2010 22:52:52 +0200
Subject: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

The library provides implementations of smart pointers, directed graphs, sets, maps, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, reentrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support. It grew out of needs and does not pretend to be universal. Tables management and strings editing are described in separate documents see Tables and Strings edit. The library is kept conform to both Ada 95 and Ada 2005 language standards.

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

The version 3.9 has experimental packages for Debian and Fedora linux. Note that due to gcc 4.4 bugs not all features are available. See release notes:

http://www.dmitry-kazakov.de/distributions/components_fedora.htm

http://www.dmitry-kazakov.de/distributions/components_debian.htm

From: Dirk Heinrichs <dirk.heinrichs@online.de>
Subject: Re: ANN: Simple components for Ada v3.9
Date: Mon, 12 Jul 2010 22:36:09 +0200
Newsgroups: comp.lang.ada

[…]

For which debian version are those packages? I assume "testing", because of gcc 4.4. OTOH you state that "APQ persistence layer is not supported because APQ is not yet packaged.", but APQ packages are available for "testing". So I'm a bit confused.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 13 Jul 2010 09:55:43 +0200
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

> […]

> For which debian version are those packages? I assume "testing", because of gcc 4.4.

Yes it is the "squeeze".

> OTOH you state that "APQ persistence layer is not supported because APQ is not yet packaged.", but APQ packages are available for "testing". So I'm a bit confused.

It wasn't there last time I looked for it. Do you have the package names (bin and dev)? I will take a look.

P.S. In any case in order to use the persistent layer of Simple Components, the gcc 4.4 must be fixed first. The current version has controlled types broken and some other severe issues. Unfortunately it is too early to switch from GNAT GPL 2009/10.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 13 Jul 2010 05:45:11 -0700 PDT
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

[…]

libapq1-dev (database-independent part)

libapq-postgresql1-dev (PostgreSQL-specific part)

Thanks to Adrian-Ken Rueegsegger for these packages. Unfortunately no other database-specific bindings are in Debian yet.

> P.S. In any case in order to use the persistent layer of Simple Components, the gcc 4.4 must be fixed first. The current version has controlled types broken and some other severe issues.

Wow, that's a pretty grave problem; if what you say is true, a fix in the stable GCC 4.4 branch is justified. What is the bugzilla number for this bug?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 13 Jul 2010 18:35:38 +0200
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

[…]

There are several. For example this one: […]

Some others can be found in the Simple Components tests.

I am not sure if I posted any of them there. I did report to AdaCore. Most of them were fixed before GNAT GPL 2009 was published.

Is a merge with GPL 2009/10 planned? I am asking because I still don't understand that complex mechanics governing FSF releases. In particular merits of posting two-three years old bugs all fixed in GNAT GPL, like the above bug.

I do have a base of bug reports I sent to AdaCore, but unfortunately I cannot post most of them, because they contain proprietary code. Then, of course, there are lots of bugs reported by other AdaCore customers. So my uneducated guess would rather be: let them do the merge first, and then we'll see.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 14 Jul 2010 07:54:10 -0700 PDT
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

[…]

Looks like this might be the patch that corrects this bug:

2009-04-17 Thomas Quinot <quinot@adacore.com>

- exp_ch7.adb (Expand_Ctrl_Function_Call): Remove incorrect special case for the case of an aggregate component, the attach call for the result is actually needed.
- exp_aggr.adb (Backend_Processing_Possible): Backend processing for an array aggregate must be disabled if the component type requires controlled actions.
- exp_ch3.adb: Minor reformatting

I'll add a backport of this patch to my long TODO list.

From: tonyg <tonythegair@googlemail.com>
Date: Mon, 12 Jul 2010 15:09:26 -0700 PDT
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

> For which debian version are those packages? I assume "testing", because of gcc 4.4. OTOH you state that "APQ persistence layer is not supported because APQ is not yet packaged.", but APQ packages are available for "testing". So I'm a bit confused. […]

I think this is because of the status of APQ rather than Dimitry's code. I have used these components recently to implement a database in ODBC and I have to say it is very good and does the job. As to which version, just use the gpr file as part of your gnat-gps project and it works very very nice.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 13 Jul 2010 10:06:52 +0200
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

[…]

Thanks, but regarding APQ it might indeed be broken. Actually I ceased to support APQ since GNAT 3.14, because APQ was not maintained. I happy to see that Debian guys want to revive it.

Though I am very disappointed with the present status of DB support in Ada.

There are too many projects, bindings are too low level. None of them attempt to make it platform independent. On the contrary, authors tend to be as much DBMS dependent as possible. There was a discussion on this in comp.lang.ada, people agreed to disagree.

From: tonyg <tonythegair@googlemail.com>
Date: Tue, 13 Jul 2010 01:37:21 -0700 PDT
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

[…]

Totally - I found it very hard to find something useful, the ODBC API packages you wrote to use with your persistent objects proved very useful though and I think they are probably the safest, most usable and thickest binding Ada programmers have available at the moment even though it was originally written for your persistent objects. It allows not so talented Ada programmers like myself to use the GNADE ODBC binding which lets face it is a pretty intimidating if not complicated beast.

[…]

From: Warren W. Gay <ve3wwg@gmail.com>
Date: Tue, 13 Jul 2010 16:59:13 +0000 UTC
Subject: Re: ANN: Simple components for Ada v3.9
Newsgroups: comp.lang.ada

[…]

> Thanks, but regarding APQ it might indeed be broken. Actually I ceased to support APQ since GNAT 3.14, because APQ was not maintained. I happy to see that Debian guys want to revive it.

I believe that Marcelo Coraça de Freitas is still maintaining APQ. I passed the reins over him some time ago and he has since moved his support to the following site AFAIK:

http://framework.kow.com.br/projects/show/apq

The original (now unmaintained) site was/is here:

https://sourceforge.net/projects/apq

I just ran out of time to keep it up myself. In fact, <cough>, I even gave up on Ada for a few years.. but I'm back now. ;-)

[see also "Simple Components for Ada v3.8" in AUJ 31-2 (Jun 2010), p.85 and "APQ 3.0 Beta1" in AUJ 30-2 (Jun 2009), p.75 —mp]

Generic Image Decoder v.01

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sat, 26 Jun 2010 02:49:20 -0700 PDT
Subject: Ann: Generic Image Decoder v.01
Newsgroups: comp.lang.ada

[…]

I have the pleasure to announce the first release of the Generic Image Decoder.

The Generic Image Decoder (GID) is an Ada package for decoding a broad variety of image formats, from any data stream, to any kind of medium, be it an in-memory bitmap, a GUI object, some other stream, arrays of floating-point initial data for scientific calculations, a browser element, a device,...

Animations are supported.

Currently supported formats are: BMP, GIF, JPEG, PNG, TGA.

URL: http://gen-img-dec.sf.net/

[…]

From: deadlyhead <deadlyhead@gmail.com>
Date: Sat, 26 Jun 2010 09:56:53 -0700 PDT
Subject: Re: Ann: Generic Image Decoder v.01
Newsgroups: comp.lang.ada

[…]

Quite excellent.

No bindings, eh? I'm impressed. Is it mostly translations? For instance, did you translate the libpng for the PNG encoding?

I'll try it out soon.

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sat, 26 Jun 2010 13:47:01 -0700 PDT
Subject: Re: Ann: Generic Image Decoder v.01
Newsgroups: comp.lang.ada

> Quite excellent.

Thanks - but try it first ;-)

[…]

JPEG is mostly translated from NanoJPEG; the rest has been written from scratch, with a good part of re-use from other Ada sources.

> I'll try it out soon.

I'd be glad to have some feedbacks.

[…]

New release of GLOBE_3D

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Mon, 16 Aug 2010 10:05:26 -0700 PDT
Subject: Ann: GLOBE_3D Aug-2010 release
Newsgroups: comp.lang.ada

[…]

A new release (with minor changes) is available @

http://globe3d.sf.net/

- A compose_rotations boolean field has been added to the type Camera; now camera rotations can meaningfully be used in an environment with gravity (fixed axes) or in a 0-gravity one (free axes).
- A bug in the video capture has been fixed: frames of any width are now captured correctly.
- The demo now includes a nice Airbus A319 model

NB: updates are tracked @ http://freshmeat.net/projects/globe_3d

[see also "GLOBE 3D" in AUJ 30-3 (Sep 2009), p.142 —mp]

Visual Ada Developer 7.4

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Wed, 4 Aug 2010 08:34:00
Subject: Ann: Visual Ada Developer VAD 7.4
Newsgroups: comp.lang.ada

Visual Ada Developer (VAD) 7.4 is now available at

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

VAD is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

VAD is distributed in the hope, that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

VAD 7.3 [sic —mp] Common description.

1. VAD (Visual Ada Developer) is a Tcl/Tk oriented Ada-95(TCL) GUI builder portable to different platforms, such as Windows NT/Vista/7,Unix (Linux), and Mac. You may use it as IDE for any Ada-95(C,C++,TCL) project.

VAD generated Ada sources, you may compile and build executable or generate TCL script to interpret with Tcl/Tk

VAD 7.4 was tested in Windows 32bit and 64bit and Linux x86-64 Kubuntu 9.10

2. Used software

GNAT GPL 2009 Ada-05 compiler (or any others)

TCL/TK 8.5.x
http://tcl.activestate.com/software/tcltk/

TCL/TK 8.6.x
http://tcl.activestate.com/software/tcltk/

Warning! VAD 7.4 has two realization for Tcl/Tk 8.5.x and Tcl/Tk 8.6.x , you need to install and test Tcl/Tk first.

From version tcl/tk 8.5.0.1 ActiveState distribution includes many of VAD used packages (Itcl,Img,Tktable,BWidgets, Tkhtml and so on).

You may choose your preferred version at link time. (I recommend to work with 8.5)

[complete list of used software removed —mp]

Warning! Many of Tcl/Tk packages were tested for TCL/TK 8.5 and 8.6 in Windows and x86-64 Linux, you may download them from my website

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

[see also "Visual Ada Developer 7.3" in AUJ 31-1 (Mar 2010), p.12 —mp]

From: Simon Wright <simon@pushface.com>
Date: Wed, 4 Aug 2010 21:32:09 -0800 PST
Subject: Ann: Visual Ada Developer VAD 7.4
Newsgroups: comp.lang.ada

> TASH 8.02 by Terry J. Westley
> http://tash.calspan.com/

This site is no longer existent, nor is Terry maintaining TASH. See

http://sourceforge.net/projects/tcladashell/

GtkAda Contributions v2.7 and v2.8

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Thu, 1 Jul 2010 19:12:22 +0200
Subject: ANN: GtkAda contributions v2.7
Newsgroups: comp.lang.ada

The described here packages are proposed as a contribution to GtkAda, an Ada bindings to GTK+. It deals with the following issues:

- Tasking support;
- Custom models for tree view widget;
- Custom cell renderers for tree view widget;
- Multi-columned derived model;
- Extension derived model (to add columns to an existing model);
- Abstract caching model for directory-like data;
- Tree view and list view widgets for navigational browsing of abstract caching models;
- File system navigation widgets with wildcard filtering;
- Resource styles;
- Capturing resources of a widget;
- Embeddable images;
- Some missing subprograms and bug fixes;
- Measurement unit selection widget and dialogs;
- Improved hue-luminance-saturation color model;
- Simplified image buttons and buttons customizable by style properties;
- Controlled Ada types for GTK+ strong and weak references;
- Simplified means to create lists of strings;
- Spawning processes synchronously and asynchronously with pipes;
- Capturing asynchronous process standard I/O by Ada tasks and by text buffers;
- Source view widget support.

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

Changes to the version 2.6:

- Quit_Error exception is propagated from operation in Gtk.Main.Router when the main loop was quitted. This change should ease proper application completion, e.g. when the main loop is quitted before the tasks making requests to GTK;
- Added GtkAda installation notes;
- Internal package Gdk.Pixbuf.Conversions was added to fix backward incompatibility introduced by GtkAda 2.14.2;
- xpm2gtkada is modified to support GtkAda 2.14.2;
- Experimental packages for Debian and Fedora.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 28 Jul 2010 21:40:50 +0200
Subject: ANN: GtkAda contributions v2.8
Newsgroups: comp.lang.ada

[…]

Changes to the version 2.7:

- Dir_Open, Dir_Close, Dir_Rewind, Dir_Read_Name were added to Gtk.Missed enumerate items of a directory;
- Remove, Rename were added to Gtk.Missed;
- File_Test was added to Gtk.Missed;
- GType_Icon was added to Gtk.Missed;
- GIO.Content_Type package provides bindings to platform-specific content typing (GContentType);
- Directory browser supports content icons;
- Gtk.Missed declares a controlled type provided to ease showing a wait cursor;
- Themed_Icon_New_With_Default_Fallbacks and Themed_Icon_New were added to Gtk.Missed;
- Packages GIO.Drive, GIO.Mount, GIO.Volume, GIO.Volume_Monitor were added to support GIO.

[see also "GtkAda Contributions v2.6" in AUJ 31-2 (Jun 2010), p.85 —mp]

VTKAda 5

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Tue, 3 Aug 2010 23:41:12 -0700 PDT
Subject: Announce : VTKAda version 5
Newsgroups: comp.lang.ada

VTKAda is an Ada-95(05) interface to VTK(Visualization Toolkit) and Qt4 graphics library VTK version 5.6.0, Qt version 4.7.0 open source, vtkc.dll(libvtkc.so) and qt4c.dll(libqt4c.so) built with Microsoft Visual Studio 2010 in Windows and GCC in Linux x86-64.

Package tested with the GNAT GPL 2009 Ada compiler in Windows 32bit and 64bit and Linux x86-64 Kubuntu 9.10 VTKAda is a powerful 2D-3D renderer system and works inside Qt4 applications.

You can get more information from my paper "Modern application development with Ada" from my website.

VTKAda and QtAda for Windows and Linux (Unix) are available from

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

[…]

QtAda 2.7.0

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Tue, 3 Aug 2010 23:37:25 -0700 PDT
Subject: Announce : QtAda version 2.7.0
Newsgroups: comp.lang.ada

QtAda is an Ada-95(05) interface to Qt4 graphics library Qt version 4.7.0 open source and qt4c.dll(libqt4c.so) built with Microsoft Visual Studio 2010 in Windows MINGW GCC Windows compiler and GCC in Linux.

Package tested with the GNAT GPL 2009 Ada compiler in Windows 32bit and 64bit and Linux x86-64 Kubuntu 9.10.

It supports GUI, SQL, Multimedia, Web, Net and many others thinks.

QtAda for Windows and Linux (Unix) is available from

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

[see also "QtAda 2.4 and relation with QtAda by Godunko et al." in AUJ 31-1 (Mar 2010), p.11 —mp]

Extensions to SPARK POGS

From: Phil Thornley <phil.jpthornley@gmail.com>
Date: Fri, 13 Aug 2010 10:21:16 -0700 PDT
Subject: SPARK POGS: List the rules used by the Simplifier
Newsgroups: comp.lang.ada

The POGS tool in the SPARK GPL toolset summarises the state of the proofs for a program. But it is not easy to determine which user rules have/have not been used by the Simplifier.

I have made a modified POGS that lists the rules used by the Simplifier as part of the summary output so, instead of just the list of rule files, it prints the following:

```
D:\SPARK\ordered_list2\ordered_lists\
ordered_lists.rlu
  ordered_user(1)   ordered_user(6)
  ordered_user(10)
  ordered_user(2)   ordered_user(7)
  ordered_user(11)
  ordered_user(3)   ordered_user(8)
  ordered_user(12)
  ordered_user(4)   ordered_user(9)
  ordered_user(13)
  ordered_user(5)

D:\SPARK\ordered_list2\ordered_lists\
delete.rlu
  delete_user(1)
  delete_user(2)

D:\SPARK\ordered_list2\ordered_lists\
initialize.rlu
  init_user(1)  init_user(4)  init_user(7)
  init_user(2)  init_user(5)  init_user(8)
  init_user(3)  init_user(6)  init_user(9)
```

http://www.sparksure.com/resources/POGSRuleList.zip is an archive with the new and modified files as well as a Windows executable.

http://www.sparksure.com/resources/pogsrulelist.patch is a patch file (created by git).

Ahven 1.8

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Wed, 2 Jun 2010 19:24:13 +0300
Subject: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

I released Ahven 1.8 today. It can be downloaded from

http://sourceforge.net/projects/ahven/files/

Ahven is a unit testing library for the Ada 95 programming language.

Version 1.8 is a bugfix release with following changes:

- Fix for double free when mixing dynamically allocated test cases with statically allocated test suites.
- Support for dynamic libraries was dropped. It was too complex (for me) to maintain Makefile/GPR-file logic which would work in the same way on Fedora, Debian, OpenBSD, and Windows.

If some packagers (like Debian people) want to enable support they can apply Makefiles and GNAT project files from Ahven 1.7 on top of 1.8.

- Support for Janus/Ada 3.1.1d was dropped.

  Janus/Ada 3.1.2beta or newer is required.

  Janus/Ada 3.1.1d support required too many work-arounds, so to make my life easier I decided to drop the support.

  If someone needs this, I welcome patches. :)

- TAP 1.3 test result format was dropped.

  Version 1.2 is supported.

The release has been tested with following compilers:

- FSF GCC/GNAT 4.3.5 and 4.4.4
- GNAT GPL 2009
- Janus/Ada 3.1.2beta
- Irvine ICCAda

The code can be compiled either as plain Ada 95 or as Ada 2005 code.

From: Jérôme Haguet <j.haguet@cadwin.com>
Date: Tue, 8 Jun 2010 08:24:50 -0700 PDT
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

Hello Tero.

FYI, Ahven 1.8 works also fine with Atego/ObjectAda 8.4 + U5 on Windows XP.

[…]

From: Dan Eilers <dan@irvine.com>
Date: Fri, 4 Jun 2010 10:12:37 -0700 PDT
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

There is a significant difference in registering tests between Ahven and Aunit, which Tero mentions, but I will expand on.

Ahven accepts parameterless procedures as test routines […]

AUnit, in contrast, requires test routines to have a parameter of type AUnit.Test_Cases.Test_Case'Class. I have thousands of procedures that I would like to test using a test driver, but unfortunately not a single one of those comes in the form AUnit requires.

If you start with an array of parameterless test procedures, AUnit requires that you traverse the array, creating a new test procedure (containing the required parameter) for each parameterless test procedure in the array. Then you can register the newly created test procedures.

The problem is that Ada doesn't have a good way for one procedure to create other procedures, that can then be registered (using 'access).

I tried using generics to instantiate the new test procedures, but ran into accessibility-level problems. I also tried using an allocator of a protected type that contains a procedure, but that results in a protected procedure, and AUnit only knows how to register normal (unprotected) procedures.

I presume that it may be possible to modify AUnit to support registering parameterless test procedures (or even better, an array of them), which would be convenient for my purposes. It would also be possible to write some sort of offline test-generator that creates the kind of test procedures that AUnit expects.

From: Stephen Leake <stephen_leake@stephe-leake.org>

Date: Sat, 05 Jun 2010 00:08:03 -0400
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

> AUnit, in contrast, requires test routines to have a parameter of type AUnit. Test_Cases.Test_Case'Class. I have thousands of procedures that I would like to test using a test driver, but unfortunately not a single one of those comes in the form AUnit requires.

Ah. Adapting an existing test suite to use the framework is an issue.

You could write one AUnit test that calls all the thousands of tests, but that would be silly.

You could probably write a tool to convert those tests to match AUnit, but that would still be a lot of work.

> The problem is that Ada doesn't have a good way for one procedure to create other procedures, that can then be registered (using 'access).

> I tried using generics to instantiate the new test procedures, but ran into accessibility-level problems. I also tried using an allocator of a protected type that contains a procedure, but that results in a protected procedure, and AUnit only knows how to register normal (unprotected) procedures.

You need to write new source code. Real software writes itself :) (a new slogan I just made up :).

> I presume that it may be possible to modify AUnit to support registering parameterless test procedures (or even better, an array of them), which would be convenient for my purposes.

Not easy; the tests are called by the standard Ada dispatching call methods.

> It would also be possible to write some sort of offline test-generator that creates the kind of test procedures that AUnit expects.

Yes.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 03 Jun 2010 08:08:12 -0400
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

Is this actually restricted to Ada 95? What prevents it from being used with Ada 2005? How does it compare to AUnit?

> The code can be compiled either as plain Ada 95 or as Ada 2005 code.

Ah; much better. Perhaps you should fix the intro statement to just say "Ada"; no one will be surprised if it doesn't work with an Ada 83 compiler.

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Fri, 4 Jun 2010 18:37:08 +0300
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

Here a little summary:

- The basic API is similar in Ahven and AUnit 1 (and 3).
- In Ahven, I have the public API stuffed into two packages: Ahven and Ahven.Framework while AUnit spreads it API into larger amount of packages.
- AUnit is Ada 2005 code only and I think it can be compiled only with GNAT (not 100% sure since I haven't tried it with other Ada 2005 compilers)
- Ahven is Ada 95 code, but can be compiled as Ada 2005 code also.

I have seen some effort to make Ahven build out of the box with several Ada 95 or Ada 2005 compilers. So far, GNAT, Janus/Ada, ObjectAda, and ICCAda have been tested.

(Unix and Windows environments)

I don't know about IBM/Rational Apex, GHS AdaMulti, or PowerAda, but I suspect that they should be okay also.

- Both are free or open-source software. AUnit is distributed under GPL and Ahven under ISC license (similar to BSD).
- Both libraries support XML results, but AUnit uses CppUnit's XML format while Ahven uses JUnit's format. In addition, Ahven can output test results in Test-Anything-Protocol (TAP) format, which is my favorite at the moment.
- AUnit's documentation is probably better. So far I have concentrated on the code.

> In AUnit, tests are 'registered' in a test_case:

Same works for Ahven.

[…]

> AUnit also has setup and teardown functions for initializing and finalizing each test and/or each Test_Case; those are very helpful.

Ahven supports setup and teardown procedures for each test.

For each Test_Case you need to rely on Initialize and Finalize procedures provided by Ada.Finalization.Controlled.

[…]

From: Simon Wright <simon@pushface.org>
Date: Sat, 05 Jun 2010 13:41:15 +0100
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

> * AUnit is Ada 2005 code only and I think it can be compiled only with GNAT (not 100% sure since I haven't tried it with other Ada 2005 compilers)

The 2005-ness occurs because AUnit uses a private copy of Ada.Containers.Lists with some 05 constructs removed so that it can be compiled with "an" Ada 95 compiler.

Unfortunately, what they mean is "with an Ada 95 compiler that recognises pragma Ada_2005 to mean that some Ada 2005 constructs are permitted even when compiling in Ada 95 mode".

This doesn't include older GNATs, and certainly won't include compilers from other vendors.

These patches update the current SVN source of AUnit to work with older GNATs. Ask me if you need a copy and the news system has mangled them or I've failed to drive Emacs correctly …

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sun, 06 Jun 2010 08:31:39 -0400
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

> Stephe, how about applying this patch to Debian's version of AUnit?

I don't see why. Debian supports the current version of GNAT, which supports the current Debian version (and newer AdaCore versions) of AUnit.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sun, 06 Jun 2010 18:20:46 +0200
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

The current version of GNAT (in Debian) allows compiling Ada 83, Ada 95 and Ada 2005 programs. I think that allowing Ada 95 programs to use AUnit is better than forbidding this. So the main objection is that this patch requires changing the aliversion; I would understand that you wouldn't want to do that.

From: Simon Wright <simon@pushface.org>
Date: Sun, 06 Jun 2010 17:45:14 +0100
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

I think it's very likely that the current Debian GNAT supports AUnit 3's use of this pragma, even in -gnat95 mode … […]

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 07 Jun 2010 04:26:51 -0400
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

> The current version of GNAT (in Debian) allows compiling Ada 83, Ada 95 and Ada 2005 programs.

True. I have no idea how many people use the old language. I should think the only reason to do that is for compatibility with a different compiler. And even then, the language version is not likely to be an issue.

For example, I used to program in DDC Ada for an embedded computer, but GNAT for development and unit test. I used GNAT in Ada 95 mode, while DDC was Ada 83. No problems.

> I think that allowing Ada 95 programs to use AUnit is better than forbidding this. So the main objection is that this patch requires changing the aliversion;

No, the main objection is that it is different than upstream. I don't have access to the AUnit test suite. I don't want to fork.

From: Simon Wright <simon@pushface.org>
Date: Mon, 07 Jun 2010 20:21:50 +0100
Subject: Re: ANN: Ahven 1.8
Newsgroups: comp.lang.ada

[…]

I've reported the issue, it's been accepted but as ever may take some time to trickle through!

[see also "Ahven 1.7" in AUJ 30-4 (Dec 2009), p.208 —mp]

Strings Edit for Ada v2.5

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 28 Jun 2010 20:56:25 +0200
Subject: ANN: Strings Edit v2.5
Newsgroups: comp.lang.ada

The package Strings_Edit provides I/O facilities. The following I/O items are supported by the package:

- Generic axis scales support;
- Integer numbers (generic, package Integer_Edit);
- Integer sub- and superscript numbers;
- Floating-point numbers (generic, package Float_Edit);
- Roman numbers (the type Roman);
- Strings;
- Ada-style quoted strings;
- UTF-8 encoded strings;
- Unicode maps and sets;
- Wildcard pattern matching.

The major differences to the standard Image/Value attributes and Text_IO procedures are:

- For numeric types, the base is neither written nor read. For instance, output of 23 as hexadecimal gives 17, not 16#17#.
- Get procedures do not skip blank characters around input tokens, except the cases when the blank characters are required by the syntax.
- Get procedures use the current string position pointer, so that they can be consequently called advancing the pointer as the tokes are recognized.
- Numeric get procedures allow to specify the expected value range. When the actual value is out of the range then depending on procedure parameters, either Constraint_Error is propagated or the value is forced to the nearest range boundary.
- Put procedures also use the current string position pointer, which allows to call them consequently.
- The format used for floating-number output is based on the number precision, instead of rather typographic approach of Text_IO. The precision can be specified either as the number of valid digits of the current base (i.e. relative) or as the position of the last valid digit (i.e. absolute).

  For instance, 12.345678 with relative precision 3 gives 12.3. With absolute precision -3, it gives 12.346.

The version 2.5 provides experimental Debian and Fedora packages.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 1 Jul 2010 03:34:57 -0700 PDT
Subject: Re: ANN: Strings Edit v2.5
Newsgroups: comp.lang.ada

[…]

Nice. I'll try to find the time to review the Debian packaging and provide feedback. Would you be willing to maintain the package in Debian in the future, i.e. deal with bug reports, reply to enquiries from users, etc.?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 2 Jul 2010 15:58:33 +0200
Subject: Re: ANN: Strings Edit v2.5
Newsgroups: comp.lang.ada

[…]

Sure, and I have other packages in the pipe line. But I didn't manage to understand the necessary procedure. It looked awfully complicated…

[see also "Strings Edit for Ada v2.4" in AUJ 31-1 (Mar 2010), p.12 —mp]

GNATGPR 0.32

From: David Sauvage <sauvage.david@gmail.com>
Date: Wed, 30 Jun 2010 14:33:43 -0700 PDT
Subject: Announce : Release of gnatgpr 0.32, access to GPR project information.
Newsgroups: comp.lang.ada

GNATGPR is an Ada 2005 GPL software, it allows the user to do simple information request on GNAT GPR project files, like for example :

- Give all included projects.
- Give all included sources (transitive or not).
- Give all included source directories (transitive or not).
- Give all main files.
- Give all main directories.
- Give all included object paths.
- Give project zombies (duplicated .o or .ali when a source file are moved from one project to another).

There are 2 ways of accessing those services:

- In a shell using the gnatgpr binary.
- In Ada using the GNAT_GPR package specification interface.

GNATGPR is based on :

- A modified version of the GNAT GPL 2008 (GPL) compiler front-end.
  - o It gets his own Namet and so on.
  - o It can extracts some especially craft comments in the gpr project file.
- AdaControl (GMGPL), for option analyzis.
  - o Only the Options_Analyzer package have been re-used (without any change).
- AUnit 2.03 (GPL), for unit testing.

Thanks to all the people behind those projects.

GNATGPR roadmap:

- Selection of the next gnatgpr engine (GCC FSF 4.4/GPRBuild/GNAT GPL 2010/GNATColl).
- Shift to GPL v3.
- Documentation.
- GNATGPR production for non-GNAT Ada compilers.
- GNAT_GPR package interface clean-up.
- Use of Debian AUnit version.
- Non transitive analysis for main files, binary directories and object paths.
- Multi languages processing.
- Multi root-projects processing.
- Comply to Debian packaging guidelines (removal of debian directory at upstream level, ...).

Debian & derived distributions repository (for package libgnatgpr0 libgnatgpr0-bin libgnatgpr0-dev):

deb http://ppa.launchpad.net/pariakanet/ppa/ubuntu lucid main

deb-src http://ppa.launchpad.net/pariakanet/ppa/ubuntu lucid main

See the project home page [1] to add the PPA repository key.

[1] https://gna.org/projects/gnatgpr

Matreshka v0.0.4 and v0.0.5

From: Vadim Godunko <vgodunko@gmail.com>
Date: Thu, 1 Jul 2010 11:30:05 -0700 PDT
Subject: Announce: Matreshka 0.0.4
Newsgroups: comp.lang.ada

Hello,

I am pleased to announce a new version of Matreshka. The goal of this project is to provide set of simple to use but powerful Ada libraries to construct information systems. Most important addition in the new version is non-validating XML processor with SAX2 interface. This version is compatible with both GNAT GPL 2009 and GNAT GPL 2010. Users of GNAT GPL 2010 can use all power of x86_64 processors, SSE2 instructions set is used to speedup several operations on strings.

Others important features are: support for unbounded form of strings of Unicode characters, including most useful operations (case conversion, folding, collation, normalization and iteration) implemented as specified by Unicode standard and non-backtracking Unicode compatible regular expressions engine. Someone can found interesting modified version of aflex (lexical scanner generator) which is extended to support full Unicode character set and by ability to specify set of characters by value of character's boolean properties.

Cross platform source code package can be downloaded from project's site:

From: coopht <coopht@gmail.com>
Date: Wed, 18 Aug 2010 13:19:24 -0700 PDT
Subject: ANN: Matreshka 0.0.5
Newsgroups: comp.lang.ada

Hello, I'm happy to announce new version of Matreshka toolkit version 0.0.5

This version includes following features:

- Text decoder (New) : various text encodings are supported
- Message translator (New) : for localization of application's messages
- New extensions of string manipulation package (New)
- XML reader (New) : socket as input source supported
- XML reader (New) : incremental parsing of input XML stream supported
- A lot of bug fixes and performance improvements.

The new version can be downloaded here:

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

[see also "Matreshka v. 0.0.3" in AUJ 31-2 (Jun 2010), p.87 —mp]

Units of Measurement for Ada v3.1

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 3 Jul 2010 20:28:13 +0200
Subject: ANN: Units of Measurement for Ada v3.1
Newsgroups: comp.lang.ada

The library provides an implementation of dimensioned values for Ada. Unit checks are made at run-time, if not optimized out by the compiler. SI and irregular measurement units are supported. Shifted units like degrees Celsius are supported too. Conversions from and back to strings are provided for all various irregular units. An extensive set of GTK widgets for dealing with dimensioned values is included, though use of GTK is not mandatory for the rest of the library.

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

This version adds experimental Debian and Fedora packages.

[see also "Units of Measurement for Ada v3.0" in AUJ 31-2 (Jun 2010), p.85 —mp]

Adasubst and Adadep for GNAT GPL 2010

From: J-P. Rosen <rosen@adalog.fr>
Date: Thu, 19 Aug 2010 14:28:12 +0200
Subject: [Ann] Adasubst and Adadep for Gnat GPL2010 released
Newsgroups: comp.lang.ada

Adalog is happy to announce new releases of AdaSubst and AdaDep. Apart from minor bug fixes, the main difference of these releases is that the precompiled versions are now for GNAT GPL 2010.

AdaSubst is a tool for semantic substitution of identifiers. It allows to change identifiers or reorganize package structures project-wide.

AdaSubst is fully aware of all Ada visibility rules, and does the right thing even in presence of overloading, use clauses, name hiding, etc.

AdaDep is a tool for analyzing dependencies. It tells which elements from a package are used by units that "with" the package.

Both tools are free software, released under the GMGPL license. Download from Adalog's component page:

http://www.adalog.fr/compo2.htm

[…]

[see also "AdaSubst & AdaDep" in AUJ 29-1 (Mar 2008), p.8 —mp]

Ada-related Products

AdaCore — GNAT GPL Edition 2010

From: AdaCore Libre Website
Date: Mon, 28 Jun 2010
Subject: GNAT GPL Edition
URL: http://libre.adacore.com/libre/tools/gnat-gpl-edition/

GNAT GPL Edition - 2010 Edition Now Available!

We are pleased to announce the release of GNAT GPL 2010, the integrated Ada,C, C++ toolset for for Academic users and FLOSS developers.

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

- First taste of Ada 2012 with features such as conditional or case expressions
- Enhanced IDE (tips, new views, better doc generator
- GNAT Project API available in GNATColl
- Support for Windows 7
- Support for MacOS X Snow Leopard (64 bit)
- Support for Visual Studio 2008 for the .NET Framework
- Support for AVR (embedded 8 bit)
- Ravenscar support for the Lego Mindstorm NXT port (future availability)

In the context of the GAP program, we hope the last 2 points above will provide new and interesting options for real-time and/or robotics classes.

GNAT GPL 2010 comes with version 4.4.1 of the GNAT Programming Studio IDE and GNATbench 2.4.0, the GNAT plug-in for Eclipse.

AdaCore is dedicated to developing the finest tools available for software development with Ada. The GNAT GPL Edition is the Ada 2005 development environment for Free Software development and the GNAT Academic Program for use in Academia. GNAT Pro for commercial/industrial development.

The GNAT GPL Edition consists of the following technologies:

- The GNAT Ada 2005 compilation system
- Various tools such as heap monitoring, unit testing, program browsing, etc.
- Ada 2005 and GNAT libraries (Containers, pattern matching, sorting, etc.)
- The GNAT Programming Studio (GPS) visual IDE

Related technologies are also available for download alongside the GNAT GPL Edition:

- AWS: to web-enable Ada applications (Ada Web Server)
- XML/Ada: to process XML streams in Ada applications
- ASIS: to develop tools for Ada software
- GtkAda: to develop modern native GUIs in Ada
- PolyORB: to build distributed systems using CORBA and the Ada distributed systems annex

AdaCore — AWS for WxWorks

From: AdaCore Press Center
Date: Wed, 25 Aug 2010
Subject: AWS available for Wind River's VxWorks
URL: http://www.adacore.com/2010/08/25/aws-vxworks/

AWS available for Wind River's VxWorks

Ada Web Server brings web-based connectivity and control to embedded applications

NEW YORK and PARIS, August 24, 2010 – AdaCore, a leading supplier of Ada development tools and support services, today announced the launch of Ada Web Server (AWS) for Wind River's VxWorks Real-Time Operating System (RTOS).

When used in conjunction with the GNAT Pro development environment, AWS enables developers to embed an Ada-based web server within any application, accessible through web browsers. With AWS for VxWorks, users can connect through a direct TCP/IP link to board-level applications for a variety of purposes, such as system control, configuration, and/or maintenance. The two-way interface is both lightweight and flexible.

In the past, AWS has been available on native operating systems, including Windows and Linux. VxWorks is the first RTOS supported by AWS, providing developers with a target-independent solution for creating user interfaces for embedded applications. Ports to additional embedded RTOS platforms are currently under development.

"Embedded applications used to be fairly self-contained and autonomous. This picture is changing rapidly, and a growing number now have a connection to the Internet or at least to a company's intranet," said Cyrille Comar, Managing Director, AdaCore. "AWS for VxWorks provides GNAT Pro customers with a very simple and integrated way to implement a user interface to their embedded devices through regular web browsers, bringing the power of modern web technology to the traditional embedded world."

"Wind River is pleased to see AWS support made available for the VxWorks platform," said Chip Downing, senior director of aerospace and defense at Wind River. "AdaCore continues to add advanced capabilities for VxWorks that expand the use cases for both products."

Key features of AWS:

- Web Parameters Module for retrieving forms or URL parameters and building an associative table for easy access
- Session Server that keeps client data from page to page
- SOAP support for developing web services and generating stubs/skeletons from a WSDL document
- Template parser, allowing the complete separation of web design from code
- HTTPS/SSL, for Secure Sockets based on the OpenSSL library
- Support for large servers and virtual hosting using dispatchers based on URI request methods or host names
- Server Push support
- Log Module to keep log file of all resources requested by servers
- Support for SOAP, SMTP, POP and LDAP protocols and the AJAX standard
- Client-side support using client API to retrieve any web page from a web page
- High-level services, including directory browser, status page for information, and web page service to build simple static page servers

About AdaCore

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

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

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

www.adacore.com

Adalog — AdaControl 1.12r4

From: J-P. Rosen <rosen@adalog.fr>
Date: Wed, 04 Aug 2010 09:36:58 +0200
Subject: Ann: AdaControl 1.12r4 (GPL2010) released
Newsgroups: comp.lang.ada

Adalog is happy to announce the release of AdaControl 1.12r4.

This is a minor release to 1.12r3 that just fixes a few bugs in unlikely corner cases, but the main improvement is that the executable versions are now provided for GNAT GPL2010.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 04 Aug 2010 14:51:12 +0200
Subject: Re: Ann: AdaControl 1.12r4 (GPL2010) released
Newsgroups: comp.lang.ada

[…]

This new version reached Debian unstable yesterday :)

[see also "AdaLog — AdaControl 1.12r3" in AUJ 31-2 (Jun 2010), p.89 —mp]

Midoan — Mika 1.2

From: Midoan Webpage
Date: Wed, 1 Sep 2010 [retrieved —mp]
Subject: Mika by MIDOAN Version 1.2 Release Notes
URL: http://www.midoan.com/download/current/ ReleaseNotes.rtf

[…]

Mika is an automatic test data generation tool for code written in a large subset of Ada 83, Ada 95 or Ada 2005. Mika uses genetic algorithms to generate inter-subprograms tests that will, by construction, exercise, during execution, the maximum possible number of branches, or decisions, in the code under test.

[…] What is new in this release

Version 1.2 (29 June 2010)

- Mika now allows MC/DC test data generation from Ada source code.

[…]

Releases History

Version 1.2 (29 June 2010)

- Improvements include:
- Tests to achieve MC/DC coverage of the code under test can now be generated automatically;
- The user can now specify the level of deepness of coverage desired;
- The handling of input dependent types, slices, loops etc is more clever : it is much less likely to lead to un-tractable tests data generation;
- The test data generation process has been improved for floating point entities.

Subset enlargements include:

- Input dependent types are accurately handled;
- Dynamic record default field expressions are accurately handled;
- Input dependent loop ranges are accurately handled;
- Conditional expressions are fully supported;

- Large integers handling has been improved;
- The valid attribute is supported.

Bug fixes include:

- Tests generation for rem and mod expressions is more powerful;
- Subprograms calls that returns unhandled expressions were not handled properly;
- View conversions with complex expressions were not handled properly;
- View conversions with long prefixes were not handled properly.

[…]

[see also "Midoan — Mika 1.1" in AUJ 30-2 (Jun 2009), p.79 —mp]

Ada Inside

Lunar lander CubeSat adopts SPARK

From: melampus <henssel@gmail.com>
Date: Thu, 10 Jun 2010 05:16:51 -0700 PDT
Subject: FYI -- Lunar lander project relies on SPARK programming language
Newsgroups: comp.lang.ada

Altran Praxis announced that its SPARK language has been selected by a new, NASA-funded US lunar mission. SPARK will be used to develop the software behind a CubeSat project being developed by a consortium comprising Vermont Technical College, Norwich University, St. Michael's College, and the University of Vermont.

EE Times EU =>

http://www.electronics-eetimes.com/en/lunar-lander-project-relies-on-spark-programming-language.html?cmp_id=7&news_id=222902326&vID=296

From: Pascal Obry <pascal@obry.net>
Date: Thu, 10 Jun 2010 18:19:54 +0200
Subject: Re: FYI -- Lunar lander project relies on SPARK programming language
Newsgroups: comp.lang.ada

[…] would be nice to know which Ada compiler will be used to generate the code for the target.

[…]

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Thu, 10 Jun 2010 21:01:17 +0300
Subject: Re: FYI -- Lunar lander project relies on SPARK programming language
Newsgroups: comp.lang.ada

[…]

An earlier article on CubeSat (Ada User Journal, September 2008, page 213) says that they use(d) SofCheck's AdaMagic Ada-to-C compiler, followed by Rowley Associates' CrossWorks C compiler.

From: Peter C. Chapin <chapinp@acm.org>
Date: Fri, 11 Jun 2010 10:22:31 -0400
Subject: Re: FYI -- Lunar lander project relies on SPARK programming language
Newsgroups: comp.lang.ada

[…]

I can speak on this issue as I am directly involved with this project. And yes, our intention is to compile the Ada to C using SofCheck's AdaMagic and then compile the C with CrossWorks. We have experience doing this with an earlier project and it works well. At the moment we are programming to the bare metal without the assistance of an operating system (although that might have to change at some point). SPARK is helpful here because the run-time support required for SPARK programs is extremely minimal due to SPARK's restrictions.

From: Bill Findlay <findlaybill@blueyonder.co.uk>
Date: Fri, 11 Jun 2010 15:34:03 +0100
Subject: Re: FYI -- Lunar lander project relies on SPARK programming language
Newsgroups: comp.lang.ada

[…]

Can you say why? Is there no Ada compiler that directly targets your CPU?

From: Peter C. Chapin <chapinp@acm.org>
Date: Sat, 12 Jun 2010 08:12:39 -0400
Subject: Re: FYI -- Lunar lander project relies on SPARK programming language
Newsgroups: comp.lang.ada

[…]

There is a GCC based tool chain supporting the MSP430. See:

http://mspgcc.sourceforge.net/

Presumably we could compile GNAT on top of this tool chain. However, Rowley's product gives excellent and "direct" support for the chips we are using: GUI debugging tools, simulation tools, IDE support, etc. Also Rowley's compiler is supported by third parties who have simple OS kernels for this platform. Our tool chain works "out of the box" after just making a few configuration adjustments to AdaMagic. We felt that there would be more problems trying to get GNAT to work.

I do not know of any Ada compiler that generates code directly for the MSP430.

SPARK implementation of the Skein algorithm

From: AdaCore Press Center
Date: Mon, 16 Aug 2010
Subject: Open Source SPARK Implementation of Skein Algorithm
URL: http://www.adacore.com/2010/08/16/spark-skein/

SPARKSkein reference implementation achieves success ahead of Crypto 2010 conference

NEW YORK, PARIS, and BATH, England, August 16, 2010 – Crypto 2010 Conference

Altran Praxis and AdaCore today announced a new reference implementation of the Skein algorithm, written and verified using the GPL 2010 Edition of the SPARK language and toolset. Skein is a cryptographic hash function and an entrant in the National Institute of Standards and Technology (NIST) hash function competition to design what will become the new Secure Hash Algorithm (SHA-3) standard. Such hash functions are used to compute short “digests” of long messages, and are one of the key building blocks of digital communication and cryptographic systems. Altran Praxis and AdaCore have open sourced the SPARKSkein reference implementation and made it available to the developer community via the Skein website. This is in time for the second SHA-3 candidate conference taking place at the Crypto 2010 conference, Santa Barbara, California, USA, from 23-24 August 2010.

The joint Altran Praxis and AdaCore project began as an informal experiment to demonstrate whether a hash algorithm like Skein could be realistically implemented in SPARK. The goals of the implementation were:

- Readability – to strike a reasonable balance of readability and performance.
- Portability – the team aimed for a single code-base that was portable and correct on all target machines of any word size and endian-ness, with no macros, ‘ifdefs’, or pre-processing of any kind.
- Performance – to ensure that the performance of the SPARK code would be close to or better than the existing C reference implementation.
- Formality – to prove at least type-safety (i.e., no exceptions) on the SPARKSkein code.

The team wanted to use the experiment to refute the claim that ‘formal is slow’ in programming languages.

The project concluded with notable success, proving that an algorithm like Skein can be written in a ‘formal’ language like SPARK without sacrificing readability or performance. Furthermore, portability was achieved. A single set of sources yielded identical results on more than fifteen platforms, covering a wide range of microprocessors and operating systems. In addition, SPARK’s type-safe nature allowed the project team to maximize compiler optimization with confidence.

Professor Stefan Lucks, a member of the Skein design team, said, “Speaking for the

Skein design team, we're very impressed by this work. SPARKSkein isn't just another implementation of Skein – essential properties of this implementation have been formally verified. It is stunning how the formal verification of the SPARK source code actually made us discover a flaw in our own reference C implementation.

Personally, I also find the SPARK code to be more readable than the equivalent C."

Rod Chapman, a Principal Engineer at Altran Praxis, led the SPARKSkein project.

He said, "This is an incredibly exciting project for us to be involved in. By open sourcing the code and donating our work, we hope to make a valuable contribution to the scientific community, as well as to showcase the capability of SPARK and its verification tools."

" Eric Botcazou, GCC expert and consultant at AdaCore provided the optimization and performance analysis for the project. He added, "The results clearly demonstrate that the SPARKSkein code is easy to understand and read, while the performance is at a comparable speed to the C code. This is solid evidence that any 'formal' code developed in this way doesn't have to be slow and impractical."

About Altran Praxis

Altran Praxis is a specialist systems and software house, focused on the engineering of systems with demanding safety, security or innovation requirements. Altran Praxis leads the world in specific areas of advanced systems engineering and innovation such as: ultra low defect software engineering, Human Machine Interface (HMI), safety engineering for complex or novel systems and tools/methods (such as SPARK) for systems engineering.

It offers clients a range of services including turnkey systems development, consultancy, training and R&D. Key market sectors are aerospace and defence, rail, nuclear, air traffic management, automotive, medical and security.

The company operates globally with active projects in the US, Asia and Europe. The headquarters of Altran Praxis are in Bath (UK) with offices in Sophia Antipolis, London, Paris, Loughborough and Bangalore. Altran Praxis is an expertise centre within, and wholly owned by, Altran which is a global leader in innovation engineering and employs 17,000 staff across the world.

www.altran-praxis.com

[…]

Indirect Information on Ada Usage

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

Job offer [Belgium]: Software Developer in Ada

[…]

Development (& designer tests) of basic software, signalling application software and monitor software (interface Basic Software / Application Software). Integration. Functional tests in lab.

Context/Tasks

Responsible of software development (design, integration and designer tests).

Functional reporting to project leader and Product leader.

Profile

- Degree: Master (4 or 5 years curriculum)
- Experience: 4 or 5 years of experience in SW design and programming - Experienced in real time embedded SW and basic SW.

Technical Skills

- Ada programming language.
- Functional SW specification (Teamwork tool).
- Configuration and Modification management (Clearcase and Clearquest tools).
- Notions in communication, safety protocols.

[…]

Job offer [United Kingdom]: Permanent Software Engineer

[…]

A permanent staff opportunity for a proven ADA / ADA95 Software Engineer [sic —mp].

The MCSP programme is a £750m update to the Royal Navy's Merlin anti-submarine and anti-surface helicopter system. The programme includes technology enhancements to the aircraft, ground preparation/analysis systems, training systems and support infrastructure.

This role is integral to the successful performance of our client's flagship programme in the UK; the candidate will be expected to adhere to challenging schedules.

Specific Job Description:

The position requires strong understanding and sound application of the Software Engineering principles and practises and a general knowledge of systems engineering and Test & Integration and Validation disciplines.

The position requires a software developer who designs, develops, documents, tests, and debugs applications software that contains logical and mathematical solutions to business/mission problems or questions in computer language for solutions by means of data processing equipment. Applies the appropriate standards, processes, procedures, and tools throughout the development life cycle.

Corrects program errors, prepares operating instructions, compiles documentation of program development, and analyses system capabilities to resolve questions of program intent, output requirements, input data acquisition, programming techniques, and controls.

[…]

The Software Developer will be responsible for design, development and unit test of elements of software on the MCSP programme. This will include:

- Plan own work within defined constraints with the assistance of engineering and technical mentors
- Provide overall technical support and assistance to engineers
- Design using object-oriented methodologies and a UML toolset (Artisan Studio)
- Code development in ADA 95 (using Greenhill's AdaMulti)
- Documentation of developed software
- Unit testing (using ADATest)

[…]

Required Skills:

ADA Programming Experience

Development with C/C++/Java

Ability to adhere to a process within a Software development environment

SC Clearance will be necessary to start in this role

Job offer [United Kingdom]: Aerospace - Software Engineer

1. The consultants must have the Aerospace/Aerospace domain experience.
2. Incumbent should be a qualified Graduate/ post graduate (Mechanical, marine) engineering.
3. The candidates should be technically good at

- Software Engineering,
- DO178B,
- DAL A,
- combination of C, C++, C#, Ada, UML,
- Requirements Management and Design Management Skills

Experience in Years: 11+

Job offer [Spain]: Analyst Programmer

[…] We are searching for an analyst programmer with the following profile:

Education: graduated in Computer Science, Telecommunication or Electronic Engineering

Professional experience: minimum 3 years

[…]

Programming languages:

-ADA 95 [sic —mp]

-Assembler for Power PC processors

Software development:

- Development of object-oriented software
- Techniques for software testing
- Development of real-time systems
- Development of drivers for hardware devices

Productivity suites: Word, Excel, Power Point, etc.

Tools:

- Tools for version control
- Tools for requirement management
- Tools for software testing and documentation

Desirable knowledge:

- Development standards: DOD/STD/2167, RTCA/DO-178B
- Knowledge of the architecture of Power PC microprocessors
- Real-Time operating systems (Integrity, VxWorks, etc.)

Experience in the following development environments:

- ADAMULTI (GreenHills)
- RHAPSODY (IBM/Telelogic)
- ACCUREV (Accurev Inc.)
- DOORS (IBM/Telelogic)

Experience: 3-5 years

[translated from Spanish —mp]

Job offer [United States]: Real-Time Embedded Software Engineer

Currently we are looking for multiple Real-Time Embedded Software Engineers for our client […]

These opportunities are long term contracting positions […]

The Real-time Embedded Software Engineers should be experienced and skilled in the following:

- 5+ years of real-time embedded software development experience
- Knowledge of DO-178B Level A software development standards
- Ability to develop software requirements from customer inputs
- Understanding and translation of software requirements into a software design.
- Experience in high-level language such as C, C++ or Ada
- Reviewing/evaluating software products produced by peers.
- Skills in performing low level and high level tests on the software and integrating the software with the end item hardware.
- Experience with one or more of the avionics communication buses including AFDX, ARINC-429, CAN, MIL-STD-1553.
- Use MS Office products (Word, Excel and Access) and Telelogic DOORS to assist in the development of software product and documentation.

Job Requirements

- Real-Time Development
- C, C++, or Ada
- Avionics communication buses including AFDX, ARINC-429, CAN, MIL-ST-1553
- Telelogic DOORS
- DO-178B Level A software development standard

Ada in Context

Ada, UML and code generation

From: Yannick Duchêne
<yannick_duchene@yahoo.fr>
Date: Tue, 31 Aug 2010 12:12:43 +0200
Subject: Ada and UML
Newsgroups: comp.lang.ada

Hi all,

A few days ago I get to have a look back at UML after a topic linking to an assessment stating graphical representations may help in design/application validation.

Someone argued there is a lack of semantic with UML which dismiss its usage; one topic I agree on, and after some reading to be sure, it seems indeed, there does not have a clearly defined semantics […]

Another topic I often read/heard about Ada and UML, is that it is too much oriented toward Java/C++, lacking as an example, handling of class and package orthogonality which is typical of Ada.

I've just discovered today it seems there is a kind of package visibility with UML expression of classes, which could match Ada concept of package: the visibility modifier ~ (the Tilde sign) which seems to mean "private at package level".

This should be sufficient for Ada. Nevertheless, there is nothing like "friend" or "protected" in Ada, as visibility is only handled by packages, so this visibility modifiers could just be disallowed. Do you think "~" is OK to express visibility handled by packages ?

For you who tried Ada and UML, did you noticed some others unsolvable matters to express Ada package in UML?

[…]

P.S. As the purpose of UML is clearly underspecification, may be the question of this thread does not really matters, as Ada is acting at implementation level.

From: Yannick Duchêne
<yannick_duchene@yahoo.fr>
Date: Tue, 31 Aug 2010 13:05:43 +0200
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

[…]

Also, the Extend relationship, could be used to express the Ada's child package relationship. But I am not sure this is valid UML to use the extend relation between two UML packages.

And what about the Composition relationship to express Ada's nested package relationship ?

From: Matteo Bordin
<matteo.bordin@gmail.com>
Date: Tue, 31 Aug 2010 09:34:11 -0700 PDT
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

> […]

> To be more exact, UML Name Space should be used instead of UML Package, as Ada's package are both package and name-space at a time. […]

Hello Yannick,

UML semantics can be found in the UML metamodel superstructure at http://www.omg.org/spec/UML/2.3/Superstructure/PDF/. As you said, UML has no formally specified semantics - but this is also true for most languages we use everyday (including Ada): I think it is really unfair to keep bashing UML for not having a "formal semantics" while basically all languages we use suffers of the same problem. The only exceptions I am aware of are SPARK and some languages of the synchronous family (Lustre, Esterel, …).

Mapping Ada on UML means treating UML as graphical coding: in my personal opinion this defeats the main purpose of UML - i.e. being a platform-independent language. Ada profiles for UML surely exists, but they make UML models implicitly not implementable on any other programming language. I personally find language-specific UML profiles a total non-sense. Having said that, it is unfortunate that an incestuous relationship between UML and Java/C++ exists.

Anyway, one reasonable approach to represent Ada packages in "standard" UML is to use a non extensible (isLeaf=true) UML singleton class. Package-level variables, constants and non-primitive operations can be mapped as static features (isStatic=True) of the singleton class. Types declared within the Ada package can be represented as nested non-singleton classes contained in the singleton class above (a UML nested classifier is equivalent to Java/C++ nested classes). Ada child packages can be mapped as nested, non-extensible singletons. As you can see, this representation does not sound very natural from a UML point-of-view, but covers the Ada semantics.

The extend relationships relates to UML types only: in UML, a Package is not a type, so it cannot be extended in the UML sense. You also cannot directly use UML Namespace because it is an abstract semantic element (an abstract metaclass).

[…]

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Wed, 01 Sep 2010 01:14:21 +0200
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

[…]

> I personally find language-specific UML profiles a total non-sense. Having said that, it is unfortunate that an incestuous relationship between UML and Java/C++ exists.

I have never talked about UML profile ;) I talked about meaning of notational elements and attempted to express them in terms of Ada construct meaning.

There is no worry about stating UML Namespace better match Ada's package than UML Package does.

This is not an UML profile. Except may be for restriction like "only use the ~ visibility modifier".

But this could be as well a required UML subset to match some design criteria, just like there are Ada subsets (Ravenscar, SPARK, and may be others).

Also note that an Ada profile and UML profile does not stand for the same thing: UML profile means possibly adding notational components to UML via prototypes [stereotypes —mp], while an Ada profile means strict subset of Ada, don't even think about adding anything. These are two opposite!

I was talking about UML subset. This should match both expectations, yours and mine. Subset may enforce clearer meaning (as an example, avoiding to have multiple notation to express the same thing), while not extended the notation, so that it is not tied to a particular language (I love your comment about it).

I agree with the criteria (UML should not be tied to any design language).

> Anyway, one reasonable approach to represent Ada packages in "standard" UML is to use a non extendible (isLeaf=true) UML singleton class.

An UML class to represent a package ? Can you tell more ? What is the difference with the name-space choice ?

[…]

I was a bit long with this reply, let go for a summary: give UML the semantics of the target language (implies this choice is made a-priori), use notation restrictions as much as needed […]

From: Matteo Bordin <matteo.bordin@gmail.com>
Date: Wed, 1 Sep 2010 01:20:30 -0700 PDT
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

[…]

One of the goals of UML is to permit UML models to be implemented on several run-time platforms: for this to be possible, either you create a language which is the union of all possible computational models, or you leave semantic variation points to permit model transformation (code generators) to inject platform-specific semantics following a precise pattern. UML chose the second option. The core idea is that, by modeling your execution platform, you can "instruct" your code generator to generate code with a fully specified semantics for a precise target platform starting from an abstract model. This permits to re-use the same model for different target platforms by just changing code generator and platform model and to perform high-level verifications without including "pollution" from the underlying run time.

> […]
>
> Also note that an Ada profile and UML profile does not stands for the same thing: UML profile means possibly adding notational components to UML via prototypes, while Ada profile means strict subset of Ada, don't even think about adding anything. These are two opposite!

A UML profile can be purely constrictive. For example, a UML profile may state that you cannot use Interfaces. This is expressed in OCL as Interface.allInstances()->size() = 0. No need for stereotypes. From this point of view, a UML Profile provides the same service as an Ada profile. A UML profile can also be used to define modeling standards (mirroring the notion of coding standard).

> […]
>
> An UML class to represent a package? Can you tell more? What is the difference with the name-space choice?

The problem is that Ada Packages can have non-primitive operations and variables, but UML Packages can't. The only place where you can put an operation in UML is inside some behavioural classifiers (Class, Interface, DataType, …). So, if you want to represent Ada packages with non-primitive operations, the only option is to map the Ada Package on a UML class. On the other side, if your Ada program is fully object-oriented (only primitive operations), you can use UML packages to map Ada packages and UML classes to map Ada tagged/record types.

In UML a Namespace is an abstract metaclass (a sort of abstract grammar rule), which is extended by concrete metaclasses. BTW, UML Class inherits from NameSpace: in UML a Class is a Namespace.

> I was a bit long with this reply, let go for a summary: give UML the semantic of the target language […], use notation restrictions as much as needed […]

I disagree ;-) This approach means using UML at the same abstraction level of a traditional programming language. A desirable modeling approach based on languages of the UML family is to separate what is platform-independent from what is platform-specific to permit the reuse of platform-independent models. Let's make an example about tasking.

You can have a platform-independent UML model where you identify "unit of concurrency".

And another, platform-specific UML model where you describe the low-level implementation in terms of (Ada, C, ARINC-653, ...) tasks.

Then you allocate the platform-independent UML model on the platform-specific one by binding elements of the first model on elements of the second one.

This way you can reuse the platform-independent model on several different run-time architectures by simply providing new platform-specific models and new bindings.

You may get more information on this approach by reading tutorials on MARTE (omgmarte.org).

From: Simon Wright <simon@pushface.org>
Date: Wed, 01 Sep 2010 19:44:09 +0100
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

[…]

I was going to draft a reply but Matteo has said almost all that I would have done.

I'd add that:

1 I wouldn't expect every possible UML construct to be translatable using this model. A given vendor/toolset might

well specify limitations, presumably as a profile.

2 I wouldn't expect to be able to tweak the UML so as to generate any arbitrary target language (Ada) construct.

3 The converse of 2: I wouldn't expect to be able to represent any arbitrary target language (Ada) program in UML so as to be able to regenerate it.

From: Matteo Bordin
<matteo.bordin@gmail.com>
Date: Thu, 2 Sep 2010 23:54:59 -0700 PDT
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

[…]

Papyrus 1.11 includes an UML2-to-Ada2005 code generator. Very primitive, written just as a counter example for some FUD about Ada (Ada can't do OO, Java is more expressive than Ada and other idiocies). BTW, it is written in Acceleo, see

http://www.papyrusuml.org/scripts/home/publigen/content/templates/show.asp?P=131&L=EN&ITEMID=15

[…]

From: Anonymous
Date: Tue, 31 Aug 2010 23:20:41 +0200
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

> Ada profiles for UML surely exists, but they make UML models implicitly not implementable on any other programming language.

There are also platform-independent profiles with semantics. I use one every day called xtUML. And translate the same models to both C++ and Ada-95. SPARK is under development.

The xtUML profile is used for ANALYSIS models. We do NOT code C++ or Ada in UML. We do analysis. Then the translation into code is just something mechanical.

From: Anonymous
Date: Thu, 02 Sep 2010 22:33:48 +0200
Subject: Re: Ada and UML
Newsgroups: comp.lang.ada

[…]

The thing we use is Executeable UML (xtUML).

I use Bridgepoint. For C++ generation we use a model compiler from Mentor for Ada we have built one from scratch.

[…]

The model compiler is in many cases quite as capable as I to analyze the model and decide if instances of one class should be allocated dynamically or created once and for all in an array. If traditional pointers or some other mechanism should be used for relations and so on.

> Just think that in Ada, you may have for example at least, where polymorphism is required, two ways to do (when feasible, sometime only one will applies) : static (via generics) and dynamic (via tagged or interface) polymorphism. How can a system know it should use this or that ? This requires a human analysis, to say "well, this can be statically known, so let's use static polymorphism" or else the opposite (just an example).

There are of course such examples. But remember that when doing the translation from model into code you have the whole model of your system available in a database and can can look at things from a much better perspective than when you are coding a single file.

An easy to understand example is relations. If you draw a simple line in the class model you state that there is a relation. But the model compiler can look at every single line of code in your system to see if you use it (navigate) or not. If it sees that you never use it, it won't have to generate the code to maintain it and if it finds you only navigate one direction it can add the code needed for it to be one way only. (Now many hardcore coders will feel lost, what does he mean one-way only, aren't they all? No in xtUML they are always bidirectional, composite, aggregate and navigability as in UML does not exist).

> Your point makes me think about the so called "Executable UML".
>
> Unfortunately, I do not know anything about it. I am just pretty sure this cannot be automated.

I think it can. We are in the process of completing a pretty large system. It consists of ~650 classes of which probably around 200 have state machines. Generated system is > 3000 Ada packages.

> If you wan to tell more... just do :)

Sorry, not allowed to give details. Call Mentor, ask for when John Wolfe or Cortland Starret can be in your area, you want to have a look at their system modelling products.

Simon mentioned distribution. We use that. In our system we can mark classes and services as distributed. Then the model compiler generates definition files for the broker and in the classes it generates all the interface code needed to distribute things when it is updated.

So, do I have anything to complain about? Yes

- Price of the tool.
- Your first project will fail. Try to combine this knowledge with the prices when you seek financing.
- Availability of model compilers
- Constantly having to defend the method
- Performance of the model compiler. Coming from a world where a recompile takes perhaps as long as 15 seconds entering a world where regeneration/recompilation takes three hours is sometimes (read always) hard. The generated code is probably faster than anything I could have written myself.

Regarding SPARK I know just about nothing. I know there has been work on modifying the Ada compiler to produce SPARK code. Do not know how far that has gone or what efforts need to be done.

There has also been work on generating VHDL. Just think about it. Model a system and end up with hardware specific for your problem.

One strange thing the non-believers tend to bring up is that software development needs to be fun. My comment is that I have now done this for six years. It has been the most interesting years of my twenty as a professional software developer. The productivity you can reach is beyond anything I had imagined before.

From: Simon Wright
<simon@pushface.org>
Newsgroups: comp.lang.ada

Subject: Re: Ada and UML
Date: Thu, 02 Sep 2010 07:14:42 +0100
[…]

> Your point makes me think about the so called "Executable UML". […]

Bridgepoint presently from Mentor Graphics,

http://www.mentor.com/products/sm/model_development/

iUML from Kennedy-Carter, http://www.kc.com/ (see 'Products')

In both cases the actions (the part that does things) are written in a tool-specific programming language, and one of the jobs of the translation engine is to turn this into compilable code.

Of particular interest to you, maybe, see

http://www.kc.com/PRODUCTS/iccg/adacode.php where you are offered two Ada code generators, one for Ada 83 and one for SPARK (I don't know how the SPARK generator gets 'interesting' pre/post/invariant conditions in there; obviously the analyst is going to have to provide them).

Sharing the body of a generic

From: Peter C. Chapin
<chapinp@acm.org>
Date: Mon, 26 Jul 2010 20:51:25 -0400
Subject: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

It has been my understanding that Ada's generics are designed in such a way as to allow implementations to share the code of a generic body across all the instantiations. I understand that doing this might involve a performance penalty relative to creating independent code for each instantiation. However, I can see that there are cases where such sharing would be desirable.

Is my understanding still accurate (was it ever accurate), for example even with Ada 2005?

[…]

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Tue, 27 Jul 2010 03:51:59 -0700 PDT
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

Allowed yes, required no.

From: Christoph Grein <christoph.grein@eurocopter.com>
Date: Mon, 26 Jul 2010 23:55:40 -0700 PDT
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

As far as I understand, GNAT replicates, RR shares generic code.

(Don't know about IBM (former Rational).)

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 27 Jul 2010 13:29:26 +0200
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

> As far as I understand, GNAT replicates, RR shares generic code.

> (Don't know about IBM (former Rational).)

My understanding is similar; indeed, Janus/Ada is the only compiler that shares generics. Unfortunately, no compiler offers the option to choose; this is an implementation decision.

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Tue, 27 Jul 2010 17:10:16 +0300
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

If I have understood correctly, Irvine's Ada compiler supports shared generics *and* provides a compile time option for this.

On the other hand, Janus/Ada doesn't provide an option. With it, shared generics are always used.

From: Anonymous
Date: Tue, 27 Jul 2010 20:06:44 +0000 UTC
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

The Replication versus the Sharing of Generic code was initially based on the Ada Optimize pragma statement. That is, when the users define the option of "Time" the Generic code would be replicated, but the "Space" option would cause the compiler to share the code body. Not using the pragma statement or the Ada 95 "off" option allowed the designer to set an implementation default.

Now as for GNAT it uses the optimization based on the GCC back end (-OX where X in 0 .. 4). GNAT still preforms a syntactical check of the Optimize pragma statement, then treat's the statement as a comment like a number of other built-in Ada pragma statement. Which allows GNAT to replicate code and let the GCC handle rather its switches to shared or not. And at this time GCC does not understand the Ada's concept replication versus the sharing code for optimization.

From: Keith Thompson <kst-u@mib.org>
Date: Tue, 27 Jul 2010 17:55:18 -0700
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

I don't believe the definition of pragma Optimize was ever that specific; as far as I know, it was always intended merely as a vague hint.

Here's the description from the Ada 83 reference manual:

> OPTIMIZE takes one of the identifiers TIME or SPACE as the single argument. This pragma is only allowed within a declarative part and it applies to the block or body enclosing the declarative part. It specifies whether time or space is the primary optimization criterion.

Using it to control generic code sharing would certainly be reasonable, but it's not required.

From: Anonymous
Date: Wed, 28 Jul 2010 11:16:24 +0000 UTC
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

Besides Ada 83 chapter B on pragmas, a little more detail can be found in 10.3, 10.6, 11.6.

There are many forms of Optimizations. Sharing vs replicated and removing dead or unused code are some of the easiest to implement. Altering algorithms and expressions can be costly.

If the optimization criterion is set to "Space" then using shared generic code reduces the memory but may increase time for the program to execute those routines. As well as when the criterion is set to "Time" replicated code may be faster at the cost of using more memory. So, the Ada Optimize pragma statement does not limited it self to only expressions it also includes the way generic codes are generated and used.

For Ada 95 RM 2.8 Pragmas

> 27 A pragma Optimize takes one of the identifiers Time, Space, or Off as the single argument. This pragma is allowed anywhere a pragma is allowed, and it applies until the end of the immediately enclosing declarative region, or for a pragma at the place of a compilation_unit, to the end of the compilation. It gives advice to the implementation as to whether time or space is the primary optimization criterion, or that optional optimizations should be turned off. It is implementation defined how this advice is followed.

also check out D.12 Other Optimizations and Determinism Rules

From: Markus Schoepflin
Date: Wed, 28 Jul 2010 10:42:32 +0200
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

And Ada compiler is certainly not required to do anything but syntax checking for pragma optimize, but at least they had code sharing in mind when specifying the pragma.

From the 2005 AARM:

> 27.a Implementation defined: Effect of pragma Optimize.
>
> 27.b Discussion: For example, a compiler might use Time vs. Space to control whether generic instantiations are implemented with a macro-expansion model, versus a shared-generic-body model.

From: Peter C. Chapin <chapinp@acm.org>
Date: Tue, 27 Jul 2010 18:23:03 -0400
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

Thanks for all the replies to my question. I want to emphasize that my interest is not so much in what is done by current compilers but rather what is allowed by the standard. Is the standard (even the latest standard) written in such a way as to make a shared implementation of generic bodies possible? It sounds like the answer is yes.

[…]

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 2 Aug 2010 21:38:57 -0500
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

I have been very vigilant to preserve the ability for generic sharing in the Ada Standard. (It's something that would be very easy to lose because of some obscure combination of features.) I can't say for sure that I have kept every such case out of the Standard, but there is agreement that at least limited sharing (when the parameters are "similar enough") ought to be a permitted implementation. "Universal sharing" (where there only one body for each generic), as used in Janus/Ada, is more controversial (but is still allowed by Ada 2005).

From: Robert A Duff
<bobduff@shell01.TheWorld.com>
Date: Tue, 03 Aug 2010 10:31:43 -0400
Subject: Re: Sharing generic bodies across instantiations.
Newsgroups: comp.lang.ada

[…]

Like Randy, I am in favor of retaining the ability to share generic body code (universal or otherwise) in Ada. I am not in favor of trying to make universal sharing efficient at run time -- that's not possible. It's a trade-off -- it may make compile time faster, at the expense of run time.

It is not feasible to share generic spec code.

On fixed-point types

From: Peter C. Chapin
<chapinp@acm.org>
Date: Sun, 15 Aug 2010 17:43:43 -0400
Subject: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

I have a need to work with ordinary fixed point types. I've read in the reference manual about them (section 3.5.9, for example), and some other things. I still have a few questions. I thought I'd start with one here.

Consider this example:

```
type Angle_Type is delta 0.0005
    range -3.1416 .. 3.1416;
Angle : Angle_Type;
...
Angle := Angle_Type'First;
while Angle < Angle_Type'Last loop
  -- Work with Angle here.
  Angle := Angle + Angle_Type'Small;
end loop;
```

Aside from Angle_Type'Last does the loop above reliably visit every possible value of Angle_Type? By "reliably" I mean "has the desired effect on all possible correct Ada implementations." I guess my question is: does Angle + Angle_Type'Small always advance Angle to the next machine number in the type?

I'm using GNAT GPL 2010 and in the example above GNAT appears to be using 2**(-11) for Angle_Type'Small. It thus uses something over 12,000 values for the type. The code above is for a test program; it is my intention to exercise every possible value.

As an aside it appears as if Angle_Type'First is actually slightly less than the -3.1416 mentioned in the type definition. My reading of the Ada standard is that this is okay. Specifically I should be getting a multiple of Angle_Type'Small that is closest to the mathematical value of -3.1416. Apparently the closest multiple is just "to the left" of the number I specified. This actually works out to be slightly awkward in my case, but that's a problem I can probably figure out.

[…]

From: Peter C. Chapin
<chapinp@acm.org>
Date: Sun, 15 Aug 2010 20:46:03 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

> The first thing I see here is that 0.0005 doesn't evenly divide 2*3.1416; is that going to be a problem?

In theory it is not a problem. My reading of the Ada Reference Manual tells me that the implementation chooses a value of Small (the actual difference between represented numbers) that is no more than the specified delta. Note that Small must be a power of 2, but can be controlled with a representation clause. In my case GNAT appears to be using 2**(-11) which is 0.0004883. Implementations can choose a smaller Small if they want.

The values in the type then become all the multiples of Small "between" the endpoints of the specified range. However, those endpoints are converted to the nearest multiple of Small (unless they fall exactly between two multiples in which case the value closer to zero wins). You can probably tell I've been reading the manual. :)

I'm not sure which multiple of 2**(-11) are involved here exactly but it appears that the mathematical value of -3.1416 is rounding down... meaning more negative. I'm not too worried about that right now. I just wondered how I could visit all the values of the type in a loop.

Probably I should use extra digits in the definition of the range to communicate my intentions to the compiler more precisely. Thus

```
type Angle_Type is delta 0.0005
  range -3.1415926535 ..
         3.1415926535;
```

That way when the compiler decides how to round the end points of the range, it will definitely do the right thing.

From: Simon Wright
<simon@pushface.org>
Date: Mon, 16 Aug 2010 09:57:57 +0100
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

> Why not use Ada.Numerics.Pi?

We had a problem where a subcontractor had defined pi to be 3.14159.. not quite good enough for a 32-bit Float.

From: Peter C. Chapin <chapinp@acm.org
Date: Mon, 16 Aug 2010 07:29:13 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

> […]
>
> Why you do not specify:
>
> **for** Angle_Type'small **use** 0.0005;

I don't think I'm too concerned about the actual value of Small used. In fact, I'd like to let the compiler choose so that it can optimize the code better. Isn't it the case that using a power of two allows for more efficient code generation for certain mathematical operations? I'm not sure, but I seem to recall reading that somewhere. If that is true, then I want that. My machine isn't very fast.

My original question wasn't about how to force the type to use a Small that I want, rather it was about how can I be sure to visit every value of the type in a loop for test purposes.

From: Stephen Leake
<stephen_leake@stephe-leake.org>
Date: Mon, 16 Aug 2010 22:03:08 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

> […] Isn't it the case that using a power of two allows for more efficient code generation for certain mathematical operations?

Marginally. The representation is an integer with a scale = 1 / angle_type'small. The scale is only used when converting to other numeric types.

> I'm not sure, but I seem to recall reading that somewhere. If that is true, then I want that. My machine isn't very fast.

Get it logical first, then optimize.

> My original question wasn't about how to force the type to use a Small that I want, rather it was about how can I be sure to visit every value of the type in a loop for test purposes.

Then you need to make the loop step by 'Small.

From: Robert A Duff
<bobduff@shell01.TheWorld.com>
Date: Mon, 16 Aug 2010 10:28:29 -0400

Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

I'd expect "*" and "/" to be faster for binary smalls, but I don't think it makes any difference for "+" and "-".

Multiplying angles by angles doesn't make much sense…

[…]

From: Peter C. Chapin <chapinp@acm.org>
Date: Mon, 16 Aug 2010 16:31:07 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

Probably true. On the other hand I anticipate needing other fixed point types before I'm done so I think it would serve me well to get used to using binary smalls.

[…]

From: Robert A Duff <bobduff@shell01.TheWorld.com
Date: Mon, 16 Aug 2010 19:01:51 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

Well, you can use binary smalls when that makes sense -- it doesn't mean you have to get used to using them when it doesn't. For calculations involving money, for example, you should look at decimal fixed point types ("type ... is delta ... digits ...").

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 16 Aug 2010 15:01:20 +0200
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…] BTW, in your example you don't visit the last value. […]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 16 Aug 2010 14:53:16 +0200
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

```
type Angle_Type is delta 0.0005
    range -3.1416 .. 3.1416;
  Angle : Angle_Type;
begin
  Angle := Angle_Type'First;
  while Angle < Angle_Type'Last loop
    ... -- Work with Angle here.
    Angle := Angle_Type'Succ (Angle);
end loop;
```

From: Peter C. Chapin <chapinp@acm.org
Date: Mon, 16 Aug 2010 16:28:27 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

That's cool. I didn't realize I could use Succ with a fixed point type.

I thought that was only for discrete types... or is a fixed point type a kind of discrete type (it would make sense for it to be). I guess I can look that one up. :)

From: Adam Beneschan <adam@irvine.com>
Date: Mon, 16 Aug 2010 13:58:05 -0700 PDT
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

It's not a discrete type. However, starting with Ada 95, 'Pred and 'Succ have been defined for fixed-point and floating-point types. For fixed-point types, 'Succ is defined to have the same result as adding the 'Small of the type. 3.5(22-24).

From: Peter C. Chapin <chapinp@acm.org>
Date: Sun, 15 Aug 2010 21:03:07 -0400
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

I need to do real number computations on a microcontroller without a floating point unit. I believe the various numeric quantities that I need can be normalized into fairly narrow ranges so it seemed to me that this is a perfect application of fixed point types. The compiler stores the numbers as integers representing the appropriate multiple of Small.

I'm not using GNAT on the microcontroller but if the Ada compiler I am using does the same thing as GNAT it will be able to fit Angle_Type as it is currently defined into a 16 bit word and compute with it using integer instructions. In my environment that is essential.

At the moment I'm looking at implementing some basic trig functions such as Sine and Cosine taking Angle_Type parameters. My test program compares the computed results against the much higher precision values from GNAT's normal numeric library. I can thus assess the overall accuracy of my implementation. I want to check every possible Angle_Type value to be sure there are no surprises. For example right now I get Constraint_Error when I attempt to compute the Sine of Angle_Type'First. This is apparently because Angle_Type'First is actually slightly less than -Pi and my simplistic implementation can't cope with that.

[…]

BTW, I will probably need a smaller delta than 0.0005 eventually (I think). But I imagine changing that will be easy enough once I understand what I'm doing. Alas, using too small a delta will probably force the compiler to use 32 bits to store an Angle_Type value and that's an undesirable thing on my machine.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 16 Aug 2010 03:03:59 -0700
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

More generally, Ada allows the programmer to specify the 'Small with an attribute definition clause, so theoretically the exact solution to Peter's problem would be:

```
Epsilon : constant :=
    Ada.Numerics.Pi / 2 ** 15;
type Angle is delta Epsilon
    range -Ada.Numerics.Pi ..
          Ada.Numerics.Pi;
    for Angle'Small use Epsilon;
```

This would, in theory, create the desired mapping from the integer range [-2**15 .. 2**15] to the real range [-Pi .. Pi] without wasting any machine numbers.

However and unfortunately, ARM 3.5.9(21) allows an implementation to reject any 'Small that is not a power of two (and, in particular, decimal fixed-point types). I'm not sure whether GNAT supports arbitrary Smalls; from the doc, it seems that it actually rounds them down to a power of two no smaller than 2**(-63).

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Tue, 17 Aug 2010 07:35:04 -0700 PDT
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

[…]

2**(-63)! That should be OK in this application, then.

The real question for Peter is, what does his target compiler do? If it does as GNAT does and allows a delta and matching small of Ada.numerics.pi / 2 ** 15 then it makes sense to work in your scheme, since literal values in the code and values for debug will be in radians. If it only allows binary smalls, then (I would) work with delta 2 ** (-15) range -1.0 .. 1.0, ie signed fractions of pi, and apply the scaling factor as required.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Tue, 17 Aug 2010 17:51:38 +0200
Subject: Re: Question about ordinary fixed point types.
Newsgroups: comp.lang.ada

> 2**(-63)! That should be OK in this application, then.

Yes but I suspect it requires 64-bit hardware. I don't think this would work on a 16-bit microcontroller.

Bit numbers in packed arrays of Boolean

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Tue, 31 Aug 2010 07:14:48 -0400
Subject: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

I was pleasantly surprised to discover that given this code compiled with GNAT 6.2.1 for an Intel processor:

```
subtype Bit_Index_16_Type is
    Integer range 0 .. 15;
type Bit_Array_16_Type is array
    (Bit_Index_16_Type) of Boolean;
  pragma Pack (Bit_Array_16_Type);
  for Bit_Array_16_Type'Size use 16;
function To_Bit_Array is new
   Ada.Unchecked_Conversion
   (Source => Interfaces.Unsigned_16,
    Target => Bit_Array_16_Type);
Word : constant
   Interfaces.Unsigned_16 :=
   2#1000_0000_0000_0000#;
Bits : constant Bit_Array_16_Type :=
   To_Bit_Array (Word);
```

the index of Bit_Array_Type indexes the bits of Unsigned_16 in little-endian order. That is, Bits (15) = 1 (most significant bit), Bits (0) = 0 (least significant bit).

LRM 13.9(5) says Bit_Array_16_Type and Unsigned_16 have the same representation, but it does not specifically address the index order.

Is this bit order required by some other clause? Do other compilers follow it?

I don't have access to GNAT for a big-endian processor; can anyone confirm what happens there?

Ideally, there would be a way to force the other bit order, as 'Bit_Order does for records.

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Tue, 31 Aug 2010 13:34:00 +0200
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

> LRM 13.9(5) says Bit_Array_16_Type and Unsigned_16 have the same representation, but it does not specifically address the index order.

At least ARM 3.6 Array Types does not states anything. But as you used Unchecked_Conversion, it is unlikely to be portable or this result could be enforced.

Let say this is the result of a compiler with descent default behaviors.

ARM 13.9 Unchecked Type Conversions says:

11.a/2 Implementation defined: The effect of unchecked conversion for instances with nonscalar result types whose effect is not defined by the language.

And do not believe the reference defines anything for an array of 16 bit-layout booleans.

Possibly depends on how the index is interpreted: as an index in the polynomial representation of the word value or as a bit index ? Both would be legitimate for a compiler.

[…]

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Tue, 31 Aug 2010 11:13:56 -0700
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

Packed arrays of Boolean have been around since Ada 80, and prior to Ada 95 were the only way to access individual bits. Note that the Boolean operations "and", "or", "xor", and "not" may be applied to such arrays.

Unfortunately, the ARM does not specify how indices are mapped to individual bit positions, so this is entirely compiler dependent. Typically compilers map the lowest index onto the lowest bit number supported by the H/W, so this depends on the platform, but the ARM allows any mapping from indices to bit positions. As a result, any use of packed arrays of Boolean to access specific bits is not portable.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Tue, 31 Aug 2010 15:34:35 +0300
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

> I was pleasantly surprised to discover that given this code compiled with GNAT 6.2.1 for an Intel processor: […]

Well, not quite, as I understand that point in the LRM. It says that (under several conditions) the *value* of the object Bits has the same representation as the *value* of the object Word.

It makes no sense to say that (or even to ask if) the two types Bit_Array_16_Type and Unsigned_16 have the same representation, because their value-sets are disjoint, so we cannot ask if they represent "the same value" in the same way.

> but it does not specifically address the index order.

Right. So we do not know at which index in Bits a given bit from Word ends up. I believe they could even be in some scrambled order, not necessarily 0 .. 15 or 15 .. 0.

> Is this bit order required by some other clause?

I believe not. This is one reason why, when I need to access specific bits in a word, I prefer to use Unsigned_xx types and their masking and shifting operations, not packed arrays. I use packed arrays only when the index order does not matter (or when portability does not matter, which is basically never).

From: Randy Brukardt <randy@rrsoftware.com>
Date: Thu, 2 Sep 2010 15:09:27 -0500
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

>> Is this bit order required by some other clause?

[…]

I don't think so, either. I've generally used a record type (with a record representation clause) if I care where the bits are. This would be a bit annoying in this case (16 distinct components), but often I've found that you don't really need the conversion in the first place if you have an appropriately represented record. […]

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Tue, 31 Aug 2010 14:41:10 +0200
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

> Well, not quite, as I understand that point in the LRM. It says that (under several conditions) the *value* of the object Bits has the same representation as the *value* of the object Word.

I suppose, for scalar types only, isn't it ? An exact reference would be welcome anyway.

Can you recall one please ?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 31 Aug 2010 15:08:02 +0200
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

RM 13.1(//2):

"The representation of an object consists of a certain number of bits (the size of the object). For an object of an elementary type, these are the bits that are normally read or updated by the machine code when loading, storing, or operating-on the value of the object. For an object of a composite type, these are the bits reserved for this object, and include bits occupied by subcomponents of the object. If the size of an object is greater than that of its subtype, the additional bits are padding bits. For an elementary object, these padding bits are normally read and updated along with the others. For a composite object, padding bits might not be read or updated in any given composite operation, depending on the implementation."

The representation = memory pattern. When RM says that S and T have same representation that is merely same pattern. It tells nothing about ordering of bits. Any combination of 8-bits is same representation of Unsigned_8 and Boolean array (1..8).

So I think Niklas is right.

I don't even know if 2**n Unsigned_8 should produce a singleton array. E.g. it is possible, but unlikely, that 2 could become (True, False, True, True, True, False, True, False).

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Tue, 31 Aug 2010 15:40:40 +0200
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

If I understand correctly, this suggests bits representation should be accessed (read/write) all the same way, whatever the type it implements.

If that is, may be this would be better to word it more explicitly. But this would be OK for elementary types only as this could always be broken for composite types with some representation clause combinations.

Providing this is OK, this could not be enforced for the example array type which is the subject of this topic. Or else, this would require some conditions about paddings.

I still feel this is implementation defined and I would not rely on it (or else I missed something).

A tricky and imaginary example by the way: imagine a clever compiler with clever optimization, which would detect the application mostly access the nth item of the array and far less often access any other items, now let us say the target CPU has a special instruction to access bit value at #0 (common practice on CISC processor), then it could choose to map this nth item on bit #0.

Do you feel the language would disallow such an optimization ? if it does not, this example shows this is implementation defined.

Side note: a compiler could do something similar for an unpacked array as well ; it could move the nth item at offset 0, to save processing of an offset while accessing an element which was determined as far more frequently accessed than the others.

Seems possible?

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Tue, 31 Aug 2010 16:07:56 +0200
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

> If I understand correctly, this suggests bits representation should be accessed (read/write) all the same way, whatever the type it implements.

The representation is accessed when bytes, words etc are read and written.

[…]

> Do you feel the language would disallow such an optimization ? if it does not, this example shows this is implementation defined.

AFAIK there is no requirement for Unchecked_Conversion be meaningful. The compiler is allowed do whatever it wishes with the representation if the type semantics is not violated.

After all, some objects can be optimized away. What is the representation of a non-resident object?

From: Niklas Holsti <niklas.holsti@tidorum.invalid>
Date: Tue, 31 Aug 2010 17:30:28 +0300
Subject: Re: bit numbers in packed arrays of Boolean
Newsgroups: comp.lang.ada

[…]

> If I understand correctly, this suggests bits representation should be accessed (read/write) all the same way, whatever the type it implements.

I'm not sure what you mean, but for me the *type* used to access a memory location is the factor that determines what the bits, and each bit, represents. The correspondence between bits in Unsigned_8 and a packed array of 8 Booleans is entirely implementation-defined, but can be inspected by Unchecked_Conversion between the types.

I'm not sure if the standard even requires, for a packed array of 8 Booleans of Size 8 bits, that each one of those bits should represent exactly one of the Boolean components. But even if one component is represented in one bit, the encoding (whether 0 is False and 1 is True, or vice versa) is implementation-defined (and could, I believe, be different depending on the index of the component... although that would be weird).

> I still feel this is implementation defined and I would not rely on it (or else I missed something).

I think Dmitry and I agree, with you, that it is implementation-defined.

[…]

> Do you feel the language would disallow such an optimization ?

No. I think that the basic RM does not say anything about the order of components in an array, whether Packed or not. (See below for Annexes.)

> Side note: a compiler could do something similar for an unpacked array as well ; it could move the nth item at offset 0, to save processing of an offset while accessing an element which was determined as far more frequently accessed than the others.

> Seems possible ?

In principle, yes.

Pragma Convention C or Fortran should impose some ordering rules, from the C/Fortran rules, and perhaps something is also implied by the Implementation Advice on the Ada/C interface in RM B.3(62.1/2 through 75). For example, the rule RM B.3(70) says that an Ada "array (..) of T" should be passed to C as "*t", where t is the C type corresponding to the Ada type T. For this to be a useful rule, the components in the Ada array must be laid out as in the C case, that is, with the address increasing monotonically with the index. Which is, of course, the natural lay-out that one would expect, in any case.

On representation clauses

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sun, 15 Aug 2010 13:33:32 +0200
Subject: Using representation clauses in networking software
Newsgroups: comp.lang.ada

Have there been any efforts to fix representation clauses so that they are suitable for expressing elements commonly found in networking protocols?

My understanding is that representation clauses are totally unsuitable for this purpose because they do not allow the programmer to express endianness. I dimly recall a technical report which argued that expressing endianness portably was impossible. However, Erlang's bit syntax seems to be a counterexample, so I wonder if that issue has been picked up in recent years.

From: Anonymous
Date: Mon, 16 Aug 2010 09:12:55 +0000 UTC
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

[…]

Representation clauses can be used for endianness but it is a little tricky the first time. After that it is easy to do. And it does not matter if you are using little or big endian Ada, as representation clauses can handle it as is. So, no modifications are needed.

And one of the first uses of the enumeration representation clause was to aid in converting programs files written in IBM EBCDIC character set into ASCII set back in the mid 1980s using Ada 83.

> So why were representation clauses added to Ada in the first place?

First, to be able to assign internal values to enumeration value such as the character set (see Standard package) or

"**type** Boolean **is** (False, True) ;"

In most C compiler today False has an internal value 0, but there were C compilers like Microsoft's that were used back in the 70s to the mid 90s that used -1, and their compiled code is still in use today, while others C compilers assigned True to 0. The representation clause allowed an Ada programmer to correct this problem without having to deal with the close source or binary code. Just interface and use the representation clause to match the Ada code to the external world and the program is good to go.

Second, is to set size and location of elements in a pack record. A simple example is the status registers, like a UART. It does not matter what the IO port uses, the status word values and bit pattern are standardized and using the representation clause with a pack record can allow a programmer to use simple Boolean logic instead of dealing with integer masking algorithms. In other words, let the compiler do the hard work.

The Address representation clause has 100s of uses. Like setting up DOS and no-OS type of interrupts. Also can be use to calculate the physical memory size for an Ada OS.

All of which allows Ada to translate the outside environment into the Ada world. Making the Ada program more portable and universal which extents the life of the program!

> Would it make sense to deprecate them?

The answer is NO!!! And they should not be expanded! They have a special place to aid the programmer in special areas. Such as insuring the code is portable when dealing with others areas that are not standard or in a pre-standard design!

An example is writing a keyboard driver. Is it easier to replace the keyboard layout every time a different keyboard is used and then re-booting or is it better to add an additional translation package using representation clause? Then allow the program to ask which layout to use from the hardware or in some cases the user.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sun, 15 Aug 2010 18:03:21 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

> I haven't tried it but

http://www.adaic.com/standards/05rm/html/RM-13-5-3.html

This only affects the interpretation of bit positions. It does not change the memory layout of components.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 16 Aug 2010 22:32:41 -0500
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

[…]

When we discussed this the last time, it was concluded that supporting out of order byte reads (especially those for unusual numbers of bits) is just too expensive for implementations for the relatively small amount of use that they would get. Moreover, such things could not be volatile or atomic on most hardware (because they would require multiple reads/writes of the various pieces - the pieces wouldn't necessarily be contiguous in memory for the non-native format), so they couldn't be used for hardware interfaces. That would eliminate half of the potential uses.

Remember that read/writes of memory are probably the most fundamental thing that a compiler does; supporting split reads would require changes to virtually every part of a compiler. (For Janus/Ada, we would have to add intermediate code instructions to support such reads, with costs in everything that handles intermediate code and target code generation.)

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Sun, 15 Aug 2010 15:44:34 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

[…]

General case first and yours then.

For representation issue, the common scheme is to use conversion when receiving something from the outside of the application and use conversion when sending something outside. This is done this way to be more efficient as this preserve the best representation for all internal uses.

This conversion may be implicitly defined via a type conversion between a type and a derived type if one has a representation clause applying.

Ex. a type T1 is a type with its universal definition, a second type T2 derived from T1 is to be stored in file or retrieved from a device using the size of X bits. Both are the same in some sense, except T2 has a representation clause. When you do "T1 (Object_Of_Type_T1)" there is an implicit conversion. The other way conversion is the same.

Your case now (without representation attribute, as this one does not exist so far) I do not know neither any representation clause for Byte Ordering.

However, you probably have a set of basic type like 16 bits words, 64 bits words or anything else. The best is to have a similar thing as the above, except this would be implemented explicitly. You would define a function From_Local_To_Network and and From_Network_To_Local (with two different types for Local_Type and Network_Type to avoid to erroneously mixing it) Just to try to meet your request (something else later) : I do not know a way to have a method which would automatically know the bit order of its platform, except perhaps receiving a standard data, like 16#1234# (0x1234), and then check if it looks like 16#3412# or like 16#1234#. Or this may be a link to a tiny-tiny library which would only contains a simple data word, which could be checked the same way. Ex. a library which would contains the same 6#1234# which would be linked as an external C stuff and could be tested for the same way.

The other way (which does not meet you requirement) : use the ability of Ada to have multiple package body for the same package specification and define two package body for the same network <->platform conversion package specification. Then, use a simple configuration option to build the application using one or the other (you will not face thousand of cases, as there are not some much commonly used platform in real life).

You may write it or reuse one (pretty sure something like this already exist in whatever license).

Or else you may suggest it for a future Ada revision (via this newsgroup or else ada-auth.org).

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 15 Aug 2010 16:32:47 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

[…]

(I agree with what you wrote. I am programming a lot of communication stuff, but never used representation clauses to handle endianness.)

> I do not know neither any representation clause for Byte Ordering.

Byte ordering is what S'Bit_Ordering is, when bytes are addressable.

IMO, Bit_Ordering was an unfortunate choice. The attribute name suggests ordering of bits in some machine storage unit, which it is not.

> Just to try to meet your request (something else later) : I do not know a way to have a method which would automatically know the bit order of its platform,

Bit order is useless if bits are not directly addressable.

> except perhaps receiving a standard data, like 16#1234# (0x1234), and then check if it looks like 16#3412# or like 16#1234#.

This would determine byte ordering. Bit ordering is when you serialize a sequence of bits {0, 0, 0, 0, 0, 0, 0, 1} and get 2#1000_0000# (little endian) 2#0000_0001# (big endian).

Most types of hardware have opaque bytes compatible with the machine, e.g. UARTs. If the hardware is not, then representation clause would not be my choice anyway. I would recode bytes or whatever units immediately as obtained from the device:

```
type Strange_Octet is mod 2**8;
Decode_Strange_Octet : array
  (Strange_Octet) of Unsigned_8 :=
  (  2#0000_0001# => 2#0000_1000#,
     2#0000_0010# => 2#0000_0100#,
     2#0000_0011# => 2#0000_1100#,
     ...
  );
```

[…]

One could introduce enumeration representation clauses for modular types, e.g.

```
type Strange_Octet is mod 2**8;
for Strange_Octet use
   (2#0000_0001# =>
   2#0000_1000#, ...);
```

Better would be to allow user-defined integer literals, so that the programmer could define its own type.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Mon, 16 Aug 2010 06:57:46 -0400
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

> Byte ordering is what S'Bit_Ordering is, when bytes are addressable.

Not exactly.

> IMO, Bit_Ordering was an unfortunate choice. The attribute name suggests ordering of bits in some machine storage unit, which it is not.

Bit_ordering specifies how to interpret the bit numbers in Ada record representation clauses.

The restriction of 'Bit_Order only being valid for the target endianness was removed in Ada 2005.

It is useful for representing record layouts that may be used in systems with different byte orders.

See http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ais/ai-00133.txt?rev=1.17 for a good discussion. The concept of 'machine scalars' is important.

Only part of this discussion made it into the ARM or AARM, so they are hard to understand.

Still, S'Bit_ordering does _not_ solve the inter-machine byte endianness problem. It could be part of a solution.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 15 Aug 2010 18:10:31 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

>> Byte ordering is what S'Bit_Ordering is, when bytes are addressable.

[…]

The attribute is semantically defined for Word_Size > Storage_Unit.

Therefore it is actually "unit ordering", or "byte ordering" when Storage_Unit is byte.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sun, 15 Aug 2010 16:44:33 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

> (I agree with what you wrote. I am programming a lot of communication stuff, but never used representation clauses to handle endianness.)

This is not surprising because the current representation clauses cannot handle it.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 15 Aug 2010 17:04:15 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

[…]

What for, if there are better ways to handle it? Representation layout clause was invented for handling something in place. It is no more actual, because reading out/writing in with an appropriate recoding is cleaner and possibly cheaper on modern hardware.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 15 Aug 2010 18:10:08 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

[…]

> There are some pretty common CPUs that handle unaligned loads quite well, so it's unclear that the load-a-byte-at-a-time-and-shift approach is a win.

> Same for serialization.

Usually there are next 2-4 protocol levels, so anything you might gain at this level will be lost anyway if you tried to keep things encoded. Worse than that, some of the layers may explicitly state endiannes at run time.

(I know at least two examples of such protocols) Dynamic representation clauses, shudder?

> So why were representation clauses added to Ada in the first place?

I think they were due to dominance of dual-ported memory and non-standardized hardware that time.

> Would it make sense to deprecate them?

Maybe, especially because they do not allow writing portable programs anyway. Under portability I understand that the hardware is fixed and the target machine varies.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 16 Aug 2010 08:40:54 +0200
Subject: Re: Using representation clauses in networking software
Newsgroups: comp.lang.ada

>> Stick to network-byte-order on the wire,

> Just when ~99% of hardware in use is little-endian?

This does not change anything.

The fact is that the *relevant* endianness is the one on the wire. Believe it or not, but it is normal practice that two little-endian machines exchange big-endian encoded numbers if the protocol mandates it.

On the instantiation of generic packages

From: Trogdor
Date: Mon, 30 Aug 2010 08:12:19 -0500
Subject: Global scope for instantiated generic package?
Newsgroups: comp.lang.ada

I wish to manipulate very large integers with values in the trillions. This would require a 64 bit integer type, so to maintain portability I specify a new type using range and let the compiler do the right thing.

```
type BigInt is range
   1 ..10_000_000_000_000;
```

I then wish to get and put these values, so I instantiate Integer_IO.

```
package BigInt_IO is new
   integer_IO(BigInt);
```

My program consists of three parts: the main body, a package spec and a package body (the package containing all the subprograms and functions).

1) Is this the way to do it, or is there a preferred way?

2) Where, specifically, do I place the above two lines (and the associated "with" line) so that I can BigInt_io.get from the main body and the package subprograms as well?

So far, every place I have tried has offended the compiler. And my text books have been of little help (I have more on order).

Thanks for the help!

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Mon, 30 Aug 2010 11:49:49 -0700
Subject: Re: Global scope for instantiated generic package?
Newsgroups: comp.lang.ada

> 1) Is this the way to do it, or is there a preffered way?

This sounds reasonable, though I'd need to have more detail to decide if it's the way I'd do it.

> 2) Where, specifically, do I place the above two lines (and the associated "with" line) so that I can BigInt_io.get from the main body and the package subprograms as well?

These would go in the (visible part of) the package specification.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 30 Aug 2010 15:47:05 +0200
Subject: Re: Global scope for instantiated generic package?
Newsgroups: comp.lang.ada

> […]

> 1) Is this the way to do it, or is there a preferred way?

I usually do it this way:

```
package Big_Integers is
  type Big_Integer is range
       1 ..10_000_000_000_000;
end Big_Integers;
```

Further instantiations based on it go into children packages:

```
with Ada.Text_IO;  use Ada.Text_IO;
package Big_Integers.IO is new
      Integer_IO (Big_Integer);
```

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 30 Aug 2010 06:30:17 -0700 PDT
Subject: Re: Global scope for instantiated generic package?
Newsgroups: comp.lang.ada

[…]

Without at least a skeleton of your sources I'm not sure I understand your question, so for clarity I'll assume you currently have:

```
with Ada.Text_IO;
package P is
  type BigInt is range
    1..10_000_000_000_000;
  package BigInt_IO is new
    Ada.Text_IO.Integer_IO (BigInt);
  procedure Proc (X : out BigInt);
end P;
package body P is
  procedure Proc (X : out BigInt) is
  begin
    BigInt_IO.Get (X); -- OK
  end Proc;
end P;
with P;
procedure Main is
  X : BigInt;
begin
  P.Proc (X); -- OK
  BigInt_IO.Get (X); -- error, BigInt_IO
                        not directly visible
  P.BigInt_IO.Get (X); -- OK
end Main;
```

You can resolve the error in two ways:

1) add a "use P;" clause in procedure Main

2) Declare BigInt_IO at library level, like so:

```
-- bigint_io.ads
with Ada.Text_IO;
with P;
package BigInt_IO is new
   Ada.Text_IO.Integer_IO (P.BigInt);
```

and then add "with BigInt_IO" clauses in both the body of P and the body of Main.

[…]

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Mon, 30 Aug 2010 15:28:42 +0200
Subject: Re: Global scope for instantiated generic package?
Newsgroups: comp.lang.ada

[…]

> 2) Where, specifically, do I place the above two lines (and the associated "with" line) so that I can BigInt_io.get from the main body and the package subprograms as well?

One way to place the declarations is in a common package.

```
with Ada.Text_IO; use Ada.Text_IO;
package Big_Stuff is
  type BigInt is range
     1..10_000_000_000_000;
  -- ... BigInt subprograms
  package BigInt_IO is new
     integer_IO(BigInt);
end Big_Stuff;
```

A main unit can then with package Big_Stuff (and "use" it); the procedure bodies in Big_Stuff's body will see BigInt_IO, too.

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: Mon, 30 Aug 2010 15:25:00 +0200
Subject: Re: Global scope for instantiated generic package?
Newsgroups: comp.lang.ada

[…]

> type BigInt is range 1..10_000_000_000_000;

This has to be done in a declarative region (i.e. inside a package or subprogram). This could be in a completely independent package:

```
-- trogdor.ads (for GNAT)
package Trogdor is
  type Integer is range
      1..10_000_000_000_000;
end Trogdor;
```

> I then wish to get and put these values, so I instantiate Integer_IO.

> package BigInt_IO is new integer_IO(BigInt);

This can be a compilation unit of its own:

```
-- trogdor_io.ads (for GNAT)
with Trogdor;
package Trogdor_IO is new
    Ada.Text_IO.Integer_IO
    (Trogdor.Integer);
```

Or you can put it inside the same package as the number type:

```
-- trogdor.ads (for GNAT)
package Trogdor is
  type Integer is range
      1..10_000_000_000_000;
  package IO is new
      Ada.Text_IO.Integer_IO (Integer);
end Trogdor;
```

> My program consists of three parts: the main body, a package spec and a package body (the package containing all the subprograms and functions).

> 1) Is this the way to do it, or is there a preferred way?

If the subprograms are specific to the program, you may want to declare them in the declarative part of the program rather than in a separate package.

SPARK and Rosetta Code

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Wed, 11 Aug 2010 10:44:58 +0200
Subject: SPARK code samples
Newsgroups: comp.lang.ada

Since there is evident growing interest in SPARK Ada and the availability of a public SPARK compiler, I welcome those who are interested in learning and testing SPARK to contribute their solutions to the Rosetta Code:

http://rosettacode.org/wiki/Main_Page

The Rosetta Code has a half of thousand programming tasks defined. The tasks are solved for almost programming language ever existed. Ada is well represented in Rosetta, but SPARK is not. (Clearly, not all tasks could be implemented in SPARK)

Rosetta is pretty liberal, everyone can register and contribute. The SPARK's page is:

http://rosettacode.org/wiki/SPARK

Thanks.

SPARK and "shadow specifications"

From: Ada novice
<ycalleecharan@gmx.com>
Date: Tue, 10 Aug 2010 01:48:30 -0700 PDT
Subject: Spark and the Ada numerics annex
Newsgroups: comp.lang.ada

Hi, I became interested in taking a look at SPARK. I've browsed through the freely available first chapter of Barnes's SPARK book and I saw that SPARK doesn't seem to cover the Ada specialized annexes (see Figure 1.1. Relationship between SPARK and Ada on pp. 11).

My interests are purely in Scientific numerical programming with Ada and the Ada numerics annex is important to me. […]

If I understand correctly, then simple mathematical functions like SQRT are elementary functions that are part of the language Ada itself and SPARK will recognize these elementary functions. But the extensive vector and matrix manipulations as added in Ada 05 won't be recognized by SPARK. Am I right?

The good things about SPARK is that it helps to make checks and hence get the programming right. Is it recommended to use SPARK even for very simple programs?

[…]

From: Yannick Duchêne
<yannick_duchene@yahoo.fr>
Date: Tue, 10 Aug 2010 14:09:24 +0200
Subject: Re: Spark and the Ada numerics annex
Newsgroups: comp.lang.ada

SPARK will applies, it does not target a specific application area. While you may feel it is less handy in some than in some others. If you ever feel SPARK looks too much stupid for some package or method, then may just hide it from SPARK and try to comment the best way ever possible too give a proof that although not checked by SPARK, argues that this "should be good".

> The good things about SPARK is that it helps to make checks and hence get the programming right. Is it recommended to use SPARK even for very simple programs?

Not only this is possible, this is also recommended in some way. The smaller an application is, the less pain you will have to prove its correctness.

SPARK does not take part in a reification process (like things would go with the B-method), so there is a lot of things which will seem obvious to you; SPARK may not know right away about you application.

For this reason, this may be better to keep it simple. Don't forget you can Hide if ever required.

```
procedure Your_Method is
  --# hide Your_Method
  ... Your proses comes here ...
  ... Your informal comments proving
     you designed it right comes here  …
end Your_Method;
```

Just don't abuse it.

From: Rod Chapman
<roderick.chapman@googlemail.com>
Date: Tue, 10 Aug 2010 03:18:53 -0700 PDT
Subject: Re: Spark and the Ada numerics annex
Newsgroups: comp.lang.ada

You can create a "shadow" specification of the various Numerics packages to get at their facilities. See section 13.1 of the book on Shadows.

From: Yannick Duchêne
<yannick_duchene@yahoo.fr>
Date: Tue, 10 Aug 2010 13:40:18 +0200
Subject: Re: Spark and the Ada numerics annex
Newsgroups: comp.lang.ada

[…]

You can already get an idea right know (although the Barnes is still a must have recommended reading), if you have a look at this AdaGem :

http://www.adacore.com/2010/02/25/gem-80/

Basically, this is re-interfacing things so that SPARK does not get angry with what it sees.

By the way, this is a natural strategy, … sure you would have thought about it yourself. I've discovered this is named "shadow specification" just at the moment, reading Rod. I did not knew before this had this name.

Conference Calendar

Dirk Craeynest

K.U.Leuven. Email: Dirk.Craeynest@cs.kuleuven.be

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

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

2010

October 03-08 ACM/IEEE 13th **International Conference on Model Driven Engineering Languages and Systems** (MoDELS'2010), Oslo, Norway. Topics include: Development of domain-specific modeling languages; Design of general-purpose modeling languages and related standards; Tools and meta-tools for modeling languages and model-based development; Evolution of modeling languages and models; Experience stories in general (successful and unsuccessful); Issues related to limitations, gaps and mismatches in current modeling standards; Experience with model-based engineering tools and traceability to/from models; etc.

☺ Oct 04 MoDELS2010 – 3rd **International Workshop on Model Based Architecting and Construction of Embedded Systems** (ACES-MB'2010). Topics include: model-oriented counterparts, together with the related analysis and development methods, of languages with particularly well-behaved semantics, such as synchronous languages and models (Lustre/SCADE, Signal/Polychrony, Esterel), super-synchronous models (TTA, Giotto), scheduling-friendly models (HRT-UML, Ada Ravenscar), etc.

October 06-07 5th **International Workshop on Systems Software Verification** (SSV'2010), Vancouver, Canada. Theme: "Real Software, Real Problems, Real Solutions". Topics include: static analysis, model-driven development, embedded systems development, programming languages, verifying compilers, software certification, software tools, experience reports, etc.

October 10-13 9th **International Conference on Generative Programming and Component Engineering** (GPCE'2010), Eindhoven, The Netherlands. Topics include: Generative techniques for Product-line architectures, Distributed, real-time and embedded systems, Model-driven development and architecture, Safety critical systems; Component-based software engineering (Reuse, distributed platforms and middleware, distributed systems, evolution, patterns, development methods, formal methods, etc.); Integration of generative and component-based approaches; Industrial applications; etc.

Oct 12-13 3rd **International Conference on Software Language Engineering** (SLE'2010). Topics include: Formalisms used in designing and specifying languages and tools that analyze such language descriptions; Language implementation techniques; Program and model transformation tools; Language evolution; Approaches to elicitation, specification, or verification of requirements for software languages; Design challenges in SLE; Applications of languages including innovative domain-specific languages or "little" languages; etc.

October 11-14 8th **International Conference on Integrated Formal Methods** (iFM'2010), Nancy, France. Topics include: the combination of (formal and semi-formal) methods for system development, covering all aspects from language design through verification and analysis techniques to tools and their integration into software engineering practice.

October 13-16 17th **Working Conference on Reverse Engineering** (WCRE'2010), Boston's North Shore, Beverly, Massachusetts, USA. Topics include: all areas of software maintenance, evolution, reengineering, and migration, such as Program comprehension, Mining software repositories, Empirical studies in reverse engineering, Redocumenting legacy systems, Reverse engineering tool support, Reengineering to distributed architectures, Software architecture recovery, Program analysis and slicing, Reengineering patterns, Program transformation and refactoring, etc.

☺ Oct 16-20 **Systems, Programming, Languages, and Applications: Software for Humanity** (SPLASH'2010), Reno, Nevada, USA. Formerly known as OOPSLA. Topics include: defining the future of software development.

☺ Oct 17 **International Workshop on Programming Support Innovations for Emerging Distributed Applications** (PSIEtA'2010). Topics include: programming support innovations that can address the incongruence between the advanced programming requirements of emerging distributed applications and the current state of the art of their programming support.

☺ Oct 17-20 25th **Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2010). Topics include: all aspects of programming languages and software engineering, broadly construed; any aspect of software development, including requirements, modeling, prototyping, design, implementation, generation, analysis, verification, testing, evaluation, project cancellation, maintenance, reuse, regeneration, replacement, and retirement of software systems; tools (such as new programming languages, dynamic or static program analyses, compilers, and garbage collectors) or techniques (such as new programming methodologies, type systems, design processes, code organization approaches, and management techniques) designed to reduce the time, effort, and/or cost of software systems.

Oct 17-21 **Onward! 2010**. Topics include: the multidisciplinarity of software development; anything to do with programming and software, such as processes, methods, languages, economics, applications, etc.

☺ Oct 18 **Workshop on Concurrency for the Application Programmer** (CAP'2010).

☺ Oct 18 2nd **Workshop on Evaluation and Usability of Programming Languages and Tools** (PLATEAU'2010). Topics include: methods, metrics and techniques for evaluating the usability of languages and language tools, such as empirical studies of programming languages; methodologies and philosophies behind language and tool evaluation; software design metrics and their relations to the underlying language; user studies of language features and software engineering tools; critical comparisons of programming paradigms, such as object-oriented vs. functional; tools to support evaluating programming languages; etc.

☺ Oct 18 9th **Workshop on Parallel/High-Performance Object-Oriented Scientific Computing** (POOSC'2010). Topics include: how object-oriented programming can benefit scientific computing; new or novel frameworks, approaches, techniques, or idioms that use object-orientation. Specific areas of interest include: alternatives or extensions, including multi-paradigmatic approaches, to mainstream object-oriented languages (e.g. C++, Java, Python); performance issues and their realized or proposed resolution; issues specific to handling or abstracting parallelism, including the handling or abstraction of heterogeneous/multicore/accelerated microarchitectures; higher level languages (e.g. domain specific languages) or their embedding into OO languages to support parallelism or specific tasks in scientific computing; proposed or realized solutions to problems hindering acceptance of object-oriented scientific computing; grand visions (of relevance); etc.

♦ Oct 24-28 ACM SIGAda **Annual International Conference on Ada and Related Technologies** (SIGAda'2010), Fairfax, Virginia, USA (a suburb of Washington, DC). Sponsored by ACM SIGAda, in cooperation with SIGBED, SIGCAS, SIGCSE, SIGPLAN, Ada-Europe, and the Ada Resource Association. Includes: workshop on OO and Ada in High Integrity Systems. Deadline for early registration: October 16, 2010.

October 25-29 Grid2010 – **Workshop on Component-Based High Performance Computing** (CBHPC'2010), Brussels, Belgium. Topics include: Programming environments and paradigms, Analysis and comparison of existing programming approaches, Tools and Environments for Coupling of Parallel Application codes, etc.

October 25-29 14th IEEE **International Enterprise Computing Conference** (EDOC'2010), Vitória, ES, Brazil. Topics include: Organization and principles of software factories; State of the art in distributed enterprise applications; Industry specific solutions, e.g. for aerospace, automotive, finance, etc.

☺ Nov 01-03 29th IEEE **International Symposium on Reliable Distributed Systems** (SRDS'2010), Delhi, India. Topics include: security, safety-critical systems and critical infrastructures, fault-tolerance in embedded systems, analytical or experimental evaluations of dependable distributed systems, formal methods and foundations for dependable distributed computing, etc.

November 08-12 13th **Brazilian Symposium on Formal Methods** (SBMF'2010), Natal, Rio Grande do Norte, Brazil. Topics include: Formal aspects of popular languages and methodologies; Logics and semantics of programming and specification languages; Type systems in computer science; Formal methods integration; Code generation; Formal design methods; Abstraction, modularization and refinement techniques; Techniques for correctness by construction; Formal methods and models for real-time, hybrid and critical systems; Models of concurrency, security and mobility; Theorem proving; Static analysis; Software certification; Teaching of, for and with formal methods; Experience reports on the use of formal methods; Industrial case studies; Tools supporting the formal development of computational systems; Development methodologies with formal foundations; etc.

☺ Dec 07-11 16th IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2010), Shanghai, China. Topics include: Parallel and Distributed Algorithms and Applications, Multi-core and Multithreaded Architectures, Resource Management and Scheduling, Security, Dependable and Trustworthy Systems, Real-Time Systems, Embedded systems, etc.

☺ Dec 08-11 11th **International Conference on Parallel and Distributed Computing, Applications, and Techniques** (PDCAT'2010), Wuhan, China. Topics include: all areas of parallel and distributed computing.

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

2011

☺ Jan 26-28 38th ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2011), Austin, Texas. Topics include: all aspects of programming languages and systems, with emphasis on how principles underpin practice.

Jan 24-25 ACM SIGPLAN **Workshop on Partial Evaluation and Program Manipulation** (PEPM'2011). Deadline for submissions: October 10, 2010 (abstracts), October 15, 2010 (papers).

☺ Jan 29 5th ACM SIGPLAN **Workshop on Programming Languages meets Program Verification** (PLPV'2011). Topics include: research at the intersection of programming languages and program verification; attempts to reduce the burden of program verification by taking advantage of particular semantic or structural properties of the programming language; all aspects, both theoretical and practical, of the integration of programming language and program verification technology. Deadline for submissions: October 11, 2010.

☺ Feb 09-10 3rd **International Symposium on Engineering Secure Software and Systems** (ESSoS'2011), Madrid, Spain. Topics include: security architecture and design for software and systems; verification techniques for security properties; systematic support for security best practices; programming paradigms for security; processes for the development of secure software and systems; etc.

☺ Feb 21-25 **Software Engineering 2011** (SE'2011), Karlsruhe, Germany. Theme (in German): Ingenieurmäßige Software-Entwicklung für kritische Anwendungen. Deadline for submissions: October 3, 2010 (abstracts), October 8, 2010 (all submissions).

☺ March 09-12 42nd ACM **Technical Symposium on Computer Science Education** (SIGCSE'2011), Dallas, Texas, USA.

March 21-25 26th ACM **Symposium on Applied Computing** (SAC'2011), TaiChung, Taiwan.

☺ Mar 21-25 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2011). Topics include: Language design and implementation; Type systems, static analysis, formal methods; Integration with other paradigms; Aspects, components, and modularity; Distributed, concurrent or parallel systems; Interoperability, versioning and software adaptation; etc.

☺ Mar 21-25 **Track on Programming Languages** (PL'2011). Topics include: Compiling Techniques, Formal Semantics and Syntax, Garbage Collection, Language Design and Implementation, Languages for Modeling, Model-Driven Development and Model Transformation, New Programming Language Ideas and Concepts, Practical Experiences with Programming Languages, Program Analysis and Verification, Programming Languages from All Paradigms, etc.

☺ Mar 21-25 **Track on Real-Time Systems** (RTS'2011). Topics include: all aspects of real-time systems design, analysis, implementation, evaluation, and case-studies; including scheduling and schedulability analysis; worst-case execution time analysis; modeling and formal methods; validation techniques; reliability; compiler support; component-based approaches; middleware and distribution technologies; programming languages and operating systems; embedded systems; etc.

Mar 21-25 4th IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2011), Berlin, Germany. Topics include: Domain specific testing including, but not limited to, security testing, embedded software testing, OO software testing, ...; Verification & validation; Quality assurance; Empirical studies; Inspections; Tools; Novel approaches to software reliability assessment; etc.

☺ Mar 28-31 14th IEEE **International Symposium on Object/component/service-oriented Real-time distributed Computing** (ISORC'2011), Newport Beach, California, USA. Topics include: Programming and system engineering (ORC paradigms, languages, RT Corba, UML, model-driven development of high integrity applications, specification, design, verification, validation, testing, maintenance, system of systems, etc.); System software (real-time kernels, middleware support for ORC, extensibility, synchronization, scheduling, fault tolerance, security, etc.); Applications (embedded systems (automotive, avionics, consumer electronics, etc), real-time object-oriented simulations, etc.); System evaluation (timeliness, worst-case execution time, dependability, fault detection and recovery time, etc.); ... Deadline for submissions: November 8, 2010 (papers).

☺ April 10-13 6th **European Conference on Computer Systems** (EuroSys'2011), Salzburg, Austria. Topics include: all areas of operating systems and distributed systems, including systems aspects of Dependable computing and storage, Distributed computing, Parallel and concurrent computing, Programming-language support, Real-time and embedded computing, Security, etc. Deadline for submissions: October 3, 2010 (abstracts), October 10, 2010 (full papers), October 22, 2010 (workshops, tutorials).

April 12-15 2nd **International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering** (PARENG'2011), Ajaccio, Corsica, France.

April 20-24 17th **International Symposium on Formal Methods** (FM'2011), Limerick, Ireland. Theme: "Formal Methods Come of Age". Topics include: advances and maturity in formal methods research, education, and deployment via tool support and industrial best practice, and their role in a variety of industries, domains, and in certification and assurance; in particular experience with practical applications of formal methods in industrial and research settings, experimental validation of tools and methods as well as construction and evolution of formal methods tools. Deadline for submissions: January 10, 2011 (papers), January 24, 2011 (tutorials, workshops).

☺ April 25-29 5th **Latin-American Symposium on Dependable Computing** (LADC'2010), São José dos Campos, São Paulo, Brazil. Topics include: Dependability of software (frameworks and software architectures for dependability, model driven dependability engineering, testing, verification, software certification, ...); Dependability of maintenance; Dependability and human issues; Security; Safety (safety-critical applications and systems, ...); etc. Deadline for submissions: October 14, 2010 (tutorials), February 4, 2011 (industry track, fast abstracts, student forum).

☺ May 16-20 25th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2011), Anchorage, Alaska, USA. Topics include: all areas of parallel and distributed processing, such as: Parallel and distributed algorithms; Applications of parallel and distributed computing; Parallel and distributed software, including parallel and multicore programming languages and compilers, runtime systems, middleware, libraries, parallel programming paradigms, programming environments and tools, etc.

☺ May 21-28 — 33rd **International Conference on Software Engineering** (ICSE'2011), Waikiki, Honolulu, Hawaii, USA. Theme: "Software by Design". Topics include: Engineering of distributed/parallel software systems; Engineering of embedded and real-time software; Engineering secure software; Patterns and frameworks; Programming languages; Reverse engineering and maintenance; Software architecture and design; Software components and reuse; Software dependability, safety and reliability; Software economics and metrics; Software tools and development environments; Theory and formal methods; etc. Deadline for submissions: October 15, 2010 (software engineering in practice), November 30, 2010 (SCORE project registration), December 10, 2010 (doctoral symposium, new ideas and emerging results, demonstrations), January 30, 2011 (student volunteers).

> May 21-28 — 2nd **Student COntest on softwaRe Engineering** (SCORE'2011). Deadline for submissions: November 30, 2010 (registration for participation), January 15, 2011 (summary reports), February 28, 2011 (final deliverable), ICSE 2011 (finals).

June 20-23 — **2011 International Conference for Computational Science and its Applications** (ICCSA'2011), Santander, Spain. Deadline for submissions: December 31, 2010 (abstracts, full papers). Deadline for early registration: April 4, 2011.

♦ June 20-24 — 16th **International Conference on Reliable Software Technologies - Ada-Europe'2011**, Edinburgh, UK. Co-located with the Ada Conference UK 2011, organized under the common name of "The Ada Connection". Sponsored by Ada-Europe, in cooperation with ACM SIGAda. Deadline for submissions: November 21, 2010 (papers, tutorials, workshops), January 8, 2011 (industrial presentations).

June 27-29 — 16th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2011), Darmstadt, Germany.

☺ June 28-30 — 49th **International Conference Objects, Models, Components, Patterns** (TOOLS Europe'2011), Zurich, Switzerland. Topics include: Applications to safety- and security-related software; Distributed and concurrent object systems; Domain specific languages and language design; Experience reports, including efforts at standardisation; Language implementation techniques, compilers, run-time systems; Multicore programming, models and patterns; Object technology, including programming techniques, languages, tools; Practical applications of program verification and analysis; Real-time object-oriented programming and design; Tools and frameworks for supporting model-driven development; Trusted and reliable components; etc. Deadline for submissions: January 28, 2011 (papers).

July 14-20 — 23rd **International Conference on Computer Aided Verification** (CAV'2011), Snowbird, Utah, USA. Topics include: Algorithms and tools for verifying models and implementations, Program analysis and software verification, Verification methods for parallel and concurrent hardware/software systems, Applications and case studies, Verification in industrial practice, etc. Deadline for submissions: January 14, 2011 (abstracts), January 21, 2011 (papers).

☺ July 25-29 — 25th **European Conference on Object-Oriented Programming** (ECOOP'2011), Lancaster, UK. Topics include: all areas of object technology and related software development technologies, such as Analysis and design methods and patterns; Distributed, concurrent, real-time systems; Language design and implementation; Modularity, components, services; Software development environments and tools; Type systems, formal methods; Compatibility, software evolution; etc. Deadline for submissions: December 15, 2010.

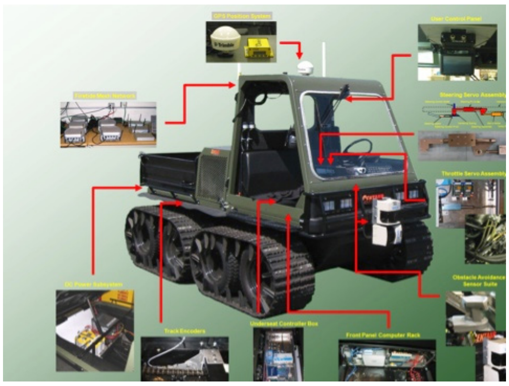
☺ Aug 30 – Sep 09 — **International Conference on Parallel Computing 2011** (ParCo'2011), Gent, Belgium. Topics include: all aspects of parallel computing, including applications, hardware and software technologies as well as languages and development environments, in particular Applications of multicores, GPU-based applications, Parallel programming languages, compilers and environments, Best practices of parallel computing, etc. Deadline for submissions: March 20, 2011 (extended abstracts of papers), March 31, 2011 (proposals for mini-symposia).

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

ACM Annual International Conference on Ada and Related Technologies:

Engineering Safe, Secure, and Reliable Software

Hyatt Fair Lakes Hotel, Fairfax, Virginia (Washington DC Area), USA
October 24-28, 2010



MITRE Robot Tractor System powered by Ada

Conference Highlights

Keynote: Transforming Software and System Development and Analysis

William B. Martin, National Security Agency

This address will put forward an evidence-based approach to assurance in the hopes of approaching these questions:

- How can software and systems be made dependable and secure in a more cost-effective manner?
- How can one obtain assurance that dependability and security have been achieved?
- Given the issues inherent in the development and assessment of today's systems, what hope do we have of building and assessing the systems of the future, systems that are likely to have billions of lines of code?

In short, the approach will require evidence allowing improvements to be objectively assessed. As well, with this dependence on evidence, the construction and analysis of the same will need to be pursued.

Keynote: Systems Software Integrity Assurance to FAA's Next Generation (NextGen) Constituents

Chris Lane, Lockheed Martin Corporation

Data Comm is a program that will enhance existing communications between the air traffic controller and the pilot by essentially sending digital messages to supplement the existing voice communications. With more reliance on Data Comm as the FAA's Next Generation systems become fielded, ensuring the communications is reliable, accurate, and most importantly safe becomes increasingly critical. RTCA DO-278 provides the guidelines for communications, navigation, surveillance, and air traffic management systems software integrity assurance. It doesn't guarantee that the software developed in accordance with these guidelines is safe but if followed it ensures that the processes are in place to properly plan, develop and verify the software. Lockheed Martin is in the process of integrating Data Comm with the En Route Automation and Modernization (ERAM) program and is developing the program in compliance with DO-278. This brings challenges as well as opportunities with the increasing reliance on commercial off the shelf (COTS) software. These challenges and some insight into developing systems to the standards of DO-278 will be discussed.

See complete conference and registration details at
http://www.sigada.org/conf/sigada2010/

More Conference Highlights

Tutorial: Developing Unmanned Systems in Ada over RTEMS

Cindy Cicalese et al, The MITRE Corporation

This hands-on tutorial provides an introduction to the development of software in Ada for unmanned systems. The authors will demonstrate how they are using Ada over RTEMS in developing the real time control software for a large unmanned ground vehicle. RTEMS is an open source, real-time operating system that provides a high performance environment for embedded applications on a range of processors and embedded hardware.

Workshop on Software Security

Stephen Michell, Maurya Software, Inc., Ottawa, Ontario, Canada

This hands-on workshop will involve a "find the vulnerability" exercise in sample code, using examples from C, C++, scripting languages and Ada.

Panel: Mitigating Risks to the Enterprise via Software Assurance

Joe Jarzombek (Department of Homeland Security)

Developers and cyber security specialists can make the mistake of focusing on the technical aspects of software security when management is more concerned about risk to the business or mission. Panel presentations and discussions will provide a focus to enhance efforts to:

- Understand the industry-wide implications of standards-based security architectures for measurement and management of enterprise IT security risks.
- Recognize the importance of security standards for enabling interoperability in security assessment, remediation, threat identification, incident management, system certification, and secure development.
- Realize how standards can make ground-truth reporting of compliance efforts economical, real-time, and accurate.
- Understand and gain access to free software assurance resources for those in acquisition, development, sustainment and support of operations.

Panel: Wouldn't It Be Nice to Have Software Labels?

Paul Black and Elizabeth Fong (National Institute of Standards and Technology)

Companies in most industries realize that efficient operation of their computer systems depends on trusted software in order to satisfy their business functions. Would it be helpful if a buyer of a software product could look at a simple label and get information on which of 2 or more products is safer to use? This would help the buyer to decide which product was best for their business. The goal of this project is to present relevant information to empower consumers to make the right decisions. Some of the possible criteria are that the software labeling and reporting are voluntary (not legislated), absolutely simple to produce, and content should be normalized across different scopes. Some of the possible criteria for contents are that the facts should be verifiable, objective, repeatable, and unambiguous.

Update on the Forthcoming Ada 2012 Standard

Ed Schonberg, Adacore, Inc.

The Ada Rapporteur Group (ARG) is nearing completion of its work on the forthcoming standard for Ada 2012. This update will present the state of the ARG work and an informal discussion of the language enhancements in the new standard.

See complete conference and registration details at

http://www.sigada.org/conf/sigada2010/

Call for Papers

The Ada Connection

16th International Conference on
Reliable Software Technologies -
Ada-Europe 2011

Ada Conference UK
2011

20 - 24 June 2011, Edinburgh, UK

http://www.ada-europe.org/conference2011

Honorary Chair

John Barnes
John Barnes Informatics, UK
jgpb@jbinfo.demon.co.uk

Conference Co-Chairs

Rod Chapman
Altran Praxis Ltd, UK
rod.chapman@altran-praxis.com

Steve Riddle
Newcastle University, UK
steve.riddle@ncl.ac.uk

Program Co-Chairs

Alexander Romanovsky
Newcastle University, UK
alexander.romanovsky@ncl.ac.uk

Tullio Vardanega
University of Padua, Italy
tullio.vardanega@math.unipd.it

Tutorial Chair

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

Exhibition Chair

Joan Atkinson
CSR, UK
joan.atkinson@ncl.ac.uk

Industrial Chair

Jamie Ayre
AdaCore, France
ayre@adacore.com

Publicity Chair

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

Finance Chair

Neil Speirs
Newcastle University, UK
neil.speirs@ncl.ac.uk

Local Chairs

Joan Atkinson
CSR, UK
joan.atkinson@ncl.ac.uk

Claire Smith
CSR, UK
claire.smith@ncl.ac.uk

acm In-Cooperation

In cooperation with
ACM SIGAda

General Information

The Ada Connection combines the 16th International Conference on Reliable Software Technologies – *Ada-Europe 2011* – with *Ada Conference UK 2011*. It will take place in Edinburgh, Scotland's capital city and the UK's most popular conference destination.

In traditional Ada-Europe style, the conference will span a full week, including a three-day technical program and vendor exhibition from Tuesday to Thursday, along with parallel tutorials and workshops on Monday and Friday. The Ada Connection will also encompass technical and vendor tracks under the banner of Ada Conference UK, which exists to promote awareness of Ada and to highlight the increased relevance of Ada in safety- and security-critical programming.

The Ada Connection will thus provide a unique opportunity for interaction and collaboration between academics and industrial practitioners.

Schedule

21 November 2010	Submission deadline for regular papers, tutorial and workshop proposals
8 January 2011	Submission of industrial presentation proposals
8 February 2011	Notification of acceptance to authors
8 March 2011	Camera-ready version of regular papers required
16 May 2011	Industrial presentations, tutorial and workshop material required
20-24 June 2011	Conference

Topics

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

All contributions, whether regular papers, industrial presentations, tutorials or workshops, should address the topics of interest to the conference, which for this edition include but are not limited to:

- **Methods and Techniques for Software Development and Maintenance**: Requirements Engineering, Object-Oriented Technologies, Model-driven Engineering, Formal Methods and Supporting Toolsets, Re-engineering and Reverse Engineering, Reuse, Software Management.
- **Software Architectures**: Architectural Styles, Service-Oriented Architectures, Cloud Service Model, Design Patterns, Frameworks, Architecture-Centered Development, Component and Class Libraries, Component-based Design and Development.
- **Enabling Technologies**: Software Development Environments, Compilers, Debuggers, Run-time Systems, Middleware Components, Concurrent and Distributed Programming, Ada Language and Technologies.
- **Software Quality**: Quality Management and Assurance, Risk Analysis, Program Analysis, Verification, Validation, Testing of Software Systems.
- **Theory and Practice of High-Integrity Systems**: Real-Time, Distribution, Fault Tolerance, Security, Reliability, Availability, Trust and Safety, Language Vulnerabilities.
- **Embedded Systems**: Multicore Architectures, HW/SW Co-Design, Reliability and Performance Analysis.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Energy, Fun and Business Games, Telecommunication, etc.
- **Experience Reports**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **The Future of Ada**: New language features, implementation and use issues; positioning in the market and in education; where should Ada stand in the software engineering curriculum; lessons learned on Ada Education and Training Activities with bearing on any of the conference topics.

Program Committee

Alejandro Alonso, Universidad Politécnica de Madrid, Spain
Ted Baker, Florida State University, USA
John Barnes, John Barnes Informatics, UK
Johann Blieberger, Technische Universität Wien, Austria
Jørgen Bundgaard, Rovsing A/S, Denmark
Bernd Burgstaller, Yonsei University, Korea
Alan Burns, University of York, UK
Jon Burton, Altran Praxis Limited, UK
Rod Chapman, Altran Praxis Limited, UK
Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium
Alfons Crespo, Universidad Politécnica de Valencia, Spain
Juan A. de la Puente, Universidad Politécnica de Madrid, Spain
Franco Gasperoni, AdaCore, France
Michael González Harbour, Universidad de Cantabria, Spain
José Javier Gutiérrez, Universidad de Cantabria, Spain
Andrew Hately, Eurocontrol Experimental Centre, France
Peter Hermann, Universität Stuttgart, Germany
Jérôme Hugues, ISAE Toulouse, France
Albert Llemosí, Universitat de les Illes Balears, Spain
Franco Mazzanti, ISTI-CNR Pisa, Italy
John McCormick, University of Northern Iowa, USA
Julio Medina, Universidad de Cantabria, Spain
Stephen Michell, Maurya Software, Canada
Javier Miranda, Universidad Las Palmas de Gran Canaria, Spain
Daniel Moldt, University of Hamburg, Germany
Jürgen Mottok, Regensburg University of Applied Sciences, Germany
Laurent Pautet, Telecom Paris, France
Luís Miguel Pinho, Polytechnic Institute of Porto, Portugal
Erhard Plödereder, Universität Stuttgart, Germany
Jorge Real, Universidad Politécnica de Valencia, Spain
Alexander Romanovsky, Newcastle University, UK
Bo I. Sandén, Colorado Technical University, USA
Sergio Sáez, Universidad Politécnica de Valencia, Spain
Ed Schonberg, AdaCore, USA
Theodor Tempelmeier, Univ. of Applied Sciences Rosenheim, Germany
Jean-Loup Terraillon, European Space Agency, The Netherlands
Elena Troubitsyna, Åbo Akademi, Finland
Santiago Urueña, GMV, Spain
Tullio Vardanega, Università di Padova, Italy
Andy Wellings, University of York, UK
Jürgen Winkler, Friedrich-Schiller Universität, Germany

Industrial Committee

Guillem Bernat, Rapita Systems, UK
Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium
Hubert Keller, Forschungszentrum Karlsruhe GmbH, Germany
Ismael Lafoz, Airbus Military, Spain
Ahlan Marriott, White-Elephant GmbH, Switzerland
Paul Parkinson, Wind River, UK
Jean-Pierre Rosen, Adalog, France
Alok Srivastava, TASC Inc, USA
Rei Stråhle, Sweden
Rod White, MBDA, UK

Call for Regular Papers

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall be in English, complete and not exceeding 14 LNCS-style pages in length. Authors should submit their work via the Web submission system accessible from the Conference Home page. The format for submission is solely PDF. Should you have problems to comply with format and submission requirements, please contact the *Program Chairs*.

Proceedings

The conference proceedings will be published in the Lecture Notes in Computer Science (LNCS) series by Springer, and will be available at the start of the conference. The authors of accepted regular papers shall prepare camera-ready submissions in full conformance with the LNCS style, not exceeding 14 pages and strictly by 8 March 2011. For format and style guidelines authors should refer to *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register for the conference by that date will prevent the paper from appearing in the proceedings.

The conference is ranked class A in the CORE ranking and is listed among the top quarter of CiteSeerX Venue Impact Factor.

Awards

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

Call for Industrial Presentations

The conference also seeks industrial presentations which deliver value and insight, but may not fit the selection process for regular papers. Authors of industrial presentations are invited to submit a short overview (at least one page) of the proposed presentation by 8 January 2011. Please follow the submission instructions on the conference website. The *Industrial Committee* will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it by 16 May 2011, aiming at a 20-minute talk. Accepted authors will also be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference. For any further information please contact the *Industrial Chair* directly.

Call for Tutorials

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

Call for Workshops

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

Call for Exhibitors

The commercial exhibition will span two days (Tuesday and Wednesday) of the main conference. Vendors and providers of software products and services should contact the *Exhibition Chair* for information and for allowing suitable planning of the exhibition space and time.

Grants for Reduced Student Fees

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

Student Programming Contest “The Ada Way”

Introduction

"The Ada Way" is an annual student programming contest organized by Ada-Europe, the international organization that promotes the knowledge and use of Ada in European academia, research and industry. A Steering Committee formed by representatives of promoting institutions oversees the organization of the contest. The Steering Committee is currently comprised of: Dirk Craeynest and Ahlan Marriott (Ada-Europe), Ricky Sward (ACM SIGAda), Jamie Ayre and Matteo Bordin (AdaCore), Jean-Pierre Fauche (Atego), Ian Broster (Rapita), Rod White (MBDA).

This initiative aims to attract students and educators to Ada in a form that is both fun and instructive. For this reason the contest is a yearly programming competition among student teams, whereby each team must have a university affiliation and be endorsed by an educator. The ideal, but not exclusive, context for participation is as part of an organized teaching/course activity in which the theme and requirements of the contest are endorsed and supported by the educator. See the "Participation Requirements" section for details.

The contest opens in September with the announcement of the theme, and allows submissions until the end of April the following year. See below for the 2010-11 edition theme and the Submissions section for the submission requirements.

Students and educators who may consider participating and want more information on "The Ada Way" in general and its 2010-11 edition in particular are invited to make contact with the Steering Committee at *board@ada-europe.org*.

Project Theme for Academic Year 2010-11: Software simulator of a football (soccer) match

The following specification intentionally leaves some room for interpretation and extension: participants are encouraged to use their intelligent creativity to firm up the derivative specification they want to work against.

The software system shall support at least the following features:

- Users must be able to play a single game; support for playing a series of matches, with fixtures and associated rules, is optional and can be omitted
- The chosen variant of the game shall be configurable in all relevant parameters, allowing for any of 5-a-side, 7-a-side, and the canonical 11-a-side formats
- The members of the squads will feature individually configurable characteristics for, at least, technical and tactical skills, speed, physical parameters including fatigue; some of those parameters shall be dynamic and evolve with the match according to some programmed logic
- Each squad shall have a (software) manager able to configure the initial players line up, the initial tactic and to issue commands for tactic changes and substitutions, all subject to the rules of the game as in the corresponding standard
- Each squad shall play according to the tactic commanded by the manager; deviations shall be permitted in so far as they result from programmable characteristics of the players
- Each match shall have one independent (software) referee and two to three subordinate (software) assistants who control the game and ensure that the applicable rules are followed; the behavior and the performance of the referee and assistants need not exhibit the physical limitations of actual humans.

The software system shall include at least:

- A software core, whether centralized or distributed, implementing all of the logic of the simulation
- One read-only graphical panel (window) for the display of the football field, the players, the ball, the referee and assistants; as for the (simulated) human figures on the pitch it shall be sufficient to represent them as moving numbered dots on the display without resorting to sophisticated graphical rendering, as in a view of a subbuteo table seen from the top
- Two distinct read-write graphical panels (windows) for the user to influence the otherwise independent action of the team managers; the panel shall display the current parameters for each player; the refresh rate of such display shall be user-configurable
- One read-only graphical panel (window) for the display of a user-configurable selection of statistics; the refresh rate of such display shall be user-configurable.

The software core shall be programmed in Ada. The software design shall permit the principal algoritms to be modified and replaced at will: in other words, the software system shall be as modular, configurable and scalable as possible. These qualities will contribute to the evaluation.

The graphical panels can be programmed in any language that the participating teams will consider fit for purpose. The graphical beauty of such panels will however be only a minor factor in the evaluation. What shall matter instead is that the interaction and the flow of data and control between the software core and the graphical panels is governed by good architectural principles and shows sufficient accuracy and performance.

To be considered for evaluation, the system shall run out of the box. The target platform may be freely chosen between Linux, Windows and MacOS. Portability across them will however be a competitive advantage.

Participation requirements

Participating teams shall be composed by a minimum of 2 and a maximum of 7 members. Each team shall have a codename and a logo. Team work may be performed as part of an organized teaching/course activity or as a volunteer project. Either way, each team must be recognised and endorsed by an academic educator.

Team members must be full-time students: they must provide evidence of their status when submitting their project. The contest is open to undergraduate and Master students. Teams may but need not include a mix of undergraduate and graduate students. Team members may belong to distinct institutions.

Submission

The software system shall be delivered in source (as a single compressed archive), accompanied by:

1. A software specification document (in PDF), which describes the principal design decisions and argues their quality, and presents the points of extension and modification in the system; the specification shall clearly single out all places at which the team made arbitrary interpretation of the specification or added or extended requirements
2. A user manual describing the compilation and installation procedures, the configuration options and the allowable use of the system (in PDF)
3. The team codename, logo and composition: name, email contact, evidence of enrollment as full-time students (in a single PDF)
4. The written endorsement to the submission by an academic or otherwise senior instructor in whose class the project was launched (in PDF).

The submission shall be made as a single compressed archive of all items listed above at the URL that will appear on this page in due time.

All sources shall be released for the good of the general public, to become reference material for educational and promotional purposes. To this end the use of GPL (GNU General Public License) is recommended, though we are not prescriptive of a specific scheme, so long as the general intent of free dissemination is preserved.

Submissions shall be accepted during the whole month of April 2011, at the Ada Way website, *http://www.ada-europe.org/AdaWay*.

Evaluation and Prize

The evaluation criteria will include:

- Coverage of requirements
- Syntatic, semantic, programmatic and design correctness
- Clarity and readability of the code
- Quality of design
- Ingenuity and cuteness of the solution
- Time and space efficiency of the solution.

The evaluation will be performed by a team of distinguished Ada experts comprised of: John Barnes (UK), Tucker Taft (US), Joyce Tokar (US), Pascal Leroy (F), Ed Schonberg (US).

The winning submission shall be announced on 31 May 2011 by a post on the site and by an email communication to all participating teams.

The prize will consist of: a framed award; one free registration and up to 3 reduced student fees for representatives of the winning team to attend to the Ada-Europe 2011 Conference; accommodation and airfare for the team representatives (with ceiling at EUR 3,000); an exhibition slot in the conference program; visibility in electronic and printed media including:

- Ada User Journal: *http://www.ada-europe.org/journal.html*
- Ada Letters: *http://www.sigada.org/ada_letters/*

For up-to-date information on Ada-Europe's student programming contest, please go to the official web site of “The Ada Way”, *http://www.ada-europe.org/AdaWay,*

Sponsors

This year's competition is sponsored by Ada-Europe, AdaCore, and Atego.

Ada and the Software Vulnerabilities Project

Alan Burns, FREng (ed.)

Department of Computer Science, University of York, York YO1 5DD UK; Tel: +44 (0)1904 432779;
email: burns@cs.york.ac.uk

Joyce L. Tokar, PhD (ed.)

Pyrrhus Software, PO Box 1352, Phoenix, AZ, 85001-1352, USA.; Tel: +1 602373 0713;
email: tokar@pyrrhusoft.com

Stephen Baird, John Barnes, Rod Chapman, Gary Dismukes, Michael González-Harbour, Stephen Michell, Brad Moore, Luís Miguel Pinho, Erhard Ploedereder, Jorge Real, J.P. Rosen, Ed Schonberg, S. Tucker Taft, T. Vardanega

Abstract

Given the large focus on software vulnerabilities in the current market place, ISO/IEC JTC 1/SC 22/WG 23 has developed a Technical Report (TR) on Vulnerabilities [1]. This TR contains vulnerabilities that may be applicable to a programming language or application. This article provides a synopsis of these vulnerabilities with respect to the Ada programming language [2].

Keywords: software vulnerabilities, software vulnerability, Ada, SPARK.

1 Introduction

Software vulnerabilities are defined as a property of a system security, requirements, design, implementation, or operation that could be accidentally triggered or intentionally exploited and result in a security failure [3]. Work on software vulnerabilities and how they enable software applications to be infiltrated and corrupted continues to be of interest world. Working Group 23 (WG 23) of the Programming Languages Subcommittee (SC 22) of the International Organization of Standards (ISO) has recently completed a Technical Report that identifies and enumerates a collection of software vulnerabilities in existing programming languages [1]. Annexes to this document are being developed to identify if the vulnerabilities defined in the TR exist in various programming languages.

A workshop was conducted in parallel with the 14th International Conference on Reliable Software Technologies – Ada-Europe 2009 to initiate the development of content of an Annex to the Technical Report that documents its applicability to the Ada and SPARK programming languages. The results of this workshop were published in [4]. Another workshop was conducted in parallel with the 2009 SIGAda conference. Work continued on this document over the course of 2009 and was completed in a short workshop at the 15th International Conference on Reliable Software Technologies – Ada-Europe 2010.

This content of this article is the final draft copy of the Ada Annex to the WG 23 TR that will be submitted to WG 23 for inclusion in the TR. An annex was also developed by Altran-Praxis for SPARK *.

Note, within the WG 23 TR each vulnerability is assigned a unique identifier such as RIP for the Inheritance vulnerability. Since the WG 23 TR was under development during the work on this Annex and there is an expectation that more vulnerabilities will be added to the TR, the sections in the Ada Annex include their corresponding unique identifier in the section heading.

References

[1] ISO/IEC JTC 1/SC 22 N 4522, ISO/IEC TR 24772, Information Technology — Programming Languages — Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use, 7 November 2009.

[2] Taft, S. Tucker, Duff, R. A., Brukardt, R. L., Ploedereder, E., Leroy, P, Ada Reference Manual, LNCS 4348, Springer, Heidelberg, 2006.

[3] NIST Special Publication 268, "Source Code Security Analysis Tool Functional Specification Version 1.0," May 2007.

[4] Proceedings of the Software Vulnerabilities Workshop of Ada-Europe 2009, in Ada User Journal, Volume 30, Number 3, September 2009, pp. 174-192.

* Editor's note: The SPARK Annex is scheduled for publication in the December issue of the Ada User Journal.

Annex Ada – Final Draft

Ada.Specific information for vulnerabilities

Every vulnerability description of Clause 6 of the main document is addressed in the annex in the same order even if there is simply a note that it is not relevant to Ada.

This Annex specifies the characteristics of the Ada programming language that are related to the vulnerabilities defined in this Technical Report. When applicable, the techniques to mitigate the vulnerability in Ada applications are described in the associated section on the vulnerability.

Ada.1 Identification of standards and associated documentation

ISO/IEC 8652:1995, Information Technology – Programming Languages—Ada.

ISO/IEC 8652:1995/COR.1:2001, Technical Corrigendum to Information Technology – Programming Languages—Ada.

ISO/IEC 8652:1995/AMD.1:2007, Amendment to Information Technology – Programming Languages—Ada.

ISO/IEC TR 15942:2000, Guidance for the Use of Ada in High Integrity Systems.

ISO/IEC TR 24718:2005, Guide for the use of the Ada Ravenscar Profile in high integrity systems.

Lecture Notes on Computer Science 5020, “Ada 2005 Rationale: The Language, the Standard Libraries,” John Barnes, Springer, 2008.

Ada 95 Quality and Style Guide, SPC-91061-CMC, version 02.01.01. Herndon, Virginia: Software Productivity Consortium, 1992.

Ada Language Reference Manual, The consolidated Ada Reference Manual, consisting of the international standard (ISO/IEC 8652:1995): *Information Technology -- Programming Languages -- Ada*, as updated by changes from *Technical Corrigendum 1* (ISO/IEC 8652:1995:TC1:2000), and Amendment 1 (ISO/IEC 8526:AMD1:2007).

IEEE 754-2008, IEEE Standard for Binary Floating Point Arithmetic, IEEE, 2008.

IEEE 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic, IEEE, 1987.

Ada.2 General terminology and concepts

Access object: An object of an access type.

Access-to-Subprogram: A pointer to a subprogram (function or procedure).

Access type: The type for objects that designate (point to) other objects.

Access value: The value of an access type; a value that is either null or designates (points at) another object.

Attributes: Predefined characteristics of types and objects; attributes may be queried using syntax of the form <entity>'<attribute_name>.

Bounded Error: An error that need not be detected either prior to or during run time, but if not detected, then the range of possible effects shall be bounded.

Case statement: A case statement provides multiple paths of execution dependent upon the value of the case expression. Only one of alternative sequences of statements will be selected.

Case expression: The case expression of a case statement is a discrete type.

Case choices: The choices of a case statement must be of the same type as the type of the expression in the case statement. All possible values of the case expression must be covered by the case choices.

Compilation unit: The smallest Ada syntactic construct that may be submitted to the compiler. For typical file-based implementations, the content of a single Ada source file is usually a single compilation unit.

Configuration pragma: A directive to the compiler that is used to select partition-wide or system-wide options. The **pragma** applies to all compilation units appearing in the compilation, unless there are none, in which case it applies to all future compilation units compiled into the same environment.

Controlled type: A type descended from the language-defined type Controlled or Limited_Controlled. A controlled type is a specialized type in Ada where an implementer can tightly control the initialization, assignment, and finalization of objects of the type. This supports techniques such as reference counting, hidden levels of indirection, reliable resource allocation, etc.

Discrete type: An integer type or an enumeration type.

Discriminant: A parameter for a composite type. It can control, for example, the bounds of a component of the type if the component is an array. A discriminant for a task type can be used to pass data to a task of the type upon creation.

Erroneous execution: The unpredictable result arising from an error that is not bounded by the language, but that, like a bounded error, need not be detected by the implementation either prior to or during run time.

Exception: Represents a kind of exceptional situation. There are set of predefined exceptions in Ada in **package** Standard: Constraint_Error, Program_Error, Storage_Error, and Tasking_Error; one of them is raised when a language-defined check fails.

Expanded name: A variable V inside subprogram S in package P can be named V, or P.S.V. The name V is called the *direct name* while the name P.S.V is called the *expanded name*.

Idempotent behaviour: The property of an operations that has the same effect whether applied just once or multiple times. An example would be an operation that rounded a number up to the nearest even integer greater than or equal to its starting value.

Implementation defined: Aspects of semantics of the language specify a set of possible effects; the implementation may choose to implement any effect in the set. Implementations are required to document their behaviour in implementation-defined situations.

Modular type: A modular type is an integer type with values in the **range** 0 .. modulus - 1. The modulus of a modular type can be up to 2**N for N-bit word architectures. A modular type has wrap-around semantics for arithmetic operations, bit-wise "and" and "or" operations, and arithmetic and logical shift operations.

Partition: A partition is a program or part of a program that can be invoked from outside the Ada implementation.

Pointer: Synonym for “access object.”

Pragma: A directive to the compiler.

Pragma Atomic: Specifies that all reads and updates of an object are indivisible.

Pragma Atomic_Components: Specifies that all reads and updates of an element of an array are indivisible.

Pragma Convention: Specifies that an Ada entity should use the conventions of another language.

Pragma Detect_Blocking: A configuration pragma that specifies that all potentially blocking operations within a protected operation shall be detected, resulting in the Program_Error exception being raised.

Pragma Discard_Names: Specifies that storage used at run-time for the names of certain entities may be reduced.

Pragma Export: Specifies an Ada entity to be accessed by a foreign language, thus allowing an Ada subprogram to be called from a foreign language, or an Ada object to be accessed from a foreign language.

Pragma Import: Specifies an entity defined in a foreign language that may be accessed from an Ada program, thus allowing a foreign-language subprogram to be called from Ada, or a foreign-language variable to be accessed from Ada.

Pragma Normalize_Scalars: A configuration pragma that specifies that an otherwise uninitialized scalar object is set to a predictable value, but out of range if possible.

Pragma Pack: Specifies that storage minimization should be the main criterion when selecting the representation of a composite type.

Pragma Restrictions: Specifies that certain language features are not to be used in a given application. For example, the **pragma** Restrictions (No_Obsolescent_Features) prohibits the use of any deprecated features. This **pragma** is a configuration pragma which means that all program units compiled into the library must obey the restriction.

Pragma Suppress: Specifies that a run-time check need not be performed because the programmer asserts it will always succeed.

Pragma Unchecked_Union: Specifies an interface correspondence between a given discriminated type and some C union. The **pragma** specifies that the associated type shall be given a representation that leaves no space for its discriminant(s).

Pragma Volatile: Specifies that all reads and updates on a volatile object are performed directly to memory.

Pragma Volatile_Components: Specifies that all reads and updates of an element of an array are performed directly to memory.

Scalar type: A discrete or a real type.

Subtype declaration: A construct that allows programmers to declare a named entity that defines a possibly restricted subset of values of an existing type or subtype, typically by imposing a constraint, such as specifying a smaller range of values.

Task: A task represents a separate thread of control that proceeds independently and concurrently between the points where it interacts with other tasks. An Ada program may be comprised of a collection of tasks.

Unsafe Programming: In recognition of the occasional need to step outside the type system or to perform “risky” operations, Ada provides clearly identified language features to do so. Examples include the generic Unchecked_Conversion for unsafe type conversions or Unchecked_Deallocation for the deallocation of heap objects regardless of the existence of surviving references to the object. If unsafe programming is employed in a unit, then the unit needs to specify the respective generic unit in its context clause, thus identifying potentially unsafe units. Similarly, there are ways to create a potentially unsafe global pointer to a local object, using the Unchecked_Access attribute. A restriction pragma may be used to disallow uses of Unchecked_Access.

Ada.3.BRS Obscure Language Features [BRS]

Ada.3.BRS.1 Terminology and features

Ada.3.BRS.2 Description of vulnerability

Ada is a rich language and provides facilities for a wide range of application areas. Because some areas are specialized, it is likely that a programmer not versed in a special area might misuse features for that area. For example, the use of tasking features for concurrent programming requires knowledge of this domain. Similarly, the use of exceptions and exception propagation and handling requires a deeper understanding of control flow issues than some programmers may possess.

Ada.3.BRS.3 Avoiding the vulnerability or mitigating its effects

The **pragma** Restrictions can be used to prevent the use of certain features of the language. Thus, if a program should not use feature X, then writing **pragma** Restrictions (No_X); ensures that any attempt to use feature X prevents the program from compiling. Similarly, features in a Specialized Needs Annex should not be used unless the application area concerned is well-understood by the programmer.

Ada.3.BRS.4 Implications for standardization

None

Ada.3.BRS.5 Bibliography

None

Ada.3.BQF Unspecified Behaviour [BQF]

Ada.3.BQF.1 Terminology and features

Generic formal subprogram: A parameter to a generic package used to specify a subprogram or operator.

Ada.3.BQF.2 Description of vulnerability

In Ada, there are two main categories of unspecified behaviour, one having to do with unspecified aspects of normal run-time behaviour, and one having to do with *bounded errors*, errors that need not be detected at run-time but for which there is a limited number of possible run-time effects (though always including the possibility of raising Program_Error).

For the normal behaviour category, there are several distinct aspects of run-time behaviour that might be unspecified, including:

- Order in which certain actions are performed at run-time;
- Number of times a given element operation is performed within an operation invoked on a composite or container object;
- Results of certain operations within a language-defined generic package if the actual associated with a particular formal subprogram does not meet stated expectations (such as “<” providing a strict weak ordering relationship);
- Whether distinct instantiations of a generic or distinct invocations of an operation produce distinct values for tags or access-to-subprogram values.

The index entry in the Ada Standard for *unspecified* provides the full list. Similarly, the index entry for *bounded error* provides the full list of references to places in the Ada Standard where a bounded error is described.

Failure can occur due to unspecified behaviour when the programmer did not fully account for the possible outcomes, and the program is executed in a context where the actual outcome was not one of those handled, resulting in the program producing an unintended result.

Ada.3.BQF.3 Avoiding the vulnerability or mitigating its effects

As in any language, the vulnerability can be reduced in Ada by avoiding situations that have unspecified behaviour, or by fully accounting for the possible outcomes.

Particular instances of this vulnerability can be avoided or mitigated in Ada in the following ways:

- For situations where order of evaluation or number of evaluations is unspecified, using only operations with no side-effects, or idempotent behaviour, will avoid the vulnerability;
- For situations involving generic formal subprograms, care should be taken that the actual subprogram satisfies all of the stated expectations;
- For situations involving unspecified values, care should be taken not to depend on equality between potentially distinct values;
- For situations involving bounded errors, care should be taken to avoid the situation completely, by ensuring in other ways that all requirements for correct operation are satisfied before invoking an operation that might result in a bounded error. See the Ada Annex section Ada.3.28 on Initialization of Variables [LAV] for a discussion of

uninitialized variables in Ada, a common cause of a bounded error.

Ada.3.BQF.4 Implications for standardization

When appropriate, language-defined checks should be added to reduce the possibility of multiple outcomes from a single construct, such as by disallowing side-effects in cases where the order of evaluation could affect the result.

Ada.3.BQF.5 Bibliography

None

Ada.3.EWF Undefined Behaviour [EWF]

Ada.3.EWF.1 Terminology and features

Abnormal Representation: The representation of an object is incomplete or does not represent any valid value of the object's subtype.

Ada.3.EWF.2 Description of vulnerability

In Ada, undefined behaviour is called *erroneous execution*, and can arise from certain errors that are not required to be detected by the implementation, and whose effects are not in general predictable.

There are various kinds of errors that can lead to erroneous execution, including:

- Changing a discriminant of a record (by assigning to the record as a whole) while there remain active references to subcomponents of the record that depend on the discriminant;
- Referring via an access value, task id, or tag, to an object, task, or type that no longer exists at the time of the reference;
- Referring to an object whose assignment was disrupted by an abort statement, prior to invoking a new assignment to the object;
- Sharing an object between multiple tasks without adequate synchronization;
- Suppressing a language-defined check that is in fact violated at run-time;
- Specifying the address or alignment of an object in an inappropriate way;
- Using Unchecked_Conversion, Address_To_Access_Conversions, or calling an imported subprogram to create a value, or reference to a value, that has an *abnormal* representation.

The full list is given in the index of the Ada Standard under *erroneous execution*.

Any occurrence of erroneous execution represents a failure situation, as the results are unpredictable, and may involve overwriting of memory, jumping to unintended locations within memory, etc.

Ada.3.EWF.3 Avoiding the vulnerability or mitigating its effects

The common errors that result in erroneous execution can be avoided in the following ways:

- All data shared between tasks should be within a protected object or marked Atomic, whenever practical;
- Any use of Unchecked_Deallocation should be carefully checked to be sure that there are no remaining references to the object;
- **pragma** Suppress should be used sparingly, and only after the code has undergone extensive verification.

The other errors that can lead to erroneous execution are less common, but clearly in any given Ada application, care must be taken when using features such as:

- **abort**;
- Unchecked_Conversion;
- Address_To_Access_Conversions;
- The results of imported subprograms;
- Discriminant-changing assignments to global variables.

The mitigations described in Section 6.EWF.5 are applicable here.

Ada.3.EWF.4 Implications for standardization

When appropriate, language-defined checks should be added to reduce the possibility of erroneous execution, such as by disallowing unsynchronized access to shared variables.

Ada.3.EWF.5 Bibliography

None

Ada.3.FAB Implementation-Defined Behaviour [FAB]

Ada.3.FAB.1 Terminology and features

None

Ada.3.FAB.2 Description of vulnerability

There are a number of situations in Ada where the language semantics are implementation defined, to allow the implementation to choose an efficient mechanism, or to match the capabilities of the target environment. Each of these situations is identified in Annex M of the Ada Standard, and implementations are required to provide documentation associated with each item in Annex M to provide the programmer with guidance on the implementation choices.

A failure can occur in an Ada application due to implementation-defined behaviour if the programmer presumed the implementation made one choice, when in fact it made a different choice that affected the results of the execution. In many cases, a compile-time message or a run-time exception will indicate the presence of such a problem. For example, the range of integers supported by a given compiler is implementation defined. However, if the programmer specifies a range for an integer type that exceeds that supported by the implementation, then a compile-time error will be indicated, and if at run time a computation exceeds the base range of an integer type, then a Constraint_Error is raised.

Failure due to implementation-defined behaviour is generally due to the programmer presuming a particular effect that is not matched by the choice made by the implementation. As indicated above, many such failures are indicated by compile-time error messages or run-time exceptions. However, there are cases where the implementation-defined behaviour might be silently misconstrued, such as if the implementation presumes Ada.Exceptions.Exception_Information returns a string with a particular format, when in fact the implementation does not use the expected format. If a program is attempting to extract information from Exception_Information for the purposes of logging propagated exceptions, then the log might end up with misleading or useless information if there is a mismatch between the programmer’s expectation and the actual implementation-defined format.

Ada.3.FAB.3 Avoiding the vulnerability or mitigating its effects

Many implementation-defined limits have associated constants declared in language-defined packages, generally **package** System. In particular, the maximum range of integers is given by System.Min_Int .. System.Max_Int, and other limits are indicated by constants such as System.Max_Binary_Modulus, System.Memory_Size, System.Max_Mantissa, etc. Other implementation-defined limits are implicit in normal ‘First and ‘Last attributes of language-defined (sub) types, such as System.Priority’First and System.Priority’Last. Furthermore, the implementation-defined representation aspects of types and subtypes can be queried by language-defined attributes. Thus, code can be parameterized to adjust to implementation-defined properties without modifying the code.

- Programmers should be aware of the contents of Annex M of the Ada Standard and avoid implementation-defined behaviour whenever possible.
- Programmers should make use of the constants and subtype attributes provided in package System and elsewhere to avoid exceeding implementation-defined limits.
- Programmers should minimize use of any predefined numeric types, as the ranges and precisions of these are all implementation defined. Instead, they should declare their own numeric types to match their particular application needs.
- When there are implementation-defined formats for strings, such as Exception_ Information, any necessary processing should be localized in packages with implementation-specific variants.

Ada.3.FAB.4 Implications for standardization

Language standards should specify relatively tight boundaries on implementation-defined behaviour whenever possible, and the standard should highlight what levels represent a portable minimum capability on which programmers may rely. For languages like Ada that allow user declaration of numeric types, the number of predefined numeric types should be minimized (for example, strongly discourage or disallow declarations of Byte_Integer, Very_Long_Integer, etc., in **package** Standard).

Ada.3.FAB.5 Bibliography

None

Ada.3.MEM Deprecated Language Features [MEM]

Ada.3.MEM.1 Terminology and features

Obsolescent Features: Ada has a number of features that have been declared to be obsolescent; this is equivalent to the term deprecated. These are documented in Annex J of the Ada Reference Manual.

Ada.3.MEM.2 Description of vulnerability

If obsolescent language features are used, then the mechanism of failure for the vulnerability is as described in Section 6.MEM.3.

Ada.3.MEM.3 Avoiding the vulnerability or mitigating its effects

- Use **pragma** Restrictions (No_Obsolescent_Features) to prevent the use of any obsolescent features.
- Refer to Annex J of the Ada reference manual to determine if a feature is obsolescent.

Ada.3.MEM.4 Implications for standardization

None.

Ada.3.MEM.5 Bibliography

None

Ada.3.NMP Pre-Processor Directives [NMP]

This vulnerability is not applicable to Ada as Ada does not have a pre-processor.

Ada.3.NAI Choice of Clear Names [NAI]

Ada.3.NAI.1 Terminology and features

Identifier: Identifier is the Ada term that corresponds to the term name.

Ada is not a case-sensitive language. Names may use an underscore character to improve clarity.

Ada.3.NAI.2 Description of vulnerability

There are two possible issues: the use of the identical name for different purposes (overloading) and the use of similar names for different purposes.
This vulnerability does not address overloading, which is covered in Section Ada.3.YOW.
The risk of confusion by the use of similar names might occur through:

- Mixed casing. Ada treats upper and lower case letters in names as identical. Thus no confusion can arise through an attempt to use Item and ITEM as distinct identifiers with different meanings.
- Underscores and periods. Ada permits single underscores in identifiers and they are significant. Thus BigDog and Big_Dog are different identifiers. But multiple underscores (which might be confused with a single underscore) are forbidden, thus Big__Dog is forbidden. Leading and trailing underscores are also forbidden. Periods are not permitted in identifiers at all.
- Singular/plural forms. Ada does permit the use of identifiers which differ solely in this manner such as Item and Items. However, the user might use the identifier Item for a single object of a type T and the identifier Items for an object denoting an array of items that is of a type array (…) of T. The use of Item where Items was intended or vice versa will be detected by the compiler because of the type violation and the program rejected so no vulnerability would arise.
- International character sets. Ada compilers strictly conform to the appropriate international standard for character sets.
- Identifier length. All characters in an identifier in Ada are significant. Thus Long_IdentifierA and Long_IdentifierB are always different. An identifier cannot be split over the end of a line. The only restriction on the length of an identifier is that enforced by the line length and this is guaranteed by the language standard to be no less than 200.

Ada permits the use of names such as X, XX, and XXX (which might all be declared as integers) and a programmer could easily, by mistake, write XX where X (or XXX) was intended. Ada does not attempt to catch such errors.
The use of the wrong name will typically result in a failure to compile so no vulnerability will arise. But, if the wrong name has the same type as the intended name, then an incorrect executable program will be generated.

Ada.3.NAI.3 Avoiding the vulnerability or mitigating its effects

This vulnerability can be avoided or mitigated in Ada in the following ways: avoid the use of similar names to denote different objects of the same type. See the Ada Quality and Style Guide.

Ada.3.NAI.4 Implications for standardization

None

Ada.3.NAI.5 Bibliography

None

Ada.3.AJN Choice of Filenames and other External Identifiers [AJN]

Ada.3.AJN.1 Terminology and features

Ada enables programs to interface to external identifiers in various ways. Filenames can be specified when files are opened by the use of the packages for input and output such as Text_IO. In addition the packages Ada.Directories, Ada.Command_Line, and Ada.Environment_Variables give access to various external features. However, in all cases, the form and meaning of the external identifiers is stated to be implementation-defined.

Ada.3.AJN.2 Description of vulnerability

As described in Section 6.AJN.

Ada.3.AJN.3 Avoiding the vulnerability or mitigating its effects

As described in Section 6.AJN.

Ada.3.AJN.4 Implications for standardization

None

Ada.3.AJN.5 Bibliography

None

Ada.3.XYR Unused Variable [XYR]

Ada.3.XYR.1 Terminology and features

Dead store: An assignment to a variable that is not used in subsequent instructions. A variable that is declared but neither read nor written to in the program is an unused variable.

Ada requires all variables to be explicitly declared.

Ada.3.XYR.2 Description of vulnerability

Variables might be unused for various reasons:

- Declared for future use. The programmer might have declared the variable knowing that it will be used when the program is complete or extended. Thus, in a farming application, a variable Pig might be declared for later use if the farm decides to expand out of dairy farming.
- The declaration is wrong. The programmer might have mistyped the identifier of the variable in its declaration, thus Peg instead of Pig.
- The intended use is wrong. The programmer might have mistyped the identifier of the variable in its use, thus Pug instead of Pig.

An unused variable declared for later use does not of itself introduce any vulnerability. The compiler will warn of its absence of use if such warnings are switched on.

If the declaration is wrong, then the program will not compile assuming that the uses are correct. Again there is no vulnerability.

If the use is wrong, then there is a vulnerability if a variable of the same type with the same name is also declared. Thus, if the program correctly declares Pig and Pug (of the same type) but inadvertently uses Pug instead of Pig, then the program will be incorrect but will compile.

Ada.3.XYR.3 Avoiding the vulnerability or mitigating its effects

- Do not declare variables of the same type with similar names. Use distinctive identifiers and the strong typing of Ada (for example through declaring specific types such as Pig_Counter **is range** 0 .. 1000; rather than just Pig: Integer;) to reduce the number of variables of the same type.

- Unused variables can be easily detected by the compiler, whereas dead stores can be detected by static analysis tools.

Ada.3.XYR.4 Implications for standardization

None

Ada.3.XYR.5 Bibliography

None

Ada.3.YOW Identifier Name Reuse [YOW]

Ada.3.YOW.1 Terminology and features

Hiding: A declaration can be *hidden*, either from direct visibility, or from all visibility, within certain parts of its scope. Where *hidden from all visibility*, it is not visible at all (neither using a direct_name nor a selector_name). Where *hidden from direct visibility*, only direct visibility is lost; visibility using a selector_name is still possible.

Homograph: Two declarations are *homographs* if they have the same name, and do not overload each other according to the rules of the language.

Ada.3.YOW.2 Description of vulnerability

Ada is a language that permits local scope, and names within nested scopes can hide identical names declared in an outer scope. As such it is susceptible to the vulnerability of 6.YOW. For subprograms and other overloaded entities the problem is reduced by the fact that hiding also takes the signatures of the entities into account. Entities with different signatures, therefore, do not hide each other.

The failure associated with common substrings of identifiers cannot happen in Ada because all characters in a name are significant (see section Ada.3.NAI).

Name collisions with keywords cannot happen in Ada because keywords are reserved. Library names Ada, System, Interfaces, and Standard can be hidden by the creation of subpackages. For all except package Standard, the expanded name Standard.Ada, Standard.System and Standard.Interfaces provide the necessary qualification to disambiguate the names.

Ada.3.YOW.3 Avoiding the vulnerability or mitigating its effects

Use *expanded names* whenever confusion may arise.

Ada.3.YOW.4 Implications for standardization

Ada could define a pragma Restrictions identifier No_Hiding that forbids the use of a declaration that results in a local homograph.

Ada.3.YOW.5 Bibliography

None

Ada.3.BJL Namespace Issues [BJL]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as Ada provides packages to control namespaces and enforces block structure semantics.

Ada.3.IHN Type System [IHN]

Ada.3.IHN.1 Terminology and features

Explicit Conversion: The Ada term explicit conversion is equivalent to the term cast in Section 6.IHN.3.

Implicit Conversion: The Ada term implicit conversion is equivalent to the term coercion.

Ada uses a strong type system based on name equivalence rules. It distinguishes types, which embody statically checkable equivalence rules, and subtypes, which associate dynamic properties with types, e.g., index ranges for array subtypes or value ranges for numeric subtypes. Subtypes are not types and their values are implicitly convertible to all other subtypes of the same type. All subtype and type conversions ensure by static or dynamic checks that the converted value is within the value range of the target type or subtype. If a static check fails, then the program is rejected by the compiler. If a dynamic check fails, then an exception Constraint_Error is raised.

To effect a transition of a value from one type to another, three kinds of conversions can be applied in Ada:

a) Implicit conversions: there are few situations in Ada that allow for implicit conversions. An example is the assignment of a value of a type to a polymorphic variable of an encompassing class. In all cases where implicit conversions are permitted, neither static nor dynamic type safety or application type semantics (see below) are endangered by the conversion.

b) Explicit conversions: various explicit conversions between related types are allowed in Ada. All such conversions ensure by static or dynamic rules that the converted value is a valid value of the target type. Violations of subtype properties cause an exception to be raised by the conversion.

c) Unchecked conversions: Conversions that are obtained by instantiating the generic subprogram Unchecked_Conversion are unsafe and enable all vulnerabilities mentioned in Section 6.IHN as the result of a breach in a strong type system. Unchecked_Conversion is occasionally needed to interface with type-less data structures, e.g., hardware registers.

A guiding principle in Ada is that, with the exception of using instances of Unchecked_Conversion, no undefined semantics can arise from conversions and the converted value is a valid value of the target type.

Ada.3.IHN.2 Description of vulnerability

Implicit conversions cause no application vulnerability, as long as resulting exceptions are properly handled.

Explicit conversions can violate the application type semantics. e.g., conversion from feet to meter, or, in general, between types that denote value of different units, without the appropriate conversion factors can cause application vulnerabilities. However, no undefined semantics can result and no values can arise that are outside the range of legal values of the target type.

Failure to apply correct conversion factors when explicitly converting among types for different units will result in application failures due to incorrect values.

Failure to handle the exceptions raised by failed checks of dynamic subtype properties cause systems, threads or components to halt unexpectedly.

Unchecked conversions circumvent the type system and therefore can cause unspecified behaviour (see Section Ada.3.AMV).

Ada.3.IHN.3 Avoiding the vulnerability or mitigating its effects

- The predefined 'Valid attribute for a given subtype may be applied to any value to ascertain if the value is a legal value of the subtype. This is especially useful when interfacing with type-less systems or after Unchecked_Conversion.

- A conceivable measure to prevent incorrect unit conversions is to restrict explicit conversions to the bodies of user-provided conversion functions that are then used as the only means to effect the transition between unit systems. These bodies are to be critically reviewed for proper conversion factors.
- Exceptions raised by type and subtype conversions shall be handled.

Ada.3.IHN.4 Implications for standardization

None

Ada.3.IHN.5 Bibliography

None

Ada.3.STR Bit Representation [STR]

Ada.3.STR.1 Terminology and features

Operational and Representation Attributes: The values of certain implementation-dependent characteristics can be obtained by querying the applicable attributes. Some attributes can be specified by the user; for example:

- X'Alignment: allows the alignment of objects on a storage unit boundary at an integral multiple of a specified value.
- X'Size: denotes the size in bits of the representation of the object.
- X'Component_Size: denotes the size in bits of components of the array type X.

Record Representation Clauses: provide a way to specify the layout of components within records, that is, their order, position, and size.

Storage Place Attributes: for a component of a record, the attributes (integer) Position, First_Bit and Last_Bit are used to specify the component position and size within the record.

Bit Ordering: Ada allows use of the attribute Bit_Order of a type to query or specify its bit ordering representation (High_Order_First and Low_Order_First). The default value is implementation defined and available at System.Bit_Order.

Atomic and Volatile: Ada can force every access to an object to be an indivisible access to the entity in memory instead of possibly partial, repeated manipulation of a local or register copy. In Ada, these properties are specified by **pragma**s.

Endianness: the programmer may specify the endianness of the representation through the use of a **pragma**.

Ada.3.STR.2 Description of vulnerability

In general, the type system of Ada protects against the vulnerabilities outlined in Section 6.STR. However, the use of Unchecked_Conversion, calling foreign language routines, and unsafe manipulation of address representations voids these guarantees.

The vulnerabilities caused by the inherent conceptual complexity of bit level programming are as described in Section 6.STR.

Ada.3.STR.3 Avoiding the vulnerability or mitigating its effects

The vulnerabilities associated with the complexity of bit-level programming can be mitigated by:

- The use of record and array types with the appropriate representation specifications added so that the objects are accessed by their logical structure rather than their physical representation. These representation specifications may address: order, position, and size of data components and fields.
- The use of pragma Atomic and pragma Atomic_Components to ensure that all updates to objects and components happen atomically.
- The use of pragma Volatile and pragma Volatile_Components to notify the compiler that objects and components must be read immediately before use as other devices or systems may be updating them between accesses of the program.
- The default object layout chosen by the compiler may be queried by the programmer to determine the expected behaviour of the final representation.

For the traditional approach to bit-level programming, Ada provides modular types and literal representations in arbitrary base from 2 to 16 to deal with numeric entities and correct handling of the sign bit. The use of **pragma** Pack on arrays of Booleans provides a type-safe way of manipulating bit strings and eliminates the use of error prone arithmetic operations.

Ada.3.STR.4 Implications for standardization

None

Ada.3.STR.5 Bibliography

None

Ada.3.PLF Floating-point Arithmetic [PLF]

Ada.3.PLF.1 Terminology and features

User-defined floating-point types: Types declared by the programmer that allow specification of digits of precision and optionally a range of values.

Static expressions: Expressions with statically known operands that are computed with exact precision by the compiler.

Fixed-point types: Real-valued types with a specified error bound (called the 'delta' of the type) that provide arithmetic operations carried out with fixed precision (rather than the relative precision of floating-point types).

Ada specifies adherence to the IEEE Floating Point Standards (IEEE-754-2008, IEEE-854-1987).

Ada.3.PLF.2 Description of vulnerability

The vulnerability in Ada is as described in Section 6.PLF.2.

Ada.3.PLF.3 Avoiding the vulnerability or mitigating its effects

- Rather than using predefined types, such as Float and Long_Float, whose precision may vary according to the target system, declare floating-point types that specify the required precision (e.g., digits 10). Additionally, specifying ranges of a floating point type enables constraint checks which prevents the propagation of infinities and NaNs.
- Avoid comparing floating-point values for equality. Instead, use comparisons that account for the approximate results of computations. Consult a numeric analyst when appropriate.
- Make use of static arithmetic expressions and static constant declarations when possible, since static expressions in Ada are computed at compile time with exact precision.
- Use Ada's standardized numeric libraries (e.g., Generic_Elementary_Functions) for common mathematical operations (trigonometric operations, logarithms, etc.).
- Use an Ada implementation that supports Annex G (Numerics) of the Ada standard, and employ the "strict mode" of that Annex in cases where additional accuracy requirements must be met by floating-point arithmetic and the operations of predefined numerics packages, as defined and guaranteed by the Annex.
- Avoid direct manipulation of bit fields of floating-point values, since such operations are generally target-specific and error-prone. Instead, make use of Ada's predefined floating-point attributes (e.g., 'Exponent).
- In cases where absolute precision is needed, consider replacement of floating-point types and operations with fixed-point types and operations.

Ada.3.PLF.4 Implications for standardization

None

Ada.3.PLF.5 Bibliography

IEEE 754-2008, IEEE Standard for Binary Floating Point Arithmetic, IEEE, 2008.

IEEE 854-1987, IEEE Standard for Radix-Independent Floating-Point Arithmetic, IEEE, 1987.

Ada.3.CCB Enumerator Issues [CCB]

Ada.3.CCB.1 Terminology and features

Enumeration Type: An enumeration type is a discrete type defined by an enumeration of its values, which may be named by identifiers or character literals. In Ada, the types Character and Boolean are enumeration types. The defining identifiers and defining character literals of an enumeration type must be distinct. The predefined order relations between values of the enumeration type follow the order of corresponding position numbers.

Enumeration Representation Clause: An enumeration representation clause may be used to specify the internal codes for enumeration literals.

Ada.3.CCB.2 Description of vulnerability

Enumeration representation specification may be used to specify non-default representations of an enumeration type, for example when interfacing with external systems. All of the values in the enumeration type must be defined in the enumeration representation specification. The numeric values of the representation must preserve the original order. For example:

```
type IO_Types is (Null_Op, Open, Close, Read,
                  Write, Sync);
for IO_Types use (Null_Op => 0, Open => 1,
                  Close => 2, Read => 4,
                  Write => 8, Sync => 16 );
```

An array may be indexed by such a type. Ada does not prescribe the implementation model for arrays indexed by an enumeration type with non-contiguous values. Two options exist: Either the array is represented “with holes” and indexed by the values of the enumeration type, or the array is represented contiguously and indexed by the position of the enumeration value rather than the value itself. In the former case, the vulnerability described in 6.CCB exists only if unsafe programming is applied to access the array or its components outside the protection of the type system. Within the type system, the semantics are well defined and safe. In the latter case, the vulnerability described in 6.CCB does not exist.

The full range of possible values of the expression in a **case** statement must be covered by the case choices. Two distinct choices of a case statement can not cover the same value. Choices can be expressed by single values or subranges of values. The **others** clause may be used as the last choice of a case statement to capture any remaining values of the case expression type that are not covered by the case choices. These restrictions are enforced at compile time. Identical rules apply to aggregates of arrays.

The remaining vulnerability is that unexpected values are captured by the **others** clause or a subrange as case choice after an additional enumeration literal has been added to the enumeration type definition. For example, when the range of the type Character was extended from 128 characters to the 256 characters in the Latin-1 character type, an **others** clause for a **case** statement with a Character type case expression originally written to capture cases associated with the 128 characters type now captures the 128 additional cases introduced by the extension of the type Character. Some of the new characters may have needed to be covered by the existing case choices or new case choices.

Ada.3.CCB.3 Avoiding the vulnerability or mitigating its effects

- For **case** statements and aggregates, do not use the **others** choice.
- For **case** statements and aggregates, mistrust subranges as choices after enumeration literals have been added anywhere but the beginning or the end of the enumeration type definition.

Ada.3.CCB.4 Implications for standardization

None

Ada.3.CCB.5 Bibliography

None

Ada.3.FLC Numeric Conversion Errors [FLC]

Ada.3.FLC.1 Terminology and features

User-defined scalar types: Types declared by the programmer for defining ordered sets of values of various kinds, namely integer, enumeration, floating-point, and fixed-point types. The typing rules of the language prevent intermixing of objects and values of distinct types.

Range check: A run-time check that ensures the result of an operation is contained within the range of allowable values for a given type or subtype, such as the check done on the operand of a type conversion.

Ada.3.FLC.2 Description of vulnerability

Ada does not permit implicit conversions between different numeric types, hence cases of implicit loss of data due to truncation cannot occur as they can in languages that allow type coercion between types of different sizes.

In the case of explicit conversions, range bound checks are applied, so no truncation can occur, and an exception will be generated if the operand of the conversion exceeds the bounds of the target type or subtype.

The occurrence of an exception on a conversion can disrupt a computation, which could potentially cause a failure mode or denial-of-service problems.

Ada permits the definition of subtypes of existing types that can impose a restricted range of values, and implicit conversions can occur for values of different subtypes belonging to the same type, but such conversions still involve range checks that prevent any loss of data or violation of the bounds of the target subtype.

Loss of precision can occur on explicit conversions from a floating-point type to an integer type, but in that case the loss of precision is being explicitly requested. Truncation cannot occur, and will lead to Constraint_Error if attempted.

There exist operations in Ada for performing shifts and rotations on values of unsigned types, but such operations are also explicit (function calls), so must be applied deliberately by the programmer, and can still only result in values that fit within the range of the result type of the operation.

Ada.3.FLC.3 Avoiding the vulnerability or mitigating its effects

- Use Ada's capabilities for user-defined scalar types and subtypes to avoid accidental mixing of logically incompatible value sets.

- Use range checks on conversions involving scalar types and subtypes to prevent generation of invalid data.
- Use static analysis tools during program development to verify that conversions cannot violate the range of their target.

Ada.3.FLC.4 Implications for standardization

None

Ada.3.FLC.5 Bibliography

None

Ada.3.CJM String Termination [CJM]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as strings in Ada are not delimited by a termination character. Ada programs that interface to languages that use null-terminated strings and manipulate such strings directly should apply the vulnerability mitigations recommended for that language.

Ada.3.XYX Boundary Beginning Violation [XYX]

Ada.3.XYX.1 Terminology and features

None

Ada.3.XYX.2 Description of vulnerability

With the exception of unsafe programming, this vulnerability is absent from Ada programs: all array indexing operations are checked automatically in Ada. The exception Constraint_Error is raised when an index value is outside the bounds of the array, and all array objects are created with explicit bounds. Pointer arithmetic cannot be used to index arrays, except through the use of unchecked conversions (see Section Ada.3.AMV). The language distinguishes arrays whose components are arrays, from multidimensional arrays, and the syntax reflects this distinction, Each index must respect the bounds of the corresponding dimension.

These bounds checks can be suppressed by means of the **pragma** Suppress, in which case the vulnerability applies; however, the presence of such **pragma**s is easily detected, and generally reserved for tight time-critical loops, even in production code.

Ada.3.XYX.3 Avoiding the vulnerability or mitigating its effects

- Do not suppress the checks provided by the language (see Section 5.9.5 of the Ada 95 Quality and Style Guide).
- Do not use unchecked conversion to manufacture index values.
- Use the attribute 'Range to describe iteration over arrays.
- Use subtypes to declare both array types and the index variables that will be used to access them.

Ada.3.XYX.4 Implications for standardization

None

Ada.3.XYX.5 Bibliography

None

Ada.3.XYZ Unchecked Array Indexing [XYZ]

Ada.3.XYZ.1 Terminology and features

None

Ada.3.XYZ.2 Description of vulnerability

All array indexing is checked automatically in Ada, and raises an exception when indexes are out of bounds. This is checked in all cases of indexing, including when arrays are passed to subprograms.

Programmers can write explicit bounds tests to prevent an exception when indexing out of bounds, but failure to do so does not result in accessing storage outside of arrays.

An explicit suppression of the checks can be requested by use of **pragma** Suppress, in which case the vulnerability would apply; however, such suppression is easily detected, and generally reserved for tight time-critical loops, even in production code.

Ada.3.XYZ.3 Avoiding the vulnerability or mitigating its effects

- Do not suppress the checks provided by the language.
- Use Ada's support for whole-array operations, such as for assignment and comparison, plus aggregates for whole-array initialization, to reduce the use of indexing.

Ada.3.XYZ.4 Implications for standardization

None

Ada.3.XYZ.5 Bibliography

None

Ada.3.XYW Unchecked Array Copying [XYW]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as Ada allows arrays to be copied by simple assignment (":="). The rules of the language ensure that no overflow can happen; instead, the exception Constraint_Error is raised if the target of the assignment is not able to contain the value assigned to it. Since array copy is provided by the language, Ada does not provide unsafe functions to copy structures by address and length.

Ada.3.XZB Buffer Overflow [XZB]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as this vulnerability can only happen as a consequence of unchecked array indexing or unchecked array copying, which do not occur in Ada (see Ada.3.XYZ and Ada.3.XYW).

Ada.3.HFC Pointer Casting and Pointer Type Changes [HFC]

Ada.3.HFC.1 Terminology and features

The mechanisms available in Ada to alter the type of a pointer value are unchecked type conversions and type conversions involving pointer types derived from a common root type. In addition, uses of the unchecked address taking capabilities can create pointer types that misrepresent the true type of the designated entity (see Section 13.10 of the Ada Language Reference Manual).

Ada.3.HFC.2 Description of vulnerability

The vulnerabilities described in Section 6.HFC exist in Ada only if unchecked type conversions or unsafe taking of addresses are applied (see Section Ada.2). Other permitted type conversions can never misrepresent the type of the designated entity.

Checked type conversions that affect the application semantics adversely are possible.

Ada.3.HFC.3 Avoiding the vulnerability or mitigating its effects

- This vulnerability can be avoided in Ada by not using the features explicitly identified as unsafe.
- Use 'Access which is always type safe.

Ada.3.HFC.4 Implications for standardization

None

Ada.3.HFC.5 Bibliography

None

Ada.3.RVG Pointer Arithmetic [RVG]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as Ada does not allow pointer arithmetic.

Ada.3.XYH Null Pointer Dereference [XYH]

In Ada, this vulnerability does not exist, since compile-time or run-time checks ensure that no null value can be dereferenced.

Ada provides an optional qualification on access types that specifies and enforces that objects of such types cannot have a null value. Non-nullness is enforced by rules that statically prohibit the assignment of either **null** or values from sources not guaranteed to be non-null.

Ada.3.XYK Dangling Reference to Heap [XYK]

Ada.3.XYK.1 Terminology and features

Ada provides a model in which whole collections of heap-allocated objects can be deallocated safely, automatically and collectively when the scope of the root access type ends.

For global access types, allocated objects can only be deallocated through an instantiation of the generic procedure Unchecked_Deallocation.

Ada.3.XYK.2 Description of vulnerability

Use of Unchecked_Deallocation can cause dangling references to the heap. The vulnerabilities described in 6.XYK exist in Ada, when this feature is used, since Unchecked_Deallocation may be applied even though there are outstanding references to the deallocated object.

Ada.3.XYK.3 Avoiding the vulnerability or mitigating its effects

- Use local access types where possible.

- Do not use Unchecked_Deallocation.
- Use Controlled types and reference counting.

Ada.3.XYK.4 Implications for standardization

None

Ada.3.XYK.5 Bibliography

None

Ada.3.SYM Templates and Generics [SYM]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as the Ada generics model is based on imposing a contract on the structure and operations of the types that can be used for instantiation. Also, explicit instantiation of the generic is required for each particular type.

Therefore, the compiler is able to check the generic body for programming errors, independently of actual instantiations. At each actual instantiation, the compiler will also check that the instantiated type meets all the requirements of the generic contract.

Ada also does not allow for 'special case' generics for a particular type, therefore behaviour is consistent for all instantiations.

Ada.3.RIP Inheritance [RIP]

Ada.3.RIP.1 Terminology and features

Overriding Indicators: If an operation is marked as "overriding", then the compiler will flag an error if the operation is incorrectly named or the parameters are not as defined in the parent. Likewise, if an operation is marked as "not overriding", then the compiler will verify that there is no operation being overridden in parent types.

Ada.3.RIP.2 Description of vulnerability

The vulnerability documented in Section 6.RIP applies to Ada.

Ada only allows a restricted form of multiple inheritance, where only one of the multiple ancestors (the parent) may define operations. All other ancestors (interfaces) can only specify the operations' signature. Therefore, Ada does not suffer from multiple inheritance derived vulnerabilities.

Ada.3.RIP.3 Avoiding the vulnerability or mitigating its effects

- Use the overriding indicators on potentially inherited subprograms to ensure that the intended contract is obeyed, thus preventing the accidental redefinition or failure to redefine an operation of the parent.
- Use the mechanisms of mitigation described in the main body of the document.

Ada.3.RIP.4 Implications for standardization

Provide mechanisms to prevent further extensions of a type hierarchy.

Ada.3.RIP.5 Bibliography

None

Ada.3.LAV Initialization of Variables [LAV]

Ada.3.LAV.1 Terminology and features

None

Ada.3.LAV.2 Description of vulnerability

In Ada, referencing an uninitialized scalar (numeric or enumeration-type) variable is considered a bounded error, with the possible outcomes being the raising of a Program_Error or Constraint_Error exception, or continuing execution with some value of the variable's type, or some other implementation-defined behaviour. The implementation is required to prevent an uninitialized variable used as an array index resulting in updating memory outside the array. Similarly, using an uninitialized variable in a **case** statement cannot result in a jump to something other than one of the case alternatives. Typically Ada implementations keep track of which variables might be uninitialized, and presume they contain any value possible for the given size of the variable, rather than presuming they are within whatever value range that might be associated with their declared type or subtype. The vulnerability associated with use of an uninitialized scalar variable is therefore that some result will be calculated incorrectly or an exception will be raised unexpectedly, rather than a completely undefined behaviour.

Pointer variables are initialized to null by default, and every dereference of a pointer is checked for a **null** value. Therefore the only vulnerability associated with pointers is that a Constraint_Error might be raised if a pointer is dereferenced that was not correctly initialized.

In general in Ada it is possible to suppress run-time checking, using **pragma** Suppress. In the presence of such a pragma, if a condition arises that would have resulted in a check failing and an exception being raised, then the behaviour is completely undefined ("erroneous" in Ada terms), and could include updating random memory or execution of unintended machine instructions.

Ada provides a generic function for unchecked conversion between (sub)types. If an uninitialized variable is passed to an instance of this generic function and the value is not within the declared range of the target subtype, then the subsequent execution is erroneous.

Failure can occur when a scalar variable (including a scalar component of a composite variable) is not initialized at its point of declaration, and there is a reference to the value of the variable on a path that never assigned to the variable. The effects are bounded as described above, with the possible effect being an incorrect result or an unexpected exception.

Ada.3.LAV.3 Avoiding the vulnerability or mitigating its effects

Scalar variables are not initialized by default in Ada. Pointer types are default-initialized to null. Default initialization for record types may be specified by the user. For controlled types (those descended from the language-defined type Controlled or Limited_Controlled), the user may also specify an Initialize procedure which is invoked on all default-initialized objects of the type.

This vulnerability can be avoided or mitigated in Ada in the following ways:

- Whenever possible, a variable should be replaced by an initialized constant, if in fact there is only one assignment to the variable, and the assignment can be performed at the point of initialization. Moving the object declaration closer to its point of use by creating a local declare block can increase the frequency at which such a replacement is possible. Note that initializing a variable with an inappropriate default value such as zero can result in hiding underlying problems, because static analysis tools or the compiler itself will then be unable to identify use before correct initialization.
- If the compiler has a mode that detects use before initialization, then this mode should be enabled and any such warnings should be treated as errors.
- The **pragma** Normalize_Scalars can be used to ensure that scalar variables are always initialized by the compiler in a repeatable fashion. This **pragma** is designed to initialize variables to an out-of-range value if there is one, to avoid hiding errors.

Ada.3.LAV.4 Implications for standardization

Some languages (e.g., Java) require that all local variables either be initialized at the point of declaration or on all paths to a reference. Such a rule could be considered for Ada.

Ada.3.LAV.5 Bibliography

None

Ada.3.XYY Wrap-around Error [XYY]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as wrap-around arithmetic in Ada is limited to modular types Arithmetic operations on such types use modulo arithmetic, and thus no such operation can create an invalid value of the type.

Ada raises the predefined exception Constraint_Error whenever an attempt is made to increment an integer above its maximum positive value or to decrement an integer below its maximum negative value. Operations to shift and rotate numeric values apply only to modular integer types, and always produce values that belong to the type. In Ada there is no confusion between logical and arithmetic shifts.

Ada.3.XZI Sign Extension Error [XZI]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as Ada does not, explicitly or implicitly, allow unsigned extension operations to apply to signed entities or vice-versa.

Ada.3.JCW Operator Precedence/Order of Evaluation [JCW]

Ada.3.JCW.1 Terminology and features

None

Ada.3.JCW.2 Description of vulnerability

Since this vulnerability is about "incorrect beliefs" of programmers, there is no way to establish a limit to how far incorrect beliefs can go. However, Ada is less susceptible to that vulnerability than many other languages, since

- Ada only has six levels of precedence and associativity is closer to common expectations. For example, an expression like A = B or C = D will be parsed as expected, i.e. (A = B) or (C = D).

- Mixed logical operators are not allowed without parentheses, i.e., "A or B or C" is legal, as well as "A and B and C", but "A and B or C" is not (must write "(A and B) or C" or "A and (B or C)".
- Assignment is not an operator in Ada.

Ada.3.JCW.3 Avoiding the vulnerability or mitigating its effects

The general mitigation measures can be applied to Ada like any other language.

Ada.3.JCW.4 Implications for standardization

None

Ada.3.JCW.5 Bibliography

None

Ada.3.SAM Side-effects and Order of Evaluation [SAM]

Ada.3.SAM.1 Terminology and features

None

Ada.3.SAM.2 Description of vulnerability

There are no operators in Ada with direct side effects on their operands using the language-defined operations, especially not the increment and decrement operation. Ada does not permit multiple assignments in a single expression or statement.

There is the possibility though to have side effects through function calls in expressions where the function modifies globally visible variables. Although functions only have "**in**" parameters, meaning that they are not allowed to modify the value of their parameters, they may modify the value of global variables. Operators in Ada are functions, so, when defined by the user, although they cannot modify their own operands, they may modify global state and therefore have side effects.

Ada allows the implementation to choose the association of the operators with operands of the same precedence level (in the absence of parentheses imposing a specific association). The operands of a binary operation are also evaluated in an arbitrary order, as happens for the parameters of any function call. In the case of user-defined operators with side effects, this implementation dependency can cause unpredictability of the side effects.

Ada.3.SAM.3 Avoiding the vulnerability or mitigating its effects

- Make use of one or more programming guidelines which prohibit functions that modify global state, and can be enforced by static analysis.
- Keep expressions simple. Complicated code is prone to error and difficult to maintain.
- Always use brackets to indicate order of evaluation of operators of the same precedence level.

Ada.3.SAM.4 Implications for standardization

Add the ability to declare in the specification of a function that it is pure, i.e., it has no side effects.

Ada.3.SAM.5 Bibliography

None

Ada.3.KOA Likely Incorrect Expression [KOA]

Ada.3.KOA.1 Terminology and features

None

Ada.3.KOA.2 Description of vulnerability

An instance of this vulnerability consists of two syntactically similar constructs such that the inadvertent substitution of one for the other may result in a program which is accepted by the compiler but does not reflect the intent of the author.

The examples given in 6.KOA are not problems in Ada because of Ada's strong typing and because an assignment is not an expression in Ada.

In Ada, a type conversion and a qualified expression are syntactically similar, differing only in the presence or absence of a single character:

```
Type_Name (Expression) -- a type conversion
```

vs.

```
Type_Name'(Expression) -- a qualified expression
```

Typically, the inadvertent substitution of one for the other results in either a semantically incorrect program which is rejected by the compiler or in a program which behaves in the same way as if the intended construct had been written. In the case of a constrained array subtype, the two constructs differ in their treatment of sliding (conversion of an array value with bounds 100 .. 103 to a subtype with bounds 200 .. 203 will succeed; qualification will fail a run-time check.

Similarly, a timed entry call and a conditional entry call with an else-part that happens to begin with a **delay** statement differ only in the use of "**else**" vs. "**or**" (or even "**then abort**" in the case of a asynchronous_select statement).

Probably the most common correctness problem resulting from the use of one kind of expression where a syntactically similar expression should have been used has to do with the use of short-circuit vs. non-short-circuit Boolean-valued operations (i.e., "**and then**" and "**or else**" vs. "**and**" and "**or**"), as in

```
if (Ptr /= null) and (Ptr.all.Count > 0) then ... end if;
  -- should have used "and then" to avoid
  dereferencing null
```

Ada.3.KOA.3 Avoiding the vulnerability or mitigating its effects

- Compilers and other static analysis tools can detect some cases (such as the preceding example) where short-circuited evaluation could prevent the failure of a run-time check.

- Developers may also choose to use short-circuit forms by default (errors resulting from the incorrect use of short-circuit forms are much less common), but this makes it more difficult for the author to express the distinction between the cases where short-circuited evaluation is known to be needed (either for correctness or for performance) and those where it is not.

Ada.3.KOA.4 Implications for standardization

None

Ada.3.KOA.5 Bibliography

None

Ada.3.XYQ Dead and Deactivated Code [XYQ]

Ada.3.XYQ.1 Terminology and features

None

Ada.3.XYQ.2 Description of vulnerability

Ada allows the usual sources of dead code (described in 6.XYQ.3) that are common to most conventional programming languages.

Ada.3.XYQ.3 Avoiding the vulnerability or mitigating its effects

Implementation specific mechanisms may be provided to support the elimination of dead code. In some cases, **pragma**s such as Restrictions, Suppress, or Discard_Names may be used to inform the compiler that some code whose generation would normally be required for certain constructs would be dead because of properties of the overall system, and that therefore the code need not be generated.

Ada.3.XYQ.4 Implications for standardization

None

Ada.3.XYQ.5 Bibliography

None

Ada.3.CLL Switch Statements and Static Analysis [CLL]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as
Ada requires that a case statement provide exactly one alternative for each value of the expression's subtype. If the value of the expression is outside of the range of this subtype (e.g., due to an uninitialized variable), then the resulting behaviour is well-defined (Constraint_Error is raised). Control does not flow from one alternative to the next. Upon reaching the end of an alternative, control is transferred to the end of the **case** statement.

Ada.3.EOJ Demarcation of Control Flow [EOJ]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as the Ada syntax describes several types of compound statements that are associated with control flow including **if** statements, **loop** statements, **case** statements, **select** statements, and extended **return** statements. Each of these forms of compound statements require unique syntax that marks the end of the compound statement.

Ada.3.TEX Loop Control Variables [TEX]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as Ada defines a **for loop** where the number of iterations is controlled by a loop control variable (called a loop parameter). This value has a constant view and cannot be updated within the sequence of statements of the body of the loop.

Ada.3.XZH Off-by-one Error [XZH]

Ada.3.XZH.1 Terminology and features

Scalar Type: A Scalar type comprises enumeration types, integer types, and real types.

Attribute: An Attribute is a characteristic of a declaration that can be queried by special syntax to return a value corresponding to the requested attribute.

The language defined attributes applicable to scalar types and array types that help address this vulnerability are 'First, 'Last, 'Range, and 'Length.

Ada.3.XZH.2 Description of vulnerability

Confusion between the need for < and <= or > and >= in a test.

A **for loop** in Ada does not involve the programmer having to specify a conditional test for loop termination. Instead, the starting and ending value of the loop are specified which eliminates this source of off by one errors. A **while loop** however, lets the programmer specify the loop termination expression, which could be susceptible to an off by one error.

Confusion as to the index range of an algorithm.

Although there are language defined attributes to symbolically reference the start and end values for a loop iteration, the language does allow the use of explicit values and loop termination tests. Off-by-one errors can result in these circumstances.

Care should be taken when using the 'Length Attribute in the loop termination expression. The expression should generally be relative to the 'First value.

The strong typing of Ada eliminates the potential for buffer overflow associated with this vulnerability. If the error is not statically caught at compile time, then a run time check generates an exception if an attempt is made to access an element outside the bounds of an array.

Failing to allow for storage of a sentinel value.

Ada does not use sentinel values to terminate arrays. There is no need to account for the storage of a sentinel value, therefore this particular vulnerability concern does not apply to Ada.

Ada.3.XZH.3 Avoiding the vulnerability or mitigating its effects

- Whenever possible, a **for loop** should be used instead of a **while loop**.
- Whenever possible, the 'First, 'Last, and 'Range attributes should be used for loop termination. If the 'Length attribute must be used, then extra care should be taken to ensure that the length expression considers the starting index value for the array.

Ada.3.XZH.4 Implications for standardization

None

Ada.3.XZH.5 Bibliography

None

Ada.3.EWD Structured Programming [EWD]

Ada.3.EWD.1 Terminology and features

None

Ada.3.EWD.2 Description of vulnerability

Ada programs can exhibit many of the vulnerabilities noted in the parent report: leaving a **loop** at an arbitrary point, local jumps (**goto**), and multiple exit points from subprograms.

It does not suffer from non-local jumps and multiple entries to subprograms.

Ada.3.EWD.3 Avoiding the vulnerability or mitigating its effects

Avoid the use of **goto**, **loop exit** statements, **return** statements in **procedure**s and more than one **return** statement in a **function**.

Ada.3.EWD.4 Implications for standardization

Pragma Restrictions could be extended to allow the use of these features to be statically checked.

Ada.3.EWD.5 Bibliography

None

Ada.3.CSJ Passing Parameters and Return Values [CSJ]

Ada.3.CSJ.1 Terminology and features

None

Ada.3.CSJ.2 Description of vulnerability

Ada employs the mechanisms (e.g., modes **in**, **out** and **in out**) that are recommended in Section 6.CSJ. These mode definitions are not optional, mode **in** being the default. The remaining vulnerability is aliasing when a large object is passed by reference.

Ada.3.CSJ.3 Avoiding the vulnerability of mitigating its effects

- Follow avoidance advice in Section 6.CSJ.

Ada.3.CSJ.4 Implications for standardization

None

Ada.3.CSJ.5 Bibliography

None

Ada.3.DCM Dangling References to Stack Frames [DCM]

Ada.3.DCM.1 Terminology and features

In Ada, the attribute 'Address yields a value of some system-specific type that is not equivalent to a pointer. The attribute 'Access provides an access value (what other languages call a pointer). Addresses and access values are not automatically convertible, although a predefined set of generic functions can be used to convert one into the other. Access values are typed, that is to say can only designate objects of a particular type or class of types.

Ada.3.DCM.2 Description of vulnerability

As in other languages, it is possible to apply the 'Address attribute to a local variable, and to make use of the resulting value outside of the lifetime of the variable. However, 'Address is very rarely used in this fashion in Ada. Most commonly, programs use 'Access to provide pointers to static objects, and the language enforces accessibility checks whenever code attempts to use this attribute to provide access to a local object outside of its scope. These accessibility checks eliminate the possibility of dangling references.

As for all other language-defined checks, accessibility checks can be disabled over any portion of a program by using the Suppress **pragma**. The attribute Unchecked_Access produces values that are exempt from accessibility checks.

Ada.3.DCM.3 Avoiding the vulnerability or mitigating its effects

- Only use 'Address attribute on static objects (e.g., a register address).
- Do not use 'Address to provide indirect untyped access to an object.
- Do not use conversion between Address and access types.
- Use access types in all circumstances when indirect access is needed.
- Do not suppress accessibility checks.
- Avoid use of the attribute Unchecked_Access.

Ada.3.DCM.4 Implications for standardization

Pragma Restrictions could be extended to restrict the use of 'Address attribute to library level static objects.

Ada.3.DCM.5 Bibliography

None

Ada.3.OTR Subprogram Signature Mismatch [OTR]

Ada.3.OTR.1 Terminology and features

Default expression: an expression of the formal object type that may be used to initialize the formal object if an actual object is not provided.

Ada.3.OTR.2 Description of vulnerability

There are two concerns identified with this vulnerability. The first is the corruption of the execution stack due to the incorrect number or type of actual parameters. The second is the corruption of the execution stack due to calls to externally compiled modules.
In Ada, at compilation time, the parameter association is checked to ensure that the type of each actual parameter is the type of the corresponding formal parameter. In addition, the formal parameter specification may include default expressions for a parameter. Hence, the procedure may be called with some actual parameters missing. In this case, if there is a default expression for the missing parameter, then the call will be compiled without any errors. If default expressions are not specified, then the procedure call with insufficient actual parameters will be flagged as an error at compilation time.

Caution must be used when specifying default expressions for formal parameters, as their use may result in successful compilation of subprogram calls with an incorrect signature. The execution stack will not be corrupted in this event but the program may be executing with unexpected values.

When calling externally compiled modules that are Ada program units, the type matching and subprogram interface signatures are monitored and checked as part of the compilation and linking of the full application. When calling externally compiled modules in other programming languages, additional steps are needed to ensure that the number and types of the parameters for these external modules are correct.

Ada.3.OTR.3 Avoiding the vulnerability or mitigating its effects

- Do not use default expressions for formal parameters.

- Interfaces between Ada program units and program units in other languages can be managed using **pragma** Import to specify subprograms that are defined externally and **pragma** Export to specify subprograms that are used externally. These **pragma**s specify the imported and exported aspects of the subprograms, this includes the calling convention. Like subprogram calls, all parameters need to be specified when using **pragma** Import and **pragma** Export.
- The **pragma** Convention may be used to identify when an Ada entity should use the calling conventions of a different programming language facilitating the correct usage of the execution stack when interfacing with other programming languages.
- In addition, the Valid attribute may be used to check if an object that is part of an interface with another language has a valid value and type.

Ada.3.OTR.4 Implications for standardization

None

Ada.3.OTR.5 Bibliography

None

Ada.3.GDL Recursion [GDL]

Ada.3.GDL.1 Terminology and features

None

Ada.3.GDL.2 Description of vulnerability

Ada permits recursion. The exception Storage_Error is raised when the recurring execution results in insufficient storage.

Ada.3.GDL.3 Avoiding the vulnerability or mitigating its effects

If recursion is used, then a Storage_Error exception handler may be used to handle insufficient storage due to recurring execution.

Alternatively, the asynchronous control construct may be used to time the execution of a recurring call and to terminate the call if the time limit is exceeded.

In Ada, the **pragma** Restrictions may be invoked with the parameter No_Recursion. In this case, the compiler will ensure that as part of the execution of a subprogram the same subprogram is not invoked.

Ada.3.GDL.4 Implications for standardization

None

Ada.3.GDL.5 Bibliography

None

Ada.3.NZN Returning Error Status [NZN]

Ada.3.NZN.1 Terminology and features

None

Ada.3.NZN.2 Description of vulnerability

Ada offers a set of predefined exceptions for error conditions that may be detected by checks that are compiled into a program. In addition, the programmer may define exceptions that are appropriate for their application. These exceptions are handled using an exception handler. Exceptions may be handled in the environment where the exception occurs or may be propagated out to an enclosing scope.

As described in Section 6.NZN, there is some complexity in understanding the exception handling methodology especially with respect to object-oriented programming and multi-threaded execution.

Ada.3.NZN.3 Avoiding the vulnerability or mitigating its effects

In addition to the mitigations defined in the main text, values delivered to an Ada program from an external device may be checked for validity prior to being used. This is achieved by testing the Valid attribute.

Ada.3.NZN.4 Implications for standardization

None

Ada.3.NZN.5 Bibliography

None

Ada.3.REU Termination Strategy [REU]

Ada.3.REU.1 Terminology and features

Ada.3.REU.2 Description of Vulnerability

An Ada system that consists of multiple tasks is subject to the same hazards as multithreaded systems in other languages. A task that fails, for example, because its execution violates a language-defined check, terminates quietly.

Any other task that attempts to communicate with a terminated task will receive the exception Tasking_Error. The undisciplined use of the **abort** statement or the asynchronous transfer of control feature may destroy the functionality of a multitasking program.

Ada.3.REU.3 Avoiding the vulnerability or mitigating its effects

- Include exception handlers for every task, so that their unexpected termination can be handled and possibly communicated to the execution environment.
- Use objects of controlled types to ensure that resources are properly released if a task terminates unexpectedly.
- The **abort** statement should be used sparingly, if at all.
- For high-integrity systems, exception handling is usually forbidden. However, a top-level exception handler can be used to restore the overall system to a coherent state.
- Define interrupt handlers to handle signals that come from the hardware or the operating system. This mechanism can also be used to add robustness to a concurrent program.
- Annex C of the Ada Reference Manual (Systems Programming) defines the package Ada.Task_Termination to be used to monitor task termination and its causes.
- Annex H of the Ada Reference Manual (High Integrity Systems) describes several **pragma**, restrictions, and other language features to be used when writing systems for high-reliability applications. For example, the **pragma** Detect_Blocking forces an implementation to detect a potentially blocking operation within a protected operation, and to raise an exception in that case.

Ada.3.REU.4 Implications for Standardization

None

Ada.3.REU.5 Bibliography

None

Ada 3.LRM Extra Intrinsics [LRM]

Ada 3.LRM.1 Terminology and features

The **pragma** Convention can specify that a given subprogram is intrinsic. The implementation of an intrinsic subprogram is known to the compiler, and the user does not specify a body for it.

Ada 3.LRM.2 Description of Vulnerability

The vulnerability does not apply to Ada, because all subprograms, whether intrinsic or not, belong to the same name space. This means that all subprograms must be explicitly declared, and the same name resolution rules apply to all of them, whether they are predefined or user-defined. If two subprograms with the same name and signature are visible (that is to say nameable) at the same place in a program, then a call using that name will be rejected as ambiguous by the compiler, and the programmer will have to specify (for example by means of a qualified name) which subprogram is meant.

Ada 3.AMV Type-breaking Reinterpretation of Data [AMV]

Ada 3.AMV.1 Terminology and features

Ada provides a generic function Unchecked_Conversion, whose purpose is to impose a given target type on a value of a distinct source type. This function must be instantiated for each pair of types between which such a reinterpretation is desired.

Ada also provides Address clauses that can be used to overlay objects of different types.
Variant records in Ada are discriminated types; the discriminant is part of the object and supports consistency checks when accessing components of a given variant. In addition, for inter-language communication, Ada also provides the **pragma** Unchecked_Union to indicate that objects of a given variant type do not store their discriminants. Objects of such types are in fact free unions.

Ada 3.AMV.2 Description of vulnerability

Unchecked_Conversion can be used to bypass the type-checking rules, and its use is thus unsafe, as in any other language. The same applies to the use of Unchecked_Union, even though the language specifies various inference rules that the compiler must use to catch statically detectable constraint violations.

Type reinterpretation is a universal programming need, and no usable programming language can exist without some mechanism that bypasses the type model. Ada provides these mechanisms with some additional safeguards, and makes their use purposely verbose, to alert the writer and the reader of a program to the presence of an unchecked operation.

Ada 3.AMV.3 Avoiding the vulnerability or mitigating its effects

- The fact that Unchecked_Conversion is a generic function that must be instantiated explicitly (and given a meaningful name) hinders its undisciplined use, and places a loud marker in the code wherever it is used. Well-written Ada code will have a small set of instantiations of Unchecked_Conversion.

- Most implementations require the source and target types to have the same size in bits, to prevent accidental truncation or sign extension.
- Unchecked_Union should only be used in multi-language programs that need to communicate data between Ada and C or C++. Otherwise the use of discriminated types prevents "punning" between values of two distinct types that happen to share storage.
- Using address clauses to obtain overlays should be avoided. If the types of the objects are the same, then a renaming declaration is preferable. Otherwise, the **pragma** Import should be used to inhibit the initialization of one of the entities so that it does not interfere with the initialization of the other one.

Ada 3.AMV.4 Implications for Standardization

None

Ada 3.AMV.5 Bibliography

None

Ada.3.XYL Memory Leak [XYL]

Ada.3.XYL.1 Terminology and features

Allocator: The Ada term for the construct that allocates storage from the heap or from a storage pool.

Storage Pool: A named location in an Ada program where all of the objects of a single access type will be allocated. A storage pool can be sized exactly to the requirements of the application by allocating only what is needed for all objects of a single type without using the centrally managed heap. Exceptions raised due to memory failures in a storage pool will not adversely affect storage allocation from other storage pools or from the heap and do not suffer from fragmentation.

The following Ada restrictions prevent the application from using any allocators:

> **pragma** Restrictions(No_Allocators): prevents the use of allocators.
>
> **pragma** Restrictions(No_Local_Allocators): prevents the use of allocators after the main program has commenced.
>
> **pragma** Restrictions(No_Implicit_Heap_Allocations): prevents the use of allocators that would use the heap, but permits allocations from storage pools.
>
> **pragma** Restrictions(No_Unchecked_Deallocations): prevents allocated storage from being returned and hence effectively enforces storage pool memory approaches or a completely static approach to access types. Storage pools are not affected by this restriction as explicit routines to free memory for a storage pool can be created.

Ada.3.XYL.2 Description of vulnerability

For objects that are allocated from the heap without the use of reference counting, the memory leak vulnerability is possible in Ada. For objects that must allocate from a storage pool, the vulnerability can be present but is restricted to the single pool and which makes it easier to detect by verification. For objects that are objects of a controlled type that uses referencing counting and that are not part of a cyclic reference structure, the vulnerability does not exist.

Ada does not mandate the use of a garbage collector, but Ada implementations are free to provide such memory reclamation. For applications that use and return memory on an implementation that provides garbage collection, the issues associated with garbage collection exist in Ada.

Ada.3.XYL.3 Avoiding the vulnerability or mitigating its effects

- Use storage pools where possible.
- Use controlled types and reference counting to implement explicit storage management systems that cannot have storage leaks.
- Use a completely static model where all storage is allocated from global memory and explicitly managed under program control.

Ada.3.XYL.4 Implications for standardization

Future Standardization of Ada should consider implementing a language-provided reference counting storage management mechanism for dynamic objects.

Ada.3.XYL.5 Bibliography

None

Ada.3.TRJ Argument Passing to Library Functions [TRJ]

Ada.3.TRJ.1 Terminology and features

Separate Compilation: Ada requires that calls on libraries are checked for illegal situations as if the called routine were declared locally.

Ada.3.TRJ.2 Description of vulnerability

The general vulnerability that parameters might have values precluded by preconditions of the called routine applies to Ada as well.

However, to the extent that the preclusion of values can be expressed as part of the type system of Ada, the preconditions are checked by the compiler statically or dynamically and thus are no longer vulnerabilities. For example, any range constraint on values of a parameter can be expressed in Ada by means of type or subtype declarations. Type violations are detected at compile time, subtype violations cause runtime exceptions.

Ada.3.TRJ.3 Avoiding the vulnerability or mitigating its effects

- Exploit the type and subtype system of Ada to express preconditions (and postconditions) on the values of parameters.
- Document all other preconditions and ensure by guidelines that either callers or callees are responsible for checking the preconditions (and postconditions). Wrapper subprograms for that purpose are particularly advisable.
- Specify the response to invalid values.

Ada.3.TRJ.4 Implications for standardization

Future standardization of Ada should consider support for arbitrary pre- and postconditions.

Ada.3.TRJ.5 Bibliography

None

Ada.3.NYY Dynamically-linked Code and Self-modifying Code [NYY]

With the exception of unsafe programming, this vulnerability is not applicable to Ada as Ada supports neither dynamic linking nor self-modifying code. The latter is possible only by exploiting other vulnerabilities of the language in the most malicious ways and even then it is still very difficult to achieve.

Ada.3.NSQ Library Signature [NSQ]

Ada.3.NSQ.1 Terminology and features

None

Ada.3.NSQ.2 Description of vulnerability

Ada provides mechanisms to explicitly interface to modules written in other languages. Pragmas Import, Export and Convention permit the name of the external unit and the interfacing convention to be specified.

Even with the use of **pragma** Import, **pragma** Export and **pragma** Convention the vulnerabilities stated in Section 6.NSQ are possible. Names and number of parameters change under maintenance; calling conventions change as compilers are updated or replaced, and languages for which Ada does not specify a calling convention may be used.

Ada.3.NSQ.3 Avoiding the vulnerability or mitigating its effects

The mitigation mechanisms of Section 6.NSQ.5 are applicable.

Ada.3.NSQ.4 Implications for standardization

Ada standardization committees can work with other programming language standardization committees to define library interfaces that include more than a program calling interface. In particular, mechanisms to qualify and quantify ranges of behaviour, such as pre-conditions, post-conditions and invariants, would be helpful.

Ada.3.NSQ.5 Bibliography

None

Ada.3.HJW Unanticipated Exceptions from Library Routines [HJW]

Ada.3.HJW.1 Terminology and features

None

Ada.3.HJW.2 Description of vulnerability

Ada programs are capable of handling exceptions at any level in the program, as long as any exception naming and delivery mechanisms are compatible between the Ada program and the library components. In such cases the normal Ada exception handling processes will apply, and either the calling unit or some subprogram or task in its call chain will catch the exception and take appropriate programmed action, or the task or program will terminate.

If the library components themselves are written in Ada, then Ada's exception handling mechanisms let all called units trap any exceptions that are generated and return error conditions instead. If such exception handling mechanisms are not put in place, then exceptions can be unexpectedly delivered to an caller.

If the interface between the Ada units and the library routine being called does not adequately address the issue of naming, generation and delivery of exceptions across the interface, then the vulnerabilities as expressed in Section 6.HJW apply.

Ada.3.HJW.3 Avoiding the vulnerability or mitigating its effects

- Ensure that the interfaces with libraries written in other languages are compatible in the naming and generation of exceptions.
- Put appropriate exception handlers in all routines that call library routines, including the catch-all exception handler **when others** =>.
- Document any exceptions that may be raised by any Ada units being used as library routines.

Ada.3.HJW.4 Implications for standardization

Ada standardization committees can work with other programming language standardization committees to define library interfaces that include more than a program calling interface. In particular, mechanisms to qualify and quantify ranges of behaviour, such as pre-conditions, post-conditions and invariants, would be helpful.

Ada.3.HJW.5 Bibliography

None

Ada Gems

The following contributions are taken from the AdaCore Gem of the Week series. The full collection of gems, discussion and related files, can be found at http://www.adacore.com/category/developers-center/gems/.

Gem #82: Type-Based Security 1: Handling Tainted Data

Yannick Moy, AdaCore

Date: 22 March 2010

Abstract: The strong type system in Ada makes it quite convenient to check at compile time that certain security properties are verified, for example that a tainted value is not used where a trusted one is expected, or that data is properly validated before being used in a sensitive context (think of SQL injection attacks).

In this series of two Gems, we present short examples of how this might be done. The first Gem discusses how to handle tainted data.

Let’s get started…

The notions of tainted data and trusted data usually refer to data coming from the user vs. data coming from the application. Tainting is viral, in that any result of a computation where one of the operands is tainted becomes tainted too.

Various C/C++ static analyzers provide checkers for tainted data that help find bugs where data from the user serves to compute the size of an allocation, so that an attacker could use this to trigger a buffer overflow leading to an Elevation of Privilege (EoP) attack.

In Ada, the compiler can provide the guarantee that no such bugs have been introduced by accident (although you can still bypass the rule if you really want to, for example by using Unchecked_Conversion or address clause overlays), provided different types are used for tainted and trusted data, with no run-time penalty. This can be done with many types of data, including basic types like integers.

Let’s say tainted data is of an integer type. The basic idea is to derive the trusted type from the tainted one, and to provide a function Value to get to the raw data inside a trusted value, like the following:

```
package Taint is
  type Trusted_Value is new Integer;
  function Value (V : Trusted_Value) return Integer;
  pragma Inline(Value);
end Taint;
```

Notice that the implementation of Value is just a type conversion:

```
package body Taint is
  function Value (V : Trusted_Value) return Integer is
  begin
    return Integer (V);
  end Value;
end Taint;
```

Then, make sure the sensitive program uses trusted data:

```
with Taint; use Taint;
procedure Sensitive (X : Trusted_Value) is
begin
  null; -- Do something sensitive with value X
end Sensitive;
```

Let’s try to pass in data from the user to the sensitive program:

```
with Taint;
with Sensitive;
procedure Bad (Some_Value : Integer) is
begin
  Sensitive (Some_Value);
end Bad;
```

The compiler returns with a type error:

```
bad.adb:6:15: expected type "Trusted_Value" defined at
taint.ads:3
bad.adb:6:15: found type "Standard.Integer"
```

Now, this does not prevent us from doing useful computations on trusted data as easily as on tainted data, including initialization with literals, case statements, array indexing, etc.

```
with Taint; use Taint;
with Sensitive;
procedure Good is
  Max_Value : constant := 100;
  X : Trusted_Value := Max_Value;
begin
  X := X + 1; -- Perform any computations on X
  Sensitive (X);
end Good;
```

Because Trusted_Value is a type derived from the tainted type (Integer), all operations allowed on tainted data are also allowed on trusted data, but operations mixing them are not allowed.

Be aware that nothing prevents the program itself from converting between tainted data and trusted data freely, but this requires inserting an explicit conversion, which can be spotted during code reviews.

To completely prevent such unintended conversions (say, to facilitate maintenance), the type used for trusted data must be made private, so that only the package which defines it can convert to and from it. With Trusted_Value being private, we should also provide a corresponding function for each literal which we used previously, as well as the operations that we’d like to allow on trusted values (note that for efficiency all operations could be inlined):

```
package Taint is
  type Trusted_Value is private;
  function Value (V : Trusted_Value) return Integer;
  function Trusted_1 return Trusted_Value;
  function Trusted_100 return Trusted_Value;
  function "+" (V, W : Trusted_Value)
    return Trusted_Value;
private
  type Trusted_Value is new Integer;
end Taint;
```

The new implementation is as expected:

```
package body Taint is
  function Value (V : Trusted_Value) return Integer is
  begin
    return Integer (V);
  end Value;
  function Trusted_1 return Trusted_Value is
  begin
    return 1;
  end Trusted_1;
  function Trusted_100 return Trusted_Value is
  begin
    return 100;
  end Trusted_100;
  function "+" (V, W : Trusted_Value)
    return Trusted_Value is
  begin
    return Trusted_Value (Integer (V) + Integer (W));
  end "+";
end Taint;
```

Of course, the client now needs to be adapted to this new interface:

```
with Taint; use Taint;
with Sensitive;
procedure Good is
  X : Trusted_Value := Trusted_100;
begin
  X := X + Trusted_1; --  Perform any computations on X
  Sensitive (X);
end Good;
```

That's it! No errors can result in tainted data being accidentally passed by the user where trusted data is expected, and future maintainers of the code won't be tempted to insert conversions when the compiler complains.

Gem #83: Type-Based Security 2: Validating the Input

Yannick Moy, AdaCore

Date: 5 April 2010

Abstract: Ada's strong type system makes it quite convenient to check at compilation time that certain security properties are verified, for example that a tainted value is not used where a trusted one is expected, or that data is properly validated before being used in a sensitive context (think of SQL injection attacks).

In the first Gem of this series of two, we discussed how to handle tainted data. In this Gem, we explain how to validate the input given to an SQL command.

Let's get started…

Input validation consists of checking a set of properties on the input which guarantee it is well-formed. This usually involves excluding a set of ill-formed inputs (black-list) or matching the input against an exhaustive set of well-formed patterns (white-list).

Here, we consider the task of validating an input for inclusion in an SQL command. This is a well-known defense against SQL injection attacks, where an attacker passes in a specially crafted string that is interpreted as a command rather than a plain string when executing the initial SQL command.

The basic idea is to define a new type SQL_Input derived from type String. Function Validate checks that the input is properly validated and fails if not. Function Valid_String returns the raw data inside a validated string, as follows:

```
package Inputs is
  type SQL_Input is new String;
  function Validate (Input : String) return SQL_Input;
  function Valid_String (Input : SQL_Input) return String;
end Inputs;
```

The implementation of Validate simply checks that the input string does not contain a dangerous character before returning it as an SQL_Input, while Valid_String is a simple type conversion:

```
with Ada.Strings.Fixed; use Ada.Strings.Fixed;
with Ada.Strings.Maps;  use Ada.Strings.Maps;
package body Inputs is
  Dangerous_Characters : constant Character_Set :=
                             To_Set ("""*^';&><</");
  function Validate (Input : String) return SQL_Input is
  begin
    if Index (Input, Dangerous_Characters) /= 0 then
      raise Constraint_Error
        with "Invalid input " & Input & " for an SQL query ";
    else
      return SQL_Input (Input);
    end if;
  end Validate;
  function Valid_String (Input : SQL_Input) return String is
  begin
    return String (Input);
  end Valid_String;
end Inputs;
```

Now, this does not prevent future uses of such type conversions in the program, whether malicious or unintended. To guard against such possibilities, we must make type SQL_Input private. To make sure we do not ourselves inadvertently convert an input string into a valid one in the implementation of package Inputs, we use this opportunity to make SQL_Input a discriminated record parameterized by the validation status.

```
with Ada.Strings.Unbounded; use Ada.Strings.Unbounded;
package Inputs is
  type SQL_Input (<>) is private;
  function Validate (Input : String) return SQL_Input;
  function Valid_String (Input : SQL_Input) return String;
  function Is_Valid (Input : SQL_Input) return Boolean;
private
  type SQL_Input (Validated : Boolean) is
```

```
   record
      case Validated is
         when True =>
            Valid_Input : Unbounded_String;
         when False =>
            Raw_Input   : Unbounded_String;
      end case;
   end record;
end Inputs;
```

Each time we access field Valid_Input, a discriminant check will be performed to ensure that the operand of type SQL_Input has been validated. Observe the use of Unbounded_String for the type of the input component, which is more convenient and flexible than using a constrained string.

Note in the implementation of Validate, that instead of raising an exception when the string cannot be validated, as in the first implementation, here we create corresponding validated or invalid input values based on the result of the check against dangerous characters. Also, an Is_Valid function has been added to allow clients to query validity of an SQL_Input value.

```
with Ada.Strings.Fixed; use Ada.Strings.Fixed;
with Ada.Strings.Maps;  use Ada.Strings.Maps;
package body Inputs is
   Dangerous_Characters : constant Character_Set :=
                                     To_Set (""""*^';&><</");
   function Validate (Input : String) return SQL_Input is
      Local_Input : constant Unbounded_String :=
To_Unbounded_String (Input);
   begin
      if Index (Input, Dangerous_Characters) /= 0 then
         return (Validated   => False,
                 Raw_Input   => Local_Input);
      else
         return (Validated   => True,
                 Valid_Input => Local_Input);
      end if;
   end Validate;
   function Valid_String (Input : SQL_Input) return String is
   begin
      return To_String (Input.Valid_Input);
   end Valid_String;
   function Is_Valid (Input : SQL_Input) return Boolean is
   begin
      return Input.Validated;
   end Is_Valid;
end Inputs;
```

That's it! As long as this interface is used, no errors can result in improper input being interpreted as a command, while ensuring that future maintainers of the code won't inadvertently be able to insert inappropriate conversions.

Of course, this minimal interface does not really provide anything other than the validation of the input. Simply having an Is_Valid function to tell whether a string is valid input data would seem to give you much the same functionality. However, you can now safely extend this package with additional capabilities, such as transformations on valid SQL inputs (for example, to optimize queries before sending them to the database), or to resolve queries faster using a local cache, and so forth. By using the private encapsulation, you are guaranteed that no client package will tamper with the validity of the SQL inputs you are manipulating.

Incidentally, the similar but distinct problem of input sanitization, where possibly invalid data is transformed into something that is known valid prior to use, can be handled in the same way.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o K.U. Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Peter Dencker
Steinäckerstr. 25
D-76275 Ettlingen-Spessartt
Germany
Email: dencker@web.de
URL: ada-deutschland.de

Ada-France

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. José Javier Gutiérrez
Ada-Spain
P.O.Box 50.403
28080-Madrid
Spain
Phone: +34-942-201-394
Fax: +34-942-201-402
Email: gutierjj@unican.es
URL: www.adaspain.org

Ada in Sweden

Ada-Sweden
attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada Switzerland

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: ada@white-elephant.ch
URL: www.ada-switzerland.ch