# ADA USER JOURNAL

Volume 30
Number 3
September 2009

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Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.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

This issue of the Ada User Journal publishes the Proceedings of the “Workshop on Vulnerabilities” which was co-located with the Ada-Europe 2009 conference, June 2009 in Brest, France. The workshop analyzed the Ada community’s answer to the effort, led by ISO/IEC JTC 1/SC 22/WG 23, to produce an ISO Technical Report on Programming Language Vulnerabilities (ISO/IEC PDTR 24772.2 – Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use). The goals of the workshop were to identify possible errors and omissions in the Technical Report with respect to the Ada language, and to eventually define the Ada annex to it.

The Proceedings start with a short report of the workshop discussion and results, by the workshop organizer, Joyce Tokar from Pyrrhus Software, USA. Afterwards, a paper by Jean-Pierre Rosen, from Adalog, France, points out the need to remove the programming language bias from the Technical Report, noting that its language independent part is prepared with C and C++ in mind. The second paper, by Steve Michell, from Maurya Software, Canada, addresses the Ada annex to the Technical Report, providing examples of the writing of the Ada section of some of the vulnerabilities. Afterwards, Tullio Vardanega, of the University of Padua, Italy, provides his view on the need for carefully consider language subsets, and support tools, in the framework of vulnerabilities mitigation.

The next paper of the workshop, by Alan Burns and Andy Wellings, from the University of York, UK, notes the importance of concurrency-related vulnerabilities, which were deferred for later treatment by WG 23. The final paper, by Rod Chapman, of Praxis High Integrity Systems, UK, presents both a list of potential benefits of enumerating vulnerabilities, and also a list of limitations and dangers of the same enumeration. It was indeed a successful workshop, and a group of people volunteered to continue the work on this topic, in order to prepare the Ada Annex of the Technical Report, and further meetings related to it are being prepared to be held at SIGAda 2009.

As for the other sections of the Journal, as usual the issue provides the News and Calendar information, by Marco Panunzio and Dirk Craeynest, the respective editors. The forthcoming events section provides details on the 2009 SIGAda conference, which will take place next November in the Tampa Bay area, Florida, USA, and on the 15th International Conference on Reliable Software Technologies – Ada-Europe 2010 that will take place June 2010 in Valencia, Spain.

The issue also publishes a set of Ada Gems, starting with the Ada 83/Ada 95 incompatibility of unconstrained arrays in generics, by Robert Dewar, of Ada Core; the use of pragma suppress, by Gary Dismukes of Ada Core; and two gems on the use of SPARK, by Yannick Moy, also of AdaCore.

*Luís Miguel Pinho*
*Porto*
*September 2009*
*Email: lmp@isep.ipp.pt*

# News

*Marco Panunzio*

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

## Contents

## Ada-related Organizations

### ARA — ACATS 3.0L

*From: Ada Information Clearinghouse*
*Date: Wed, 1 Jul 2009*
*Subject: Ada Conformity Assessment Test Suite*
*URL: http://www.adaic.com/whatsnew.html*

ACATS Modification List 3.0L and the associated test files have been posted.

[see also "ARA — ACATS 3.0J and 3.0K" in AUJ 30-2 (Jun 2009), p.69 —mp]

### Ada 2005 R2 Reference Manual

*From: Ada Information Clearinghouse*
*Date: Fri, 10 July 2009*
*Subject: Ada 2005 R2 Reference Manual*
*URL: http://www.adaic.com/standards/ada1z.html*

Draft 8 of Ada 2005 R2 was posted.

It includes some of the changes that are expected to be included Amendment 2 to Ada 95, including bounded container forms.

[see also "Ada 2005 R2 Reference Manual" in AUJ 30-2 (Jun 2008), p.69 —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 2010 — Call for Papers

*From: Dirk Craeynest <dirk@asgard.cs.kuleuven.be>*
*Date: Sun, 30 Aug 2009 21:33:56 +0200 CEST*
*Subject: CfP 15th Conf. Reliable Software Technologies, Ada-Europe 2010*
*Newsgroups: comp.lang.ada .fr.comp.lang.ada,comp.lang.misc*

-------------------------------------------------

CALL FOR PAPERS

15th International Conference on

Reliable Software Technologies - Ada-Europe 2010

14 - 18 June 2010, Valencia, Spain

http://www.ada-europe.org/ conference2010.html

Organized by Ada-Europe,

in cooperation with ACM SIGAda (approval pending)

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

-------------------------------------------------

Ada-Europe organizes annual international conferences since the early 80's. This is the 15th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain ('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04), York, UK ('05), Porto, Portugal ('06), Geneva, Switzerland ('07), Venice, Italy ('08), and Brest, France ('09).

General Information

------------------

The 15th International Conference on Reliable Software Technologies - Ada-Europe 2010 will take place in Valencia, Spain. Following its traditional 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.

Schedule

--------

16 November 2009: Submission of regular papers, tutorial and workshop proposals

11 January 2010: Submission of industrial presentation proposals

01 February 2010: Notification of acceptance to all authors

01 March 2010: Camera-ready version of regular papers required

10 May 2010: Industrial presentations, tutorial and workshop material required

14-18 June 2010: Conference

Topics

------

The conference has successfully established itself as an international forum for providers, practitioners and researchers into reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies. To gather experience on the latest periodic revision of the Ada language standard, contributions that present and discuss the potential of the revised language are especially welcome.

All prospective 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 Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- Software Architectures: 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 Technology.
- 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, Trust and Safety, Languages Vulnerabilities.
- Embedded Systems: Multicore Architectures, Architecture Modeling, HW/SW Co-Design, Reliability and Performance Analysis.
- Mainstream and Emerging Applications: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Energy, Games and Serious Games, etc.
- Experience Reports: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- Ada and Education: Where does Ada stand in the software engineering curriculum; how learning Ada serves the curriculum; what it takes to form a fluent Ada user; lessons learned on Education and Training Activities with bearing on any of the conference topics.

Call for Regular Papers

----------------------

Authors of regular papers which are to undergo peer review for acceptance are invited to submit original contributions. Paper submissions shall be in English, complete and not exceeding 14 LNCS-style pages in length. Authors should submit their work via the 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 1 March 2010. For format and style guidelines authors should refer to the following URL: http://www.springer.de/comp/lncs/authors.html.

Failure to comply and to register for the conference 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 may deliver value and insight, but do not fit the selection process for regular papers. Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation to the Conference Chair by 11 January 2010. The Industrial Program Committee will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it to the Conference Chair by 10 May 2010, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference.

Call for Tutorials

-----------------

Tutorials should address subjects that fall within the scope of the conference and may be proposed as either half- or full-day events.

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

Call for Workshops

-----------------

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

Call for Exhibitors

------------------

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

Grants for Students

------------------

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

Organizing Committee

-------------------

Conference Chair

Jorge Real, Universidad Politécnica de Valencia, Spain

jorge@disca.upv.es

Program Co-Chairs

Jorge Real, Universidad Politécnica de Valencia, Spain

jorge@disca.upv.es

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

Ahlan Marriott, White Elephant GmbH, Switzerland

Ada@white-elephant.ch

Industrial Chair

Erhard Plödereder, University of Stuttgart, Germany

ploedere@informatik.uni-stuttgart.de

Publicity Chair

Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium

Dirk.Craeynest@cs.kuleuven.be

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

Roderick Chapman, Praxis High Integrity Systems, 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

Raymond Devillers, Université Libre de Bruxelles, Belgium

Franco Gasperoni, AdaCore, France

Michael González Harbour, Universidad de Cantabria, Spain

José Javier Gutiérrez, Universidad de Cantabria, Spain

Andrew Hately, Eurocontrol CRDS, Hungary

Peter Hermann, Universität Stuttgart, Germany

Jérôme Hugues, ISAE Toulouse, France

Hubert Keller, Institut für Angewandte Informatik, Germany

Albert Llemosí, Universitat de les Illes Balears, Spain

Kristina Lundqvist, Mälardalen University, Sweden & MIT, USA

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

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, University of Newcastle upon Tyne, UK

Jean-Pierre Rosen, Adalog, France

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

Santiago Urueña, Grupo de Mecánicade Vuelo, Spain

Tullio Vardanega, Università di Padova, Italy

Francois Vernadat, LAAS-CNRS & INSA Toulouse, France

Daniel Wengelin, Saab, Sweden

Andy Wellings, University of York, UK

Jürgen Winkler, Friedrich-Schiller Universität, Germany

Luigi Zaffalon, University of Applied Sciences, Switzerland

Industrial Committee
--------------------

Guillem Bernat, Rapita Systems, UK

Roderick Chapman, Praxis High Integrity Systems, UK

Dirk Craeynest, Aubay Belgium & K.U.Leuven, Belgium

Pierre Dissaux, Ellidiss Technologies, France

Franco Gasperoni, AdaCore, France

Hubert Keller, Forschungszentrum Karlsruhe GmbH, Germany

Ismael Lafoz, EADS CASA, Spain

Ahlan Marriott, White-Elephant GmbH, Switzerland

Erhard Plödereder, Universität Stuttgart, Germany

José Simó, Universidad Politécnica de Valencia, Spain

Alok Srivastava, Northrop Grumman, USA

Rei Stråhle, Saab Systems, Sweden

[…]

## Call for SIGAda Award Nominations

*From: Ricky E. Sward <ricky.sward@msn.com>*
*Date: Fri, 28 Aug 2009 09:38:36 -0700 PDT*
*Subject: Call for SIGAda Award Nominations*
*Newsgroups: comp.lang.ada*

Dear Members of the Ada Community:

As the new Chair of SIGAda, I'm sending you this call for nominations for the SIGAda Outstanding Ada Community Contribution Award and the ACM SIGAda Distinguished Service Award. Nominations are due on September 28th to SIGAda-Award@listserv.acm.org.

On Thursday, Nov 5th, these SIGAda Awards will be presented in a special morning plenary session at the SIGAda 2009 conference in Tampa, Florida (See http://www.sigada.org/conf/sigada2009/).

The ACM SIGAda Awards recognize individuals and organizations who have made outstanding contributions to the Ada community and to SIGAda.

The two categories of awards are:

(1) Outstanding Ada Community Contribution Award

-- For broad, lasting contributions to Ada technology & usage.

(2) ACM SIGAda Distinguished Service Award

-- For exceptional contributions to SIGAda activities & products.

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

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

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

The nomination form is available on the SIGAda website at http://www.acm.org/sigada/exec/awards/awards.html. You need to visit this website to see past award winners' names, so you don't nominate someone who has already won an award in a category. A picture of the statuette used for the award is also on this site. Submit your nomination as an e-mail or e-mail attachment to SIGAda-AWARD@listserv.acm.org.

The ACM SIGAda Awards Committee, composed 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 September 28th.

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

Thank you,

Ricky E. Sward

Chair, ACM SIGAda

## Ada Semantic Interface Specification (ASIS)

### ASIS and the Ada 2005 reference manual

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Fri, 5 Jun 2009 10:01:54 -0700 PDT*
*Subject: ASIS : questions about the ASIS status against the Ada reference*
*Newsgroups: comp.lang.ada*

Hello ASIS fans,

I'm interested in ASIS as it is an abstraction for representing Ada sources. One may set up its own abstract representation, but as such a thing already exist, it is better to use the existing one. Better, because such an interface is a rather big stuff, and it is better to reuse to avoid errors and get the benefits of long time worked issues.

But I have some doubt about ASIS. Well, at least, one.

Looking at the ASIS package specifications and even more at the ASIS issues at ada-auth, I began to feel that a lot of efforts are still running at making ASIS able to fully represent all possible Ada 2005 constructions.

From here, comes these fundamental questions :

- Is the ability of ASIS to represent all Ada 2005 constructs formally provable?

- Can ASIS be viewed as another expression of a part of the Ada 2005 reference in a special formalism?

While I know ASIS is mainly useful for code analysis, audits, statistics, etc, I though about another question which may give an answer to these two latter ones: formally speaking (although perhaps not practicable), is it theorically possible to imagine a compiler built around ASIS ? If the answer is yes, then the answer to the two previous questions would be yes as well.

*From: Stephen Leake <stephen_leake@stephe-leake.org>*
*Date: Sat, 06 Jun 2009 02:25:45 -0400*
*Subject: Re: ASIS : questions about the ASIS status against the Ada reference*
*Newsgroups: comp.lang.ada*

> […]

> - Is the ability of ASIS to represent all Ada 2005 constructs formally provable ?

I doubt it. But why do you want to know?

> - Can ASIS be viewed as another expression of a part of the Ada 2005 reference in a special formalism?

Probably.

> While I know ASIS is mainly useful for code analysis, audits, statistics, etc, I though about another question which may give an answer to these two latter ones: formally speaking (although perhaps not practicable), is it theorically possible to imagine a compiler built around ASIS?

I don't see how; ASIS presumes a compiler that generates the information ASIS uses. Why build another compiler on top of that?

*From: Pascal Obry <pascal@obry.net>*
*Date: Sat, 06 Jun 2009 11:24:54 +0200*
*Subject: Re: ASIS : questions about the ASIS status against the Ada reference*
*Newsgroups: comp.lang.ada*

> […]

> formally speaking (although perhaps not practicable), is it theorically possible to imagine a compiler builded around ASIS?

I would say yes. The internal representation of the Ada code could be based on ASIS. Yet I don't think there is a single Ada compiler that has followed this path.

*From: Pascal Obry <pascal@obry.net>*
*Date: Fri, 05 Jun 2009 19:23:28 +0200*
*Subject: Re: ASIS : questions about the ASIS status against the Ada reference*
*Newsgroups: comp.lang.ada*

[…]

> While I know ASIS is mainly useful for code analysis, audits, statistics, etc,

Don't forget code generation. For example AWS generates WSDL documents out of Ada specs.

## Ada and Education

### Praxis — SPARK Training September 2009

*From: Praxis HIS*
*Subject: SPARK Training*
*Date: September, 2009*
*URL: http://www.praxis-his.com/sparkada/training.asp*

SPARK Training - Overview

We run six courses in SPARK, details of which are below.

The schedule for public courses is shown below. Exclusive courses for clients, either at our offices or on-site, are also available – please contact us for details.

Course 1: Two Day Overview

A 2-day day "extended tutorial" for managers and engineers that presents the principles and practice of high assurance software engineering with SPARK.

The course explains the rationale of SPARK, outlines the language and the principles of static code analysis, and presents the role of SPARK in systematic program development.

The course also covers the design of the SPARK language and the various types of analysis and verification that can be performed.

The second day of the course concentrates on practical issues, such as how SPARK matches contemporary standards for high assurance software and software processes such as CMM and PSP/TSP. Finally, the issues (and problems) of adopting SPARK will be considered, followed by case-studies of SPARK usage in the aerospace, rail and security domains.

This course includes some pencil-and-paper exercises, but does not involve computer-based practical sessions or SPARK programming. Students requiring a thorough understanding of the practical use of SPARK are referred to the longer "Software Engineering with SPARK" course.

Course 2: Software Engineering with SPARK

A 4-day course for managers, regulators and engineers, which presents the principles of the development of high integrity software, and the related certification requirements.

It then explains the rationale of SPARK, outlines the language and the principles of static code analysis, and presents the role of the SPARK Examiner in systematic program development. The course also covers fundamental SPARK design issues, such as appropriate use of packages such as abstract machines and data types, as well as the use of SPARK refinement, system interfaces, library mechanisms, etc. Some of the more advanced facilities of the SPARK Examiner, for run-time error checking for example, are presented.

Course 3: Advanced SPARK Program Design and Verification

A course for engineers who have already attended the "Software Engineering with SPARK" course or are experienced SPARK users. This course covers the advanced use of SPARK, particularly in the context of proof of exception freedom and code correctness.

Attendees are taught to understand the relationship between SPARK source code and the verification conditions generated for proof, leading to an understanding of the impact of good SPARK design principles on code verification.

Advanced facilities of the SPARK Examiner are presented, and tuition in planning, conducting and managing the verification activities is supplemented by the use of the SPARK proof tools, particularly the Simplifier.

The course has a strongly practical flavour, interweaving guidance and lecture material with topical tutorial sessions which reinforce the lecture material via relevant examples. Each tutorial session commences with a step-by-step example which provides detailed guidance, followed by additional exercises which can be tried in the tutorial sessions or used after the course to gain additional practical experience. Note that this course does not cover the RavenSPARK language profile or the use of the Proof Checker tool.

Course 4: Concurrent Software Design with RavenSPARK

The Ada95 Ravenscar profile defines a subset of the Ada95 tasking facilities that are appropriate for the construction of high assurance software. This one-day course introduces the Ravenscar profile and how it has been included in the core SPARK language. The course will cover the additional annotations in SPARK that are used to describe packages that contain tasks and protected objects and the additional analyses implemented by the Examiner to eliminate the potential for defects in Ravenscar programs.

Delegates for this course should have already attended the introductory "Software Engineering with SPARK" course, or should be experienced SPARK users. This course may be taken as a one-day stand-alone module, or may directly follow a "Software Engineering with SPARK" course.

Course 5: Introduction to the Proof Checker

This course introduces the SPARK Proof Checker, and how it can augment the abilities of the Examiner and Simplifier in the verification of SPARK programs. Attendees will cover the basics of using the Checker's interactive proof engine to prove VCs that remain after simplification.

The course also covers the use of the Proof Checker to establish the validity of user-defined proof rules. Delegates for this course should have already attended the “Advanced SPARK Program Design and Verification” course.

Course 6: UML to SPARK

This course covers the rationale for integrating SPARK with UML, and the generation of SPARK from UML. The majority of the course consists of a practical session, where delegates will produce SPARK from a partially completed UML model.

Delegates for this course should have already attended the introductory "Software Engineering with SPARK" course, or should be experienced SPARK users.

No knowledge of UML or experience of using UML tools is assumed.

This course may be taken as a one-day stand-alone module, or may directly follow a "Software Engineering with SPARK" course.

Public Course Dates for 2009 - UK

Course 1 – "Two Day Overview"

TBD - come back soon for future course dates.

Course 2 – "Software Engineering with SPARK"

7th - 10th September 2009, Bath, UK.

Course 3 – "Advanced SPARK Program Design and Verification"

22nd - 24th September 2009, Bath, UK.

Course 4 – "Concurrent Software Design with RavenSPARK"

TBD - come back soon for future course dates.

Course 5 – "Introduction to the Proof Checker"

25th September 2009, Bath, UK.

Course 6– "UML to SPARK"

TBD - come back soon for future course dates.

On-site training in the UK and around the world

Praxis High Integrity Systems can run training courses at a customer's facilities as required. Training worldwide is available from our partner company AdaCore. Training in North America is also available from our partner company Pyrrhus Software.

Enquiries and Reservations

For enquiries and reservations for the course, please contact us.

[see also "Praxis — SPARK Training March 2009" in AUJ 29.4 (Dec 2008), p.229 —mp]

## Ada-related Resources

### Ada 95 Quality and Style

*From: Rick Duley <rickduley@gmail.com>*
*Date: Sat, 20 Jun 2009 22:21:33 -0700 PDT*
*Subject: Ada Quality and Style*
*Newsgroups: comp.lang.ada*

I have my trusty "Ada95 Quality and Style: Guidelines for Professional Programmers". I cannot find a new version. Is there one? If so, where can I obtain a copy?

*From: John McCormick <mccormick@cs.uni.edu>*
*Date: Sun, 21 Jun 2009 04:48:46 -0700 PDT*
*Subject: Re: Ada Quality and Style*
*Newsgroups: comp.lang.ada*

[…]

There was a discussion at the SIGAda 2008 meeting in Portland about updating the style guide. Geoff Smith suggested that the process be open to the Ada community. He created a wikibook using the Ada 95 version as a starting point. I'm not sure how much work has been done on it. Certainly there is still some work to be done to add formatting markup, but the entire document is there and ready for people to add guidance for Ada 2005 additions.

http://en.wikibooks.org/wiki/Ada_Style_Guide

### Ada implementation of FFT

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Fri, 17 Jul 2009 09:27:21 +0200*
*Subject: Re: Fourier*
*Newsgroups: comp.lang.ada*

> Does someone know an Ada implementation of some Fourier Transform? Or where I can find it?

One implementation of FFT is mentioned at AdaIC:

http://www.adaic.org/links/libs.html

(the last in the list)

*From: tmoran@acm.org*
*Date: Fri, 17 Jul 2009 07:37:25 +0000 UTC*
*Subject: Re: Fourier*
*Newsgroups: comp.lang.ada*

> One implementation of FFT is mentioned at AdaIC:

There are many around - do you want an FFT for general (composite) N, or just powers of two. Do you want to link in highly optimized routines for particular sizes, or just something simple to use.

*From: Pablo Vieira Rego <pablittto@gmail.com>*
*Date: Fri, 17 Jul 2009 04:49:19 -0700 PDT*
*Subject: Re: Fourier*
*Newsgroups: comp.lang.ada*

[…]

I look for general N, simple to use, basic to analyse data energy coherence.

*From: tmoran@acm.org*
*Subject: Re: Fourier*
*Date: Fri, 17 Jul 2009 23:53:06 +0000 UTC*
*Newsgroups: comp.lang.ada*

[…]

I use a Glassman algorithm version. It's fast for highly composite N (eg, powers of 2), but slow for prime sizes. For FFTing images, that could be very slow indeed, so I use a companion utility to find the operation count for each K in N .. N+20 to see if a small extension of the data will substantially speed things up.

*From: John B. Matthews <jmatthews@wright.edu>*
*Date: Fri, 17 Jul 2009 10:27:45 -0400*
*Subject: Re: Fourier*
*Newsgroups: comp.lang.ada*

[…]

Google says: <http://fftwada.sourceforge.net/>

## Ada-related Tools

### Simple Components 3.4, 3.5 and 3.6

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sun, 14 Jun 2009 16:23:31 +0200*
*Subject: ANN: Simple Components for Ada v3.4*
*Newsgroups: comp.lang.ada*

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

The current version provides implementations of smart pointers, 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; strings editing and tables management.

Changes to the previous version:

1. The Test_Sequencer procedure made Ada 95 conform;

2. The package Parsers.Multiline_Source.Latin1_Text_IO was added to support Latin-1 text I/O;

3. The package Parsers.Multiline_Source.Wide_Text_IO was added to support wide text I/O;

4. The generic procedure Parsers.Multiline_Source.Get_UTF8_Text was added for matching texts in UTF-8 sources;

5. Parsers.Generic_Lexer.Ada_2005_Blanks and Parsers.Generic_Source.Get_Ada_2005_Blank were added to support Ada 2005 in UTF-8 encoding;

6. Task-safe implementation of reference counting was added;

7. Task-safe implementation of persistent storage was added;

8. Tracing implementation for GNAT Ada for reference counting object was added;

9. Get_Class abstract operation was added to the persistent storage interface;

10. Generic_Chebyshev_Polynomials was added to sum Chebyshev series of the first kind;

11. Gamma function approximation for Float;

12. Parsers.Generic_Source.Text_IO expands tabs;

13. Procedures Get_Location and Skip were added to Parsers.Multiline_Source;

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Thu, 9 Jul 2009 20:23:56 +0200*
*Subject: ANN: Simple Components for Ada v3.5*
*Newsgroups: comp.lang.ada*

[…]

Changes to the previous version:

Minor bug fixes in example test_ada_parser.adb fixed and in Parsers.Generic_Source.Get_Cpp_Blank.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Fri, 7 Aug 2009 16:33:56 +0200*
*Subject: ANN: Simple Components for Ada v3.6*
*Newsgroups: comp.lang.ada*

[…]

Changes to the previous version:

1. Packages for text output of the persistent storage index were added;

2. Is_Directory was added to Persistent.Directory;

3. Handles and sets of backward links added.

[see also "Simple Components 3.3" in AUJ 30‑1 (Mar 2009), p.7 —mp]

### Fuzzy Sets for Ada v5.4

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Sun, 14 Jun 2009 16:29:49 +0200*
*Subject: ANN: Fuzzy Sets for Ada v5.4*
*Newsgroups: comp.lang.ada*

The current version includes distributions of string edit, interval arithmetic and simple components packages. It provides implementations of:

1. Confidence factors with the operations not, and, or, xor, +, *;

2. Classical fuzzy sets with the set-theoretic operations and the operations of the possibility theory;

3. Intuitionistic fuzzy sets with the operations on them;

4. Fuzzy logic based on the intuitionistic fuzzy sets and the possibility theory;

5. Fuzzy numbers both integer and floating-point ones with conventional arithmetical operations;

6. Dimensioned fuzzy numbers;

7. Fuzzy linguistic variables and sets of linguistic variables with operations on them;

8. Dimensioned fuzzy linguistic variables and sets;

9. String-oriented I/O is supported;

10. GUI interface based on GTK+ (The GIMP Toolkit) with fuzzy set editors, truth values widgets and renderers, linguistic variables sets editors.

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

The current version works with GNAT GPL 2009 and when GUI interface is used with GtkAda 2.14.

### GNAT GPL 2009 for 32-bit Mac OS X

*From: Simon J. Wright <simon.j.wright@mac.com>*
*Date: Fri, 12 Jun 2009 14:49:57 -0700 PDT*
*Subject: GNAT GPL 2009 for 32-bit Mac OS X*
*Newsgroups: comp.lang.ada*

AdaCore's GNAT GPL 2009 is built for the x86_64 architecture only.

Some Apple libraries (particularly Tk) come in 32-bit only.

This new build, on Sourceforge in the GnuAda project, is 32-bit only:

https://sourceforge.net/project/showfiles.php?group_id=12974&package_id=258771

### On the Open Ravenscar Kernel

*From: Emilio Salazar <emilio.salazar@gmail.com>*
*Date: Wed, 29 Jul 2009 06:49:35 -0700 PDT*
*Subject: Re: Open Ravenscar Real-Time Kernel*
*Newsgroups: comp.lang.ada*

> Does anyone know the status of the Open Ravenscar Real-Time Kernel (ORK+) project? www.openravenscar.org is dead.

[…]

Please, try with this link

http://polaris.dit.upm.es/~str/ork/index.html

[…]

### GLOBE_3D

*From: GLOBE 3D Project Webpage*
*Date: Tue, 01 Sep 2009*
*Subject: GLOBE_3D - Ada and GL 3D Engine*
*URL: http://globe3d.sourceforge.net/*

GLOBE_3D stands for GL Object Based Engine for 3D.

GL stands for Graphics Library, created by SGI. SGI stands for Silicon Graphics, Inc. .

Short description: GLOBE_3D is a free, open-source, real-time 3D Engine written in Ada, based on OpenGL.

Features:

- unconditionally portable sources (one set of sources for all platforms)
- real-time rendering; fast with a 3D hardware-accelerated graphics card
- full eye movements and rotations
- displays combinations of colours, materials, textures
- transparency
- multiple area rendering with the portal technique, e.g. for inner scenes
- collision detection [NEW]
- binary space partition (BSP)
- input-output of 3D objects or groups of objects linked to each other by portals
- easy management of resources like textures (.bmp, .tga), BSP trees and objects stored in .zip files
- screenshots (.bmp) and video captures (.avi)
- multi-view support
- vectorized geometry support

Goodies:

- randomly extruded surface generator

Tools:

- VRML virtual world compiler, through the wrl2ada translator
- GMax / 3D Studio Max scene exporter & compiler, through the max2ada translator
- Compilation of game maps or levels from the Doom 3, Quake 4 or GTK Radiant level editors through the d3a (to Ada) translator or the d3g (to .g3d) tool.

Download:

Download the archive at the SourceForge project page.

It contains:

- a ready-to-run demo built for Windows and Linux
- the tools mentioned above
- fresh bindings to GL / OpenGL, GLU and FreeGLUT
- a single, standalone and complete set of Ada sources, successfully built on the following system / cpu / compiler combinations:

OS;CPU;Compiler

MS Windows 95,98,NT,2K,XP;Intel x86 (32 bit);GNU - GNAT

MS Windows 95,98,NT,2K,XP;Intel x86 (32 bit);Aonix - ObjectAda

Linux;Intel x86 (32 bit);GNU - GNAT

Mac OS X;PowerPC (64 bit);GNU - GNAT

OpenBSD;(one of several);GNU - GNAT

For any other system, GLOBE_3D should work provided that the system supports (Open)GL and GLU.

Availability of FreeGLUT or OpenGLUT is needed for the demo.

I welcome your scripts, makefiles etc.

[verbatim from the home page of the project —mp]

## GWenerator 0.97 and 0.975

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Mon, 01 Jun 2009 19:49:29 +0200*
*Subject: Ann: GWenerator 0.97*
*Newsgroups: comp.lang.ada*

[…]

A new version of GWenerator is out! Latest changes:

0.97 (01-Jun-2009):

- a test application with all of a resource's dialogs is now generated, too, on request
- width/height settings in dialogs now refer to client area (bug fix)
- CONTROLs of "Button" and "Edit" classes... understood as alternative to 'typed' button/editbox controls
- many more resources files from MS Visual Studio, Borland Resource Workshop, ResEdit and others are now translated

Visit: http://sf.net/projects/gnavi/

With GWenerator you can design Graphical User Interfaces with existing software like Visual Studio or the free ResEdit ( http://resedit.net ), and program Windows applications in Ada using the GWindows object-oriented library.

GWenerator produces Ada sources corresponding to dialogs and menus, as a background task. The command-line equivalent, rc2gw, does the same job on request.

Of course, GWenerator's own GUI is itself produced this way - kind of a self-demo.

The archive contains some other examples and numerous stress-tests downloaded from Internet.

But the better is to play around and send feedbacks or ask questions on the GNAVI mailing list…

[…]

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Wed, 8 Jul 2009 13:06:44 -0700 PDT*
*Subject: Ann: GWenerator 0.975*
*Newsgroups: comp.lang.ada*

[…]

A new version of GWenerator is out! Latest changes:

0.975 (05-Jul-2009):

- new Initialize_controls option: initialize some controls with fake contents, for test/debug; analogous to Ada's Normalize_Scalars pragma
- better support for: progress bars, tree views, list views, static borders, some special window styles

Visit: http://sf.net/projects/gnavi/

With GWenerator you can design Graphical User Interfaces with existing software like Visual Studio or the free ResEdit

( http://resedit.net ), and program Windows applications in Ada using the GWindows object-oriented library.

GWenerator produces Ada sources corresponding to dialogs and menus, as a background task. The command-line equivalent, rc2gw, does the same job on request. On request, GWenerator produces a test application with all dialogs.

Unlike some other GUI libraries, GWindows is Windows-only, but a built application can hold in a single .exe (no need to provide any run-time framework, toolkit, dll's and worry about version conflicts), which make deployment very easy in a Windows-centric environment.

The archive contains numerous examples and stress-tests downloaded from Internet.

[see also "GWenerator" in AUJ 30.1 (Mar 2009), p.8 —mp]

## Visual Ada Developer 7.2

*From: Leonid Dulman <leonid_dulman@yahoo.co.uk>*
*Date: Tue, 7 Jul 2009 11:45:11 +0300*
*Subject: announce : Visual Ada Developer VAD 7.2*
*Newsgroups: comp.lang.ada*

Visual Ada Developer (VAD) 7.2 is now available

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 7.2 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/9x,Unix(Linux), Mac and OS/2.

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 7.2 was tested in Windows 32bit and 64bit and Linux x86-64 Fedora 11, Kubuntu 9.04

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/

W A R N I N G ! VAD 7.2 has two realization for tcl/tk 8.5.x and tcl/tk 8.6.x , you need to install and test tcl/tk at 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 choice needed version at link time. (I recommend to work with 8.6)

[…]

## GTKAda Contributions v2.4

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Fri, 7 Aug 2009 16:11:23 +0200*
*Subject: ANN: GtkAda Contributions v2.4 with GtkSourceView support*
*Newsgroups: comp.lang.ada*

This library is proposed as a contribution to GtkAda, an Ada bindings to GTK+. It deals with the following issues:

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

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

Changes to the previous version:

1. Support for translation of stack traceback into symbolic form by pasting from the clipboard;

[see also "GTKAda Contributions" in AUJ 29‑3 (Sep 2008), p.152 —mp]

## Status of wxAda

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Tue, 11 Aug 2009 20:20:14 -0700 PDT*
*Subject: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

I was thinking about wxWidgets to target Mac OS, because it ties to the underlying platform (I do not want something which would seem not to have been made for Mac, especially on Mac) and because also it will allow me to test on Windows (as I do not own a Mac)… well, to be exhaustive, the licensing terms are appealing too.

So I've ended to learn about the Ada binding for wxWidget, that is, wxAda, but all links to this either brings me to Tigris or SourceForge, which both do not contain any file at all for this project, as you can check yourself:

http://wxada.tigris.org/servlets/ProjectDocumentList

(zero source files in SVN repository)

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

(no source archives at all)

[…]

Do someone know how this adventure turns out ?

[…]

*From: Luke <Lucretia9000@yahoo.co.uk>*
*Date: Wed, 12 Aug 2009 06:01:24 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

While I mention that it is dead on the tigris page, I have recently started looking at it again from a slightly different angle. But currently, there is no usable version.

[…]

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Thu, 13 Aug 2009 03:03:10 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

What is the new way to bind to wxWidgets you are thinking about ?

I guess its big work, as wxWidgets (previously wxWindows) is all in C++, and interfacing C++ with Ada, typically requires an intermediate binding in C… a tedious task.

If I can talk about my specific concerns: if I cannot use wxAda, I will simply do as for the Windows API, creating a binding step by step, as comes the needs (the standard Ada Windows API binding is not like what I was looking for).

As I said, I was interested in wxWidget rather than others, because of its license terms and because it is an interface to the underlying OS, not an emulation which creates all its own widgets (I've seen screenshots of GTK on Mac, it look like GNOME on Linux, and Qt does an emulation too, although it lies better).

To go further, Mac OS seems to like wxWidget, as it seems there are wxWidgets headers in the standard Mac OS development environment (not GTK nor Qt) beside of the system and system framework headers. So wxWidgets seems a good choice.

Back to the C++ implementation of wxWidgets: the most recommended API for Mac OS is Cocoa, which is Objective-C, which can be interfaced as simply as C (if I believe to the literature about it).

So one moment, I thought about simply writing an Ada binding to Cocoa, but as I do not own a Mac, I could not run any test at all (this will be frightening).

If there was a Cocoa for Windows, this would be all fine, I could run tests on Windows.. but there is not.

*From: Luke <Lucretia9000@yahoo.co.uk>*
*Date: Thu, 13 Aug 2009 04:19:07 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

> What is the new way to bind to wxWidgets you are thinking about ?

Well, to try to automate it using python scripts (if this doesn't, work another attempt with SWIG). Rather than define all tagged types as non-limited types, they will be limited and they will have to be allocated dynamically, this is how other bindings to C++ UI toolkits work and really is the only way due to C++ semantics.

> I guess its big work, as wxWidgets (previously wxWindows) is all in C++, and interfacing C++ with Ada, typically requires an intermediate binding in C... a tedious task.

Yeah, bit job due to the size of wxWidgets, although I have no intention of supporting the entire lib, there's stuff wxAda just doesn't require.

> As I said, I was interested in wxWidget rather than others, because of its license terms […]

Yup, the license and also the ease of programming with wxWidgets is what got me started with it in the first place.

I think at the moment your options are this:

1) Use AdaGtk on all platforms for now.

2) Develop your UI code in whatever language for whatever platform, e.g. C++ for Win32, Obj-C for Apple and then create the app code in Ada providing some sort of interface between the 2.

*From: Luke <Lucretia9000@yahoo.co.uk>*
*Date: Thu, 13 Aug 2009 04:20:01 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

> Is some one interested in sponsoring an Ada to wxWidgets binding ?

Not that I know of and it is something I have considered, but it would have to be rather special as I have a full time job and I'm not in a position to give it up yet.

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Thu, 13 Aug 2009 04:52:52 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

I've got full time, as I do not have a job, so if someone is interested, I will be there. I could follow your recommendations for the implementation if you wish.

[…]

*From: Micronian Coder <micronian2@gmail.com>*
*Date: Thu, 13 Aug 2009 10:03:53 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

There is a library called Agar that is written in C. I have never used it, but I found it interesting that someone added a thin Ada binding to it. Currently, the Ada binding depends on AdaSDL since Agar depends on SDL and OpenGL. However, version 1.4 of Agar is supposed to allow for more variety of graphics drivers. In addition, Agar is available for these platforms: FreeBSD, IRIX, Linux, MacOS Classic, MacOS X, NetBSD, OpenBSD and Windows. It even mentions Game Cube support for anyone who is interested in that.

From the website libagar.org :

Agar is a modern open-source, cross-platform toolkit for graphical applications implemented in C, C++ and Ada (with bindings to other languages in development). Designed for ease of integration, it follows the philosophy of building the GUI around the application and not the other way around. Unlike most other GUI toolkits, Agar takes maximum advantage of hardware graphics acceleration when it is available via OpenGL, but it also supports traditional framebuffer interfaces such as SDL direct video. The Agar API is entirely thread-safe when Agar is compiled with optional threads support.

The distribution includes two libraries: Agar-GUI implements the base Agar GUI system and a comprehensive set of standard widgets. Agar-GUI is object-oriented and relies heavily on inheritance, virtual functions and virtual filesystems. This functionality is implemented by the GUI-independent Agar-Core library, which also includes various utility and portability interfaces.

The Agar distribution also includes some more specialized libraries aimed at specific applications, such as Agar-MATH, Agar-RG, Agar-VG and Agar-DEV. Some of our other toolkits which extend (and rely on) Agar include FreeSG, Edacious and cadtools.

Agar is free software. Its source code is freely usable and re-usable by everyone under a BSD license, which allows use in commercial applications free of charge. Agar is stable, well-maintained and has been growing organically since early 2002. The Agar project is sponsored by Csoft.net: Security conscious, high-availability Unix hosting on redundant server arrays.

[…]

*From: Luke <Lucretia9000@yahoo.co.uk>*
*Date: Thu, 13 Aug 2009 10:44:48 -0700 PDT*
*Subject: Re: wxAda : is it vanished ?*
*Newsgroups: comp.lang.ada*

[…]

I'm aware of it and know the person who did the bindings. But, as the OP pointed out, he is actually wanting per platform-native widgets.

## QtAda 3.0 and 2.2

*From: Vadim Godunko <vgodunko@gmail.com>*
*Date: Sun, 7 Jun 2009 12:07:40 -0700 PDT*
*Subject: Announce: QtAda 3.0 & QtAda 2.2*
*Newsgroups: comp.lang.ada*

We are pleased to announce the immediate availability of the QtAda 3.0.0 and QtAda 2.2.0. You can download multi platform source code package or Microsoft Windows binary package from the our download page:

http://www.qtada.com/en/download.html

QtAda is an Ada2005 language bindings to the Qt libraries and a set of useful tools. QtAda allows easily to create cross-platform powerful graphical user interface completely on Ada 2005. QtAda applications will work on most popular platforms -- Microsoft Windows, Mac OS X, Linux/Unix -- without any changes and platform specific code. QtAda allows to use all power of visual GUI development with Qt Designer.

New in QtAda 2.2.0:

- GNAT Project Files are installed for all examples
- improved support for QTextFormat and subclasses
- improved support for QFileDialog
- many bugfixes in QtSql bindings

New in QtAda 3.0.0:

- bindings for all QObject and QGraphicsItem classes and subclasses was reimplemented completely
- subclassing of QObject and QGraphicsItem classes and subclasses was simplified significantly

New on QtAda site:

- documentation for QtAda 3.0 and QtAda 2.2 is available on-line for now

[see also "QtAda 2.1.0" in AUJ 30-1 (Mar 2009), p.9 —mp]

## QtAda 2.1.1 for Qt 4.5.2

*From: Leonid Dulman <leonid_dulman@yahoo.co.uk>*
*Date: Thu, 16 Jul 2009 14:00:33 +0300*
*Subject: Ann : QtAda version 2.1.1 rebuild for Qt 4.5.2*
*Newsgroups: comp.lang.ada*

QtAda is an Ada-95(05) interface to Qt4 graphics library Qt version 4.5.2 open source and qt4c.dll(libqt4c.dll) built with Microsoft Visual Studio 2005 and MINGW GCC compiler in Windows and (libqt4c.so) gcc 4.3.4 in Linux.

Package tested with GNAT GPL 2009 Ada compiler in Windows 32bit and 64bit and Linux x86-64 Fedora 11 and Kubuntu 9.04 QtAda for Windows and Linux (Unix) is available from http://users1.jabry.com/adastudio/index.html

## TclAdaShell 20090611

*From: Simon J. Wright <simon.j.wright@mac.com>*
*Date: Fri, 12 Jun 2009 14:58:52 -0700 PDT*
*Subject: tcladashell-20090611*
*Newsgroups: comp.lang.ada*

This new release of TclAdaShell is uploaded to

https://sourceforge.net/projects/tcladashell/

It's a minor release: you no longer need gprbuild (which is a fine utility, but not everyone has it yet).

If you are on an Intel Mac, you won't be able to use AdaCore's GNAT GPL 2009, which is 64-bit only, whereas the Tk framework is 32-bit only; there's a 32-bit GNAT GPL 2009 at

https://sourceforge.net/project/showfiles.php?group_id=12974&package_id=258771

[see also "TASH 20090207" in AUJ 30-1 (Mar 2009), p.9 —mp]

## CairoAda 1.8

*From: Damien Carbonne <damien.carbonne@free.fr>*
*Date: Sat, 01 Aug 2009 11:59:47 +0200*
*Subject: ANN: CairoAda 1.8 update*
*Newsgroups: comp.lang.ada*

A new version of CairoAda is available.

http://sourceforge.net/projects/cairoada

Main changes are related to:

- Documentation
- More style checks
- User font API that was corrected
- Improvement in GNAT project files (checked on Linux and Windows)
- Support of GNAT GPL 2009
- Corrections in RSVG and addition of missing functions (RSVG 2.26).

Notes:

1) RSVG / Gdk.Cairo work with GtkAda provided with GNAT GPL 2009, but may need changes to work with latest version of GtkAda found on svn repository (there was a change in Gdk_Pixbuf).

2) Latest version of RSVG is needed. On Windows, no ready to use binary version of librsvg 2.26 seems available yet (I didn't find one and built one myself).

3) Support for FreeType should arrive soon. A binding has been started but has not been uploaded yet.

[…]

[see also "CairoAda 1.8" in AUJ 29-4 (Dec 2008), p.232 —mp]

## WinPCap/LibPCap Wrappers

*From: John McCabe*
*Date: Wed, 26 Aug 2009 16:43:16 +0100*
*Subject: WinPCap/LibPCap Wrappers?*
*Newsgroups: comp.lang.ada*

[…]

Sorry to have to ask this here, but a web search was a bit fruitless!

Does anyone know of an Ada wrapper to the WinPCap and/or libpcap libraries?

I'd like to be able to read in files created by Wireshark and manipulate the data in a sensible way. I guess I could do it with C++, but I'd much rather use Ada :-)

[…]

*From: Björn <ssh9614@hotmail.com>*
*Date: Wed, 26 Aug 2009 13:17:18 -0700 PDT*
*Subject: Re: WinPCap/LibPCap Wrappers?*
*Newsgroups: comp.lang.ada*

[…]

The following might be of help (only tested with libpcap):

http://www.mediafire.com/?sharekey=b27d9dbe9e00e3f4e62ea590dc5e5dbbe04e75f6e8ebb871

*From: Steve D*
*Date: Wed, 26 Aug 2009 21:04:55 -0700*
*Subject: Re: WinPCap/LibPCap Wrappers?*
*Newsgroups: comp.lang.ada*

[…]

The files are kind of big, but I have had wireshark dump the files as XML (one of the options). It made it relatively easy to sift through data.

## Doxygen for Ada

*From: Pablo Vieira Rego <pablittto@gmail.com>*
*Date: Mon, 20 Jul 2009 08:38:42 -0700 PDT*
*Subject: Doxygen for Ada*
*Newsgroups: comp.lang.ada*

Does someone knows some Doxygen documentation script which runs over Ada code?

*From: Oliver Kellogg <okellogg@freenet.de>*
*Date: Mon, 20 Jul 2009 11:50:30 -0700 PDT*
*Subject: Re: Doxygen for Ada*
*Newsgroups: comp.lang.ada*

[…]

AdaBrowse (http://home.datacomm.ch/t_wolf/tw/ada95/adabrowse/) does not use the exact Doxygen syntax but is useful, nonetheless.

*From: Pablo Vieira Rego <pablittto@gmail.com>*
*Date: Wed, 22 Jul 2009 05:13:29 -0700 PDT*
*Subject: Re: Doxygen for Ada*
*Newsgroups: comp.lang.ada*

[…]

Hi Oliver, thanks for the answer. I was looking for a script which generates .tex files, but meanwhile it's good.

*From: Maciej Sobczak <maciej@msobczak.com>*
*Date: Tue, 21 Jul 2009 01:07:07 -0700 PDT*
*Subject: Re: Doxygen for Ada*
*Newsgroups: comp.lang.ada*

[…]

I have written a script that generates pretty nice docs in the Doxygen/Javadoc/etc. style, but it works differently from other tools. Instead of analyzing the code on the ASIS level it assumes that the source code is "well formatted" and processes it based on the vertical and horizontal white space patterns. Due to the regular structure of the Ada grammar the whole task is reduced to simple regexp crunching without any serious compile-like analysis - and is therefore blazing fast. I don't claim that its language coverage is complete, but it proved to be very useful.

The script generates a set of HTML pages with indexes and stuff.

As pointed above, it works *only* when the code is "well formatted", which means that newlines and indentation should be used according to common conventions. Fortunately, most of the Ada code that was ever written is well formatted in this sense. I was able to use this script for the libraries that come with GNAT, and also for AWS, PolyORB, etc.

Please contact me privately […] if you are interested in this script. If there will be a wider interest I will publish it on the web.

[…]

## Status of Hibachi

*From: John McCabe <john@nospam.assen.demon.co.uk>*
*Date: Mon, 08 Jun 2009 10:13:23 +0100*
*Subject: Hibachi - is it dead?*
*Newsgroups: comp.lang.ada*

Just thought I'd ask as someone on here might know.

*From: britt.snodgrass@gmail.com*
*Date: Mon, 8 Jun 2009 08:11:47 -0700 PDT*
*Subject: Re: Hibachi - is it dead?*
*Newsgroups: comp.lang.ada*

[…]

I haven't yet bothered to remove myself from the "hibatchi-dev" list.

With one exception, the *only* list traffic I've seen in over a year is this automatic message posted every two months:

> Thomas,

> Projects are required to keep meta data up to date using the MyFoundation

Portal (http://portal.eclipse.org/). The following problems were found with this project's meta-data:

- There is no next/future release of this project. All Eclipse projects must have a "next release" planned and scheduled.

[…]

*From: Tom Grosman <grosman@aonix.fr>*
*Date: Wed, 24 Jun 2009 16:01:23 +0200*
*Subject: Re: Hibachi - is it dead?*
*Newsgroups: comp.lang.ada*

[…]

Sorry about the tardy response. I was on holiday and your question got lost in the pile when I got back.

We (Aonix) are still using and developing the Hibachi technology via AonixADT, our Eclipse IDE for ObjectAda, however other than bug fixes, most of the work we are doing has been specific to our products (eg. adding support for specific ObjectAda tool chains). We have not yet rolled these changes into the Hibachi sources.

There hasn't been any activity on the Hibachi project in a while. I had to step down as project leader for health reasons, and since then there has not really been anyone driving it. I had hopes that there would be someone to pick up the baton and move things forward, but unfortunately, that didn't happen.

The project is still there, an excellent base from which to create an extensible robust Ada development framework, but for various reasons, there hasn't been the critical mass needed to support it.

[see also "Hibachi Project Lead" in AUJ 29-3 (Sep 2008), p.156 —mp]

### Zip-Ada v.33

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Thu, 2 Jul 2009 14:49:02 -0700 PDT*
*Subject: Ann: Zip-Ada v.33*
*Newsgroups: comp.lang.ada*

[…]

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

- Changes in '33', 18-Jun-2009:
  - o UnZip: added extract_as_text option (cf. UnZipAda with -a option)
  - o Zip: Zip_comment function added (cf. UnZipAda with -z option)

Zip-Ada is now used in the following musical software:

http://www.huygens-fokker.org/scala/

[see also "Zip-Ada" in AUJ 30-1 (Mar 2009), p.11 —mp]

### L10n and i18n support in Ada

*From: Vadim Godunko <vgodunko@gmail.com>*
*Date: Fri, 7 Aug 2009 14:56:41 -0700 PDT*
*Subject: Announce: localization, internationalization and globalization for Ada*
*Newsgroups: comp.lang.ada*

[…]

Ada standard library doesn't include features for localization, internationalization and globalization which is very important for many kinds of information systems. I am pleased to announce the first release of library for this domain.

It allows to manipulate string information encoded using full Unicode character set, supports upper/ lower case conversions, normalization, collation (string comparison in the user preferred order, not in code point order), characters and grapheme cluster cursors on strings. You can download it here:

http://download.qtada.com/matreshka-0.0.1.tar.gz

Any feedback is welcome!

## Ada-related Products

### AdaCore — GNAT Pro for all Current VxWorks Platforms

*From: AdaCore Press Center*
*Date: Monday June 1, 2009*
*Subject: AdaCore Expands GNAT Pro Offerings to All Current VxWorks Platforms*
*URL: http://www.adacore.com/2009/06/01/vxworks-67/*

SAN DIEGO, NEW YORK and PARIS, June 1, 2009 - Avionics USA - AdaCore, a leading supplier of Ada development tools and support services, today announced the availability of GNAT Pro on all active versions of Wind River's VxWorks real-time operating system. This list includes VxWorks 5, VxWorks 6, supporting both asymmetric multiprocessing (AMP) and symmetric multiprocessing (SMP) configurations, VxWorks 653, VxWorks DO-178B, and VxWorks MILS. GNAT Pro now gives Ada users the widest selection of VxWorks platforms to choose from, whether for new programs or for migration of existing codebases to newer operating systems. In addition, all Wind River platforms supporting deployment in DO-178B certified environments are also supported.

The GNAT Pro product offering for VxWorks satisfies a wide variety of customer application requirements and deployment environments. GNAT Pro for VxWorks 5 and 6 are suitable for general-purpose software development. The specialized GNAT Pro High-Integrity Edition for DO-178B, available for the VxWorks 653 and VxWorks DO-178B Platforms, is targeted to RTCA DO-178B and IEC 61508 safety-critical systems.

For safety-certified environments, AdaCore's GNAT Pro High-Integrity Edition supports three specific runtime environments that optimize size, performance, and certification effort.

- The zero-foot-print (ZFP) library simplifies safety certification for sequential applications.
- The Ravenscar library adds support for tasking or multi-processing.
- The Cert library adds ARINC653 awareness and supports advanced integrated modular avionics (IMA) features like inter-process or inter-partition communication and synchronization.

These certification libraries are used as a part of multiple DO-178B Level A certified systems.

GNAT Pro is also available for VxWorks 6, supporting both symmetric multiprocessing (SMP) and asymmetric multiprocessing (AMP) multicore implementation models. This capability allows customers to deploy Ada on advanced multicore processors, and enables high levels of optimization for a variety of challenging multiprocessor designs and configurations.

AdaCore's GNAT Pro product for VxWorks MILS brings a powerful application development environment to the aerospace and defense industry for developing Ada applications within a MILS (multiple independent levels of security) architecture.

GNAT Pro for VxWorks MILS combines with SPARK Pro (an Integrated Development Environment combined with the SPARK language tool set) to support security certification of user applications or their components to Evaluation Assurance Level (EAL) 5 and higher. SPARK is an Ada subset extended with a contract language that allows a program's specification to be precisely expressed and verified. It directly supports semi-formal and formal methods as required by EAL 5 and higher, and has a strong track record for both safety-critical and high-security systems. An example of the latter is the NSA-sponsored Tokeneer project (see: http://www.adacore.com/2008/10/06/nsa-releases-secure-software-project-to-open-source-community/)

"Ada is an important technology for Wind River aerospace and defense customers, and AdaCore leads the way in providing Ada toolsets across the full spectrum of

VxWorks platforms," said Rob Hoffman, Vice President and General Manager of Aerospace and Defense at Wind River.“ Ada is well suited for the high-integrity domains that we target, and we are pleased to see GNAT Pro available for our newest products, including VxWorks 6.7 and VxWorks MILS."

"Wind River's platforms are a key market for AdaCore, so we placed a high priority on making GNAT Pro available on all the latest versions of VxWorks," said Robert Dewar, President and CEO of AdaCore." And our new support for VxWorks MILS, in particular the SPARK Pro tool set, provides a unique capability of creating applications suitable for medium to high assurance deployment. AdaCore's SPARK Pro tool set is the only program analysis tool to directly help developers certify their high assurance applications in MILS architectures."

About AdaCore

Founded in 1994, AdaCore is the leading provider of commercial software solutions for Ada, a state-of-the-art programming language designed for large, long-lived applications where safety, security, and reliability are critical.

AdaCore's flagship product is the GNAT Pro development environment, which comes with expert on-line support and is available on more platforms than any other Ada technology. AdaCore has an extensive world-wide customer base; see http://www.AdaCore.com/home/company/customers/ for further information.

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

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

## AdaCore — GNAT GPL for Lego Mindstorms NXT

*From: AdaCore Libre Site*
*Date: Tuesday September 1, 2009*
*Subject: GNAT GPL for LEGO MINDSTORMS NXT*
*URL: http://libre.adacore.com/libre/tools/mindstorms/*

GNAT for Lego Mindstorms NXT is a GPL port for the GNAT compilation system to the Lego Mindstorms NXT robotic platform. Originally born as an education-oriented project at MIT Media Lab, the Lego Mindstorms has evolved into a successful commercial product for education in robotics in a multitude of Universities and high schools across the globe.

The latest revision of the platform includes a 32-bit processor and supports several different sensors able to detect distance, colors and sounds and to communicate via the Bluetooth protocol.

GNAT GPL Edition for the Lego Mindstorms NXT platform brings the possibility of experimenting with embedded systems development using the Ada 2005 and SPARK languages to an education-oriented robotic platform. Entire embedded systems, including software, hardware and sensors interfacing, and wireless communications can be developed and verified using the GPL editions of GNAT and SPARK.

Technical Details

GNAT GPL Edition for Lego Mindstorms NXT relies on the nxtOSEK operating system to manage real-time and concurrent execution.

GNAT for Lego Mindstorms requires to erase the original Lego firmware; the original firmware can be however restored if necessary.

Here's what's included:

- GNAT GPL edition targeting the LEGO MINDSTORMS NXT Platform
- Bindings to nxtOSEK, an open, real-time operating system providing facilities for real-time and concurrent systems
- Bindings to access MINDSTORMS hardware (processor, sensors and motors), including Bluetooth
- Getting Started material, examples of applications which can be used as teaching material
- Availability for the LEGO MINDSTORMS robotic building system (coming soon).

Lego Mindstorms NXT NOT included

More information

[http://www.slideshare.net/AdaCore/gnat-gpl-for-mindstorms?type=powerpoint —mp]

[…]

Authors and Contributors

GNAT for LEGO MINDSTORMS is developed and maintained by AdaCore.

Piotr Piwko contributed to the development during his internship at AdaCore in Autumn 2008.

For more information on the LEGO MINDSTORMS NXT platform, see http://mindstorms.lego.com

[verbatim from the project website —mp]

## AdaLog — AdaControl 1.11r4

*From: Jean-Pierre Rosen <rosen@adalog.fr>*
*Date: Wed, 01 Jul 2009 11:13:52 +0200*
*Subject: New release of AdaControl (1.11r4)*
*Newsgroups: comp.lang.ada*

This is really a new release, not a new version (i.e. no new rules).

Apart from some small bug fixes, the main difference is that the compiled versions are now for GNAT-GPL2009.

As usual, it can be downloaded from http://www.adalog.fr/adacontrol2.htm

[see also "AdaLog — AdaControl 1.11r3" in AUJ 30.2 (Jun 2009), p.78 —mp]

# Ada and GNU/Linux

## Static linking with gcc-4.4

*From: Markus Schoepflin <markus.schoepflin@comsoft.de>*
*Date: Fri, 07 Aug 2009 09:54:00 +0200*
*Subject: Static linking*
*Newsgroups: comp.lang.ada*

[…]

up to now (gcc-4.3.x) we have been happily using 'gnatmake … -largs -static' to create statically linked executable. This has stopped working with 4.4:

```
 > touch foo.adb && gnatmake foo -largs
   -static
gcc-4.4 -c foo.adb
gnatbind -x foo.ali
gnatlink foo.ali -static
/usr/bin/ld: cannot find -lgnat-4.4
collect2: ld returned 1 exit status
gnatlink: error when calling /usr/bin/gcc-
4.4
gnatmake: *** link failed.
```

Now I have been told that -static is a binder, not a linker argument, and indeed this works:

```
 > touch foo.adb && gnatmake foo
   -bargs -static
gcc-4.4 -c foo.adb
gnatbind -static -x foo.ali
gnatlink foo.ali
```

But this is not a static executable:

```
 > ldd foo
  linux-gate.so.1 =>  (0xb7f9e000)
  libc.so.6 => /lib/i686/cmov/libc.so.6
               (0xb7e2d000)
               /lib/ld-linux.so.2
               (0xb7f9f000)
```

Looking at the manual for the binder I'm told: '-static Link against a static GNAT run time.'.

OK, so this works as advertised, no dynamic GNAT runtime used.

Now what is the correct way to create a static executable? This seems to work, but is it correct?

```
 > touch foo.adb && gnatmake foo
    -bargs -static -largs -static
gcc-4.4 -c foo.adb
gnatbind -static -x foo.ali
gnatlink foo.ali -static
 > ldd foo
   not a dynamic executable
```

And why did only passing '-largs -static' work for gcc-3.3.x?

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Fri, 7 Aug 2009 03:15:38 -0700 PDT*
*Subject: Re: Static linking*
*Newsgroups: comp.lang.ada*

> […] Looking at the manual for the binder I'm told: '-static Link against a static GNAT run time.'. OK, so this works as advertised, no dynamic GNAT runtime used.

> […]

libc6 does not exist as a static library anymore; on GNU/Linux there is no way to create a completely static executable; you have to link against (at least) libc.so.6 dynamically. So, the executable you obtain with -bargs -static is as static as you can get :)

> > touch foo.adb && gnatmake foo -bargs -static -largs -static

> gcc-4.4 -c foo.adb

> gnatbind -static -x foo.ali

> gnatlink foo.ali -static

> > ldd foo

>   not a dynamic executable

>

> And why did only passing '-largs -static' work for gcc-3.3.x?

I don't know and it seems strange to me that it worked at all if the binder was not called with -static. Maybe others on comp.lang.ada have more to tell us?

*From: Markus Schoepflin <markus.schoepflin@comsoft.de>*
*Date: Fri, 07 Aug 2009 12:46:36 +0200*
*Subject: Re: Static linking*
*Newsgroups: comp.lang.ada*

Now I'm really confused. The command just below your answer in my original mail works and gives me a static executable.

[…]

*From: Alex R. Mosteo <alejandro@mosteo.com>*
*Date: Wed, 12 Aug 2009 16:20:43 +0200*
*Subject: Re: Static linking*
*Newsgroups: comp.lang.ada*

[…]

Not entirely sure is the same issue, but maybe it helps our understanding. I got also different results using some same switches when using GNAT GPL 2008 and GPL 2009. I opened a GAP support ticket and their reply was that some defaults had changed, but you could force the same results with explicit switches. In my case it involved mixed Ada/C++ linking and they also said that they were working on some internal cleaning that would improve the situation.

There's another confusing issue in which you don't have to check the ld documentation but gcc/g++ one, which is the one getting the switches in the end.

I don't know if I could quote the support reply I got but, in addition to Ludovic comment about "as static as it can be" given the shared libc library, the switches I'm using right now to control linking are:

```
package Binder is
   for Default_Switches ("Ada")
        use ("-static");
   - -  -static/-shared makes the gnat
         runtime static or shared
end Binder;

package Linker is
   for Default_Switches ("Ada") use
     ("-Wl,-Bstatic",  - - Starts static
                              linking section
      "-lz",          - - Sample libraries that
                          I want statically linked.
      "-lgsl",
      "-lgslcblas",
      "-Wl,-Bdynamic", - - Starts shared
                             linking section
      "-ldl"          - - Sample library
                          dynamically linked in
     );
end Linker;
```

[…]

## Cross-compilation for sparc64-freebsd

*From: Allison Phillips <cam@ally.com>*
*Date: Sat, 29 Aug 2009 10:53:23 +0200*
*Subject: compiling for sparc64*
*Newsgroups: comp.lang.ada*

I have a linux-i686 pc and I need to cross-compile a program with GNAT to a sparc64-freebsd target.

Are there some parameters for the compiler?

I searched in google and in the old messages of this newsgroup, but no answers.

I don't need to have a sparc64 version of GNAT, only the compiled program.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Sat, 29 Aug 2009 07:45:30 -0700 PDT*
*Subject: Re: compiling for sparc64*
*Newsgroups: comp.lang.ada*

You need to build a cross-compiler from the gnat sources. GCC requires the native and cross compilers to be from the same sources; therefore you have to build your cross-compiler from the exact same sources as your native compiler.

To build a cross-compiler, follow the installation instructions[1],

passing --target=sparc64-freebsd to ../src/configure.

[1] http://gcc.gnu.org/install/

[…]

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Sat, 29 Aug 2009 07:57:19 -0700 PDT*
*Subject: Re: compiling for sparc64*
*Newsgroups: comp.lang.ada*

I feel I have to add to this… All UltraSPARC processors since circa 1994 are 64-bit but can also run 32-bit code natively. There is a memory and performance cost with running 64-bit code because the larger addresses take more room in the cache. For this reason, on Solaris, Sun recommends that applications that can work within 4 GiB be compiled in 32-bit mode. The norm is a 64-bit kernel running multiple 32-bit apps and a few large 64-bit apps that absolutely require the large address space and trade some performance for it.

You may want to check whether that also applies to FreeBSD or not.

GCC supports both sparc and sparc64.

## Debian Squeeze — Status of gnat-4.4

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Tue, 28 Jul 2009 11:59:08 -0700 PDT*
*Subject: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

My plan for the next release of Debian, code-named Squeeze, is as follows:

2009-02-15: Debian 5.0 "Lenny" released

2009-07-28: gnat-4.4 reaches unstable

2009-09-30: libgnat-4.4 ABI freeze; start of transition of all Ada packages

2010-02-15: end of support for Debian 4.0 "Etch" (i.e. Lenny+12 months)

2010-02-28: end of transition and, probably, freeze of all packages in Debian

2010-03-*: removal of gnat-4.3 sometime between 2010-08 and 2011-02: release of Squeeze, i.e. between18 and 24 months after Lenny.

Today marks the first milestone on this roadmap. gnat-4.4 includes all of the patches from gnat-4.3 (adjusted as needed) but also some patches from GCC 4.5 needed to make the Distributed

Systems Annex work again (with PolyORB).

I need your help testing the compiler and its run-time libraries against your programs and fixing any bugs you find. During this time, I am willing to patch the compiler and its run-time library as needed for stability or even to backport new features from the trunk (GCC 4.5). After the ABI freeze, no changes will be accepted to the run-time library anymore (i.e. the .ali files will be frozen) so as to guarantee binary compatibility of all Ada packages.

If you have some software that you would like to package for Debian, I encourage you to use gnat-4.4, instead of gnat-4.3, starting now. gnat-4.4 will be the only Ada compiler in Debian Squeeze.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Thu, 30 Jul 2009 06:53:19 -0700 PDT*
*Subject: Re: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

[…]

I'm sorry but I'll have to change this plan drastically. The recent announcement of the new two-year time-based release schedule [1] caught me by surprise, especially the fact that Squeeze will actually have a one-year release schedule :(

[1] http://www.debian.org/News/2009/20090729

This leaves only 4 months to complete the transition of all Debian packages. Because of this, I'm officially abandoning my goal of providing support for multi-arch (32 and 64-bit) and cross compilers in Squeeze and I will shorten the time for compiler testing. The new schedule is:

2009-07-28: gnat-4.4 reaches unstable. Immediate start of transition for other packages.

2009-08-31: libgnat-4.4 ABI freeze and re-upload of dependent packages as necessary.

2009-09-30: compiler freeze.

2009-11-30: end of transition and package freeze (release-critical bug fixes still allowed).

2010-12-31: gnat-4.3 removed from Debian.

It is possible that I cannot complete the transition of all packages due to lack of time (remember: I'm a volunteer working on my spare time on Debian!). In such an event, I will request *REMOVAL* of some packages from Debian. The packages that have a Request for Help or Request for Adoption bug open against them will be the first packages dropped; they are:

gnat-gps (RFH: http://bugs.debian.org/496905)

libtexttools (RFA: http://bugs.debian.org/477474)

gnade (RFA: http://bugs.debian.org/496787)

So, if you use or have any interest in those packages, please help with the transition. You do not have to be a Debian Developer to help or even to adopt a package officially. If I do not receive any help, I will conclude that nobody is interested enough to justify the effort and simply drop the packages mentioned.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Mon, 10 Aug 2009 13:23:47 -0700 PDT*
*Subject: Re: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

Someone asked me privately, but I feel this is of interest to everyone:

> Do you have plans to add gprbuild and ASIS also?

I have plans to update ASIS to the GNAT GPL 2009 version; this will be part of the library transition that I announced here. I will in fact update all existing packages.

As for gprbuild, this is a new package and I'm not planning to do it myself as I am in fact trying to concentrate on fewer packages. If someone is interested in packaging GPTBuild, please do not hesitate; as I have said repeatedly:

- I will help would-be packagers of Ada software in Debian; just contact me privately if you need pointers.
- you do not have to be a Debian Developer before you contribute (in fact the opposite is true: you must own and maintain packages in Debian before you are even considered for Debian Developer status).

The Debian release manager has admitted that his announcement [1] concerning the new release schedule was premature. So the plan for a freeze in December 2009 is officially abandoned [2]. A new plan and schedule are due in September.

[1] http://www.debian.org/News/2009/20090729

[2] http://www.debian.org/News/2009/20090730

This does have an impact on my own schedule and I am no longer in a big hurry; I will await the decision in September and, based on that, will decide on my own release goals. Tentatively:

- enable support for multiarch (i.e. for i386/amd64 and others: emit 64-bit binaries on 32-bit platforms and vice-versa; for mips/mipsel, emit little-endian code on big-endian platforms and vice-versa). This is a new feature in GCC 4.4.
- enable support for the Distributed Systems Annex through the addition of PolyORB. Xavier Grave and Reto Buerki are working on this and have already managed to get CORBA and MOMA (Message-Oriented Middleware Architecture) working. I am sponsoring the package for them.

The bottom line is: for now, please continue to test gnat-4.4 as extensively as you can, and please consider helping out with a package you are particularly interested in. I have already received valuable feedback in the form of bug reports (and encouragements, thanks for that!) and I trust that Debian Squeeze will not only be worthy of Debian's reputation, but its support for Ada will raise the bar even higher.

*From: Tero Koskinen <tero.koskinen@iki.fi>*
*Date: Thu, 30 Jul 2009 23:41:16 +0300*
*Subject: Re: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

> Where can one find gnat-4.4 binary packages?

> I installed Sid inside Debian 5.0.1 chroot […]

Oh, and I am using i386 platform (no access to amd64).

According to http://packages.debian.org/sid/gnat-4.4, there is no packages for i386, is this correct?

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Sat, 1 Aug 2009 06:02:27 -0700 PDT*
*Subject: Re: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

> […]

> The package built successfully on i386[1] but apparently it has not reached the archive yet. Funny that hppa, s390 and sparc should reach the archive before i386 does :)

> Please retry in a day or two; sorry for the delay. I'll monitor that space and complain to the Debian buildd admins if the i386 package doesn't show up by saturday evening.

The i386 packages are now in the archive; i386 users rejoice :)

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Wed, 5 Aug 2009 10:30:57 -0700 PDT*
*Subject: Re: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

[…]

> I think gnat-4.4 should either include gcc as a dependency, call gcc-4.4, or something like that.

It is normally patched to call gcc-4.4, yes. I'll look into it. If there is a place where it calls gnatgcc instead (which is part of package gnat), I'll fix it. Thanks for reporting.

> Btw, I noticed that many Ada packages (aws, adacontrol) still depend on gnat-4.3, are those updated at some point also?

Yes, when I get around to it. That will involve updates to newer versions and soname bumps.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Mon, 10 Aug 2009 13:43:37 -0700 PDT*
*Subject: Re: Ada in Debian: gnat-4.4 is in unstable; please test*
*Newsgroups: comp.lang.ada*

[…]

Before anyone who doesn't know GCC very well asks: GCC has had support for multiarch for several years now but the Ada front-end was lagging behind; GCC 4.4 adds support for multiarch in the Ada front-end.

Also, multiarch is difficult. There are deep issues with holistic system architectures and decisions to be made; 32/64-bit support is hairy enough as it is but consider it only one particular case of multi-arch support. I mentioned little-endian/big-endian already and we can have multiple combinations. For example, I gather modern PowerPC processors can run all four combinations of 32-bit/LE, 32-bit/ BE, 64-bit/LE and 64-bit/BE *concurrently*, not to mention emulated architectures, the Synergistic Processing Units of the Cell processor and GPUs used for number crunching.

[…]

### Debian — On the mapping of Ada tasks to OS threads

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: 26 Aug 2009 14:20:31 +0200*
*Subject: [Debian] Switching tasking implementation?*
*Newsgroups: comp.lang.ada*

If I remember correctly, the version of GNAT distributed with Debian supports more than one tasking implementation.

Do I remember correctly? How do I select which implementation to use?

[…]

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: 26 Aug 2009 15:00:33 +0200*
*Subject: Re: Switching tasking implementation?*
*Newsgroups: comp.lang.ada*

[…] What I am interested in, is to compare the performance of tasking based on OS threads with tasking based on user-space threads.

[…]

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Wed, 26 Aug 2009 10:38:28 -0700 PDT*
*Subject: Re: Switching tasking implementation?*
*Newsgroups: comp.lang.ada*

[…] the two run-time libraries shipped in Debian allow you to choose either zero-cost or setjump/longjump exception handling mechanisms.

The tasking model always uses the 1:1 model where each Ada task corresponds to a kernel thread. I vaguely remember there used to be a n:1 model where all Ada tasks would run in the same kernel thread, doing scheduling in userspace. I'm not sure whether this model is still available or not nowadays. Could you please look this up in the GCC installation manual? This was a configure option IIRC.

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: 27 Aug 2009 15:54:10 +0200*
*Subject: Re: Switching tasking implementation?*
*Newsgroups: comp.lang.ada*

[…]

Thanks. It doesn't look like it is an option any more.

## Ada and Microsoft

### Implementation of a COM interface

*From: Pablo Vieira Rego <pablittto@gmail.com>*
*Date: Thu, 30 Jul 2009 09:26:11 -0700 PDT*
*Subject: Component Object Model*
*Newsgroups: comp.lang.ada*

Does someone know how to implement COM interface in Ada? I found an article which cites a how-to in adapower.com, but the link does not exist there.

[…]

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Thu, 30 Jul 2009 10:11:51 -0700 PDT*
*Subject: Re: Component Object Model*
*Newsgroups: comp.lang.ada*

[…]

Did you heard about Orbit ? It is a CORBA binding for Ada

You may see there to learn more :

http://orbitada.sourceforge.net/

http://sourceforge.net/projects/orbitada/

It seems to be a kind of a de-facto standard (I often see mentions to Orbit in the Ada area)

*From: David Gressett <gresset1@airmail.net>*
*Date: Thu, 30 Jul 2009 12:58:12 -0700 PDT*
*Subject: Re: Component Object Model*
*Newsgroups: comp.lang.ada*

[…]

Look on SourceForge for the GNAVI project.

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Thu, 30 Jul 2009 12:37:04 -0700 PDT*
*Subject: Re: Component Object Model*
*Newsgroups: comp.lang.ada*

Note: if the link which does not work is the one of GNATCOM, I confirm it does not work for me too.

More infos which may help you :

"CORBA vs COM/DCOM" (french - do you read french ?)

http://www-lsr.imag.fr/users/Christine.Plumejeaud/NFE107-fichesLecture/CORBA%20vs%20COM.ppt

"Interfacing Ada 95 to Microsoft COM and DCOM Technologies" (PDF - english, this one)

http://www.sigada.org/conf/sigada99/proceedings/p9-botton.pdf

[…]

## References to Publications

### COTS Journal — "Ada Language Suits Real-Time Safety-Critical Needs"

*From: AdaCore Press Center*
*Date: Monday August 31, 2009*
*Subject: Ada Language Suits Real-Time Safety-Critical Needs*
*URL: http://www.adacore.com/category/press-center/articles/*

[read the article at:

http://www.cotsjournalonline.com/magazine/articles/view/100982/pg:1 —mp]

### VME Critical Systems — On language subsetting for safety

*From: AdaCore Press Center*
*Date: Wednesday June 24, 2009*
*Subject: When less is more: Programming language technology for safety*
*URL: http://www.adacore.com/category/press-center/articles/*

[read the article at:

http://www.vmecritical.com/articles/id/?4030 —mp]

### Embedded Computing Design — "Making static analysis a part of code review"

*From: AdaCore Press Center*
*Date: Friday June 19, 2009*
*Subject: Making static analysis a part of code review*
*URL: http://www.adacore.com/category/press-center/articles/*

[read the article at:

http://www.embedded-computing.com/articles/id/?4014 —mp]

## Ada Inside

### AdaCore — GNAT Pro used for Airbus A350 XWB

*From: AdaCore Press Center*
*Date: Monday June 1, 2009*
*Subject: Thales Aerospace Division Selects GNAT Pro for Airbus A350 XWB (Xtra Wide-Body)*
*URL: http://www.adacore.com/2009/06/01/a350/*

SAN DIEGO, NEW YORK and PARIS, June 1, 2009 - Avionics USA - AdaCore, a leading supplier of Ada development tools and support services, today announced that international electronics and systems group Thales has chosen the GNAT Pro technology, including several safety-qualified tools, to develop critical systems for the new Airbus A350 XWB (Xtra Wide-Body) family.

Thales will use the GNAT Pro High-Integrity Edition for DO-178B and the Ada 2005 language to build the Air Data Inertial Reference Unit (ADIRU) for the A350 XWB (Xtra Wide-Body). The ADIRU provides precise in-flight positioning information, and the new system will therefore need to be certified to the highest safety levels. It will meet Level A of the DO-178B standard and use ARINC 653 multi-partition operating system MACS2.

The project promises to advance the state of the art in safety-critical development through a number of innovations including the application of Agile Programming techniques and the safe use of Object-Oriented Programming (OOP) features. Thales will be using the Ada 2005 version of the Ada language, which has introduced additional support for real-time and safety-critical systems and for safe/reliable OOP.

“When it comes to safety-critical systems, the Ada language has an impressive pedigree and track record,” said Francois Brun, Software Design Authority at Thales Aerospace Division - Navigation Unit. “The combination of GNAT Pro and Ada 2005 provides the technology we need to develop the ADIRU software for the new Airbus A350 XWB (Xtra Wide-Body).”

AdaCore is also providing a Qualified Code Standard Checker and a Coverage tool for this program. These tools will be key to the development process of the ADIRU and the generation of safety evidence. AdaCore’s Coverage solution adopts an innovative approach by providing MC/DC coverage information on uninstrumented source code, through the use of a PowerPC simulator. Tool qualification material gives credit to the output of the tools, which speeds up the certification process.

“Thales is leading the way when it comes to safety-critical and avionics system development,” said Michaël Friess, Technical Sales Manager of AdaCore.

“It is a pleasure to partner with such a technically advanced team. Our long collaboration with Thales and our regular exchanges with their technical staff have led to strategic advances in AdaCore’s offering. ADIRU is another example of a project that helps our technology progress, and that shows customers that they can count on AdaCore as a reliable partner in their development process.”

The A350 XWB (Xtra Wide-Body) Family is Airbus’ response to widespread market demand for a series of highly efficient, medium-capacity, long-range, wide-body aircraft. With a range of up to 8,300 nm / 15,400 km, it is available in three basic passenger versions: the A350-800, the A350-900 and the A350-1000.

### Indirect Information on Ada Usage

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

*Job offer [United Kingdom]:* Software Engineer

The role is to develop Ada 95 software applications and create tests based on previously prepared UML use cases. The Artisan Studio development environment uses templates to help produce an Object-oriented software framework for the Tactical Processor system. The application software is intended to run as part of an open architecture.

General Requirements

- Avionics experience desirable
- Ability to work in a team under tight timescales essential
- Experience of DO-178B useful

Software Test Engineer

- 5+ years test experience of large systems at the system/ integration level
- Not unit testers
- Use case familiarisation - source of requirements
- Artisan Studio familiarisation - exploring Artisan use case model (or similar tool)
- Writing test scripts against use cases
- Hardware/Software integration
- Target environment using 'Green Hills AdaMULTI'
- DOORS

*Job offer [United Kingdom]:* Software Design Engineer

[…]

A Software Developer with expertise in design, development, and integration of real-time embedded software projects for a major military and maritime helicopter.

Duties / Responsibilities:

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

- Planning and control of own work
- Real time embedded software development with UML
- High-level and detailed design using object-oriented methodologies and a UML toolset (Artisan Studio)
- Code development in Ada 95

Desirable Skills:

- Development in Ada 95
- Familiarity with using object oriented methodologies (OOD) and a UML toolset (Artisan Studio)

[…]

*Job offer [United Kingdom]:*

[…] 2x Ada 83/95 developers to start ASAP on initial 6 month assignments.

Essential Skills:

- Ada 83 and 95 development experience (not test/verification)
- Strong Ada/Hardware electronics interfacing experience

Bonus Skills:

- Rational Rose
- HOOD

*Job offer [United Kingdom]:* Ada Unit Test Engineers

[…]

Qualifications: Experience or knowledge of battery charging systems

[…] strong skills in Ada95 and UML.

You will be working on the module testing of Ada95 SPARK applications written for the avionics industry to meet DO-178B & DEF STAN-00-55 SIL 4 standards.

The design and requirements are in UML (Artisan) and DOORS (Rational).

Any additional knowledge of automated test tools such as VectorCAST or LDRA would be useful.

[…]

*Job offer [United Kingdom]:* Senior Software Engineer - Ada, C, Assembler

[…] an experienced software professional to join a team designing real-time safety critical software.

The position requires full-lifecycle software design experience as your duties include diagnosis of software failures and integration testing.

[…]

Software Engineer Key Responsibilities:

- Analysis of product software failures at the high-level code and assembler levels
- Performance profiling of product software
- Formal development of real-time safety critical software (SIL2 and 4) - Ada83/95, C, Assembler
- Development of unit and integration test environments
- Creation of diagnostic and development support tools

Software Engineer Skills and Qualifications:

- Graduate in relevant engineering discipline or other numerate subject
- Extensive experience in real-time software design including software architecture, scheduling and multi-lane synchronization
- Ability to comprehend new software rapidly in order to diagnose the causes of product failure
- Experience of the software development process for safety-related (SIL 2 and 4) developments.
- Proficient in Ada83, Ada95, C and assembler in technical applications
- Proficient in software unit test, software integration test and hardware/software integration test techniques.

*Job offer [Belgium]:*

[…] 2x Ada Developers to join a global Transport organization at their offices in Belgium.

To be successful you need:

- Multi-year experience Ada83 and/or Ada95 programming (5 years preferred)
- Very strong C++ development experience
- Industrial experience required of multi-threaded applications programming in high-level languages
- Experience of Microsoft Office tools
- Good knowledge of the transport industry
- Very good communication skills in English and French

*Job offer [Italy]:*

The ideal candidate must have at least 5 years' experience in C/C++ and/or Ada programming (Ada 83 and Ada 95, knowledge of Ada 2005 is desirable). Required experience in the phases of analysis, design, development and testing of real-time applications.

[…]

Knowledge of the Structured Analysis and Design Method, as well as Object-Oriented Analysis and Design are an advantage.

Knowledge of DODAF, SysML, UML 2.0, MDA and model-driven development techniques are an advantage.

[…]

[Translated from Italian —mp]

*Job offer [United States]:*

The Software Engineer provides functional and empirical analysis related to the design, development, and implementation of software systems, including, but not limited to application software, utility software, development software, and diagnostic software. The Software Engineer participates in the development of test strategies, devices, and systems. Have the technical background and skills to perform in all phases of software design, development, documentation, and implementation. The Software Engineer shall have the ability to design and develop with of advanced networking techniques.

Specific Duties:

- Detailed software design
- Design walkthroughs
- Code implementation
- Code walkthroughs
- Unit and integration testing
- Software documentation
- Requirements management
- Teamwork & team building

Required Knowledge - Skills - and Abilities: Bachelors Degree required.

Additional Requirements:

[…]

- Must be capable of developing software or prototype applications in at least one high-level programming language (such as C/C++, Java, Ada) in a Windows or Unix development environment.
- Must be capable of implementing advanced solutions based on requirements specifications and user feedback with minimal management and technical supervision.
- Must have a good understanding of the software development process, object-oriented development, and information engineering or relevant CASE tools.

[…]

- Must have demonstrated ability to develop and debug applications in desired programming language and environment, with minimal management and technical feedback.

Desired Knowledge - Skills - and Abilities:

Preferred:

- Master's Degree in computer science, electronics engineering or other engineering or technical discipline preferred.

## Ada in Context

### Class attribute and non-tagged incomplete types

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Sat, 1 Aug 2009 23:55:16 -0700 PDT*
*Subject: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

[…]

I've learned something mostly surprising today: in Ada 95, it was allowed to use the Class attribute with a prefix which was an incomplete type… even not tagged.

I've learned about it here :

http://www.adaic.org/standards/05rat/html/Rat-1-3-3.html

>> ------------------

The introduction of tagged incomplete types clarifies the ability to write

type T_Ptr is access all T'Class;

This was allowed in Ada 95 even though we had not declared T as tagged at this point. Of course it implied that T would be tagged. In Ada 2005 this is frowned upon since we should now declare that T is tagged incomplete if we wish to declare a class wide access type. For compatibility the old feature has been retained but banished to Annex J for obsolescent features.

>> ------------------

The part of the mentioned annex J

>> ------------------

For the first subtype S of a type T declared by an incomplete_type_declaration that is not tagged, the following attribute is defined:

S'Class

Denotes the first subtype of the incomplete class-wide type rooted at T. The completion of T shall declare a tagged type. Such an attribute reference shall occur in the same library unit as the incomplete_type_declaration.

>> ------------------

But why was it allowed ?

There was no way to declare an incomplete tagged?

So why to allow reference to a Class attribute in such circumstances?

This seems weird… to allow to make reference to a class attribute of a none-tagged type (even if the complete view is tagged).

Was this a design error or were there some reasons or is there something I did not understand?

*From: Adam Beneschan <adam@irvine.com>*
*Date: Mon, 3 Aug 2009 08:26:24 -0700 PDT*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

> But why was it allowed ?

There was no way to declare an incomplete tagged ?

Correct. This just wasn't in the syntax.

> So why to allow reference to a Class attribute in such circumstances ?

You could do that if you knew that the type was going to be declared as tagged when you got around to declaring the full type.

Keep in mind that this isn't a privacy issue. The main reason for incomplete types is so that you can declare linked lists and things like that. Allowing a 'Class reference to a type not yet declared as tagged allowed things like this:

```
type Rec;
type Rec_Acc is access all Rec'Class;
type Rec is tagged record
   …
   Next : Rec_Acc;
end record;
```

so that you could have a heterogeneous list of records.

> This seems weird … to allow to make reference to a class attribute of a none-tagged type (even if the complete view is tagged)

It doesn't seem so weird to me; from a programmer's point of view, the main purpose of incomplete types is so that you can define and refer to the type's name ahead of time; therefore, there really isn't any reason (for a programmer) to look at the type as an untagged type at all. It's a little different where private types are concerned, because there are two views of the type---the partial view, which is for packages that can't see inside the private part and thus can't see the full declaration of the type; and the full view, for places that *can* see the private part. It's possible for the partial view to be untagged and the full view to be tagged, so allowing 'Class in a place where only the partial view is visible would be wrong. But the "two views" doesn't really apply to incomplete types. (From the compiler's standpoint, it does; but I'm talking from a programmer's standpoint here.)

There is one exception to all of the above, and that's the case where an incomplete type in the private part of a package is completed in the package *body*. Not having "tagged incomplete" types prevented certain useful uses of the type, so the "is tagged;" syntax was added. I think that being able to use this in my Rec example above, so that you can now declare an incomplete type as tagged before applying 'Class, was more of a side benefit. (AI95-326)

> Was this a design error or were there some reasons or is there something I did not understand ?

You could call it a small design error, but only a small one since the cases where it made a difference were pretty obscure.

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Mon, 3 Aug 2009 20:45:02 +0200*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

> […] The main reason for incomplete types is so that you can declare linked lists and things like that. […]

Well, I would argue that logically the above could be rather:

```
type Rec;
type Rec_Acc is access all Rec;
type Rec'Root is tagged record
   …
   Next : Rec_Acc;
end record;
```

Here **'Root** is defined so that **T'Class'Root = T**. So Rec is class-wide.

Of course, from the stand point that any type has a class rooted in it, Rec'Class is fully OK. But if we wanted further to pretend that some types are more "classy" than others, we could maintain such distinction in a way like above.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Wed, 12 Aug 2009 20:50:47 -0500*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

> […] in Ada 95, this was allowed to use the Class attribute with a prefix which was an incomplete type.... even not tagged.

Gee, guys, that's pretty boring. If you want something mind blowing in this area, look into 7.3.1(8-9)

(http://www.adaic.org/standards/05rm/html/RM-7-3-1.html).

More than 90% of the ARG (that is, everyone other than Tucker, and he only had a vague recollection of it) was unaware of this rule when I stumbled across it updating notes. And no one can remember why we would want such a rule - it seems completely privacy breaking, and for no good reason. But there is an ACATS test for it, so it probably will work in your favorite compiler…and thus we were unwilling to take it out (somebody probably depends on it, and forcing a switch to a tagged private type would have other consequences making it not necessarily a trivial work-around).

A quick example:

```
package P is
  type Priv is private;
  type Acc is access all Priv'Class;
  - - Legal!!!!!!
private
  type Priv is tagged null record;
end P;
```

*From: Niklas Holsti <niklas.holsti@tidorum.fi>*
*Date: Thu, 13 Aug 2009 11:18:47 +0300*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

[…]

I don't understand -- 7.3.1(9) says (both in RM 95 and RM 05) that the 'Class attribute is "allowed only from the beginning of the private part in which the full view is declared…"

> A quick example:

>

> package P is

>   type Priv is private;

>   type Acc is access all Priv'Class; -- Legal!!!!!!

> private

>   type Priv is tagged null record;

> end P;

This conflicts with 7.3.1(9) because Priv'Class is used before the private part. GNAT (Debian Lenny) says:

p.ads:3:27: tagged type required, found private type "Priv" defined at line 2

If the declaration of type Acc is moved below "private", GNAT accepts the code, as it should by 7.3.1(8-9).

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Thu, 13 Aug 2009 09:16:29 -0400*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

[…]

I agree with Niklas. The above is illegal.

The purpose of the rule is so you can create recursive types, as in:

```
package P is
  type Priv is private;
private
  type Acc is access all Priv'Class;
      - - Legal!!!!!!
  type Priv is tagged
      record
         Next : Acc;
         ... - -etc.
      end record;
end P;
```

without exposing the taggedness to clients. Note that Acc is in the private part, so there's no privacy-breaking going on.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Thu, 13 Aug 2009 23:21:39 -0500*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

[…]

Another example of why you ought to come to meetings more often. I don't recall anyone figuring that out per-se (although it is obvious that you can use it in this way). All this special rule eliminates is the need to declare an incomplete type in this one unusual case (the private type in cases like this is best declared as tagged, so this shouldn't happen very often). Seems like a pile of work for an implementation and for the language just to eliminate one line in a private part.

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Fri, 14 Aug 2009 15:01:15 -0400*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

[…]

> Seems like a pile of work for an implementation and for the language just to eliminate one line in a private part.

The problem is that declaring that incomplete type is illegal. Eliminating the need for one _illegal_ line isn't such a bad thing. ;-)

```
package P is
  type Priv is private;
private
  type Priv is tagged; - - Illegal!
  type Acc is access all Priv'Class;
  type Priv is tagged
    record
      Next : Acc;
    end record;
end P;
```

We could have made it legal, I suppose. But having three views of a type might make more trouble for the compiler than the existing rules.

On the third hand, compilers have to deal with:

```
package P is
  type Priv is private;
private
  task type Priv is ...;
end P;

package body P is
  task body Priv is separate;
end P;

separate (P)
task body Priv is ...;
```

where there are four things called Priv, which are all really the same thing.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Fri, 14 Aug 2009 18:41:22 -0500*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

> The problem is that declaring that incomplete type is illegal.

> Eliminating the need for one _illegal_ line isn't such a bad thing.

Why is it illegal? Doesn't seem like it ought to be on the face of it.

> We could have made it legal, I suppose. But having three views of a type might make more trouble for the compiler than the existing rules.

Apparently, you haven't thought too hard about the semantics of limited views. :-) *Every* private type that can be used in a limited view has (at least) three views (the incomplete view from the limited view being the third).

I believe we (Tucker and I? On the ARG list? I don't remember) were discussing eliminating some of the restrictions on the declaration and completion of incomplete types in order to make the use of "incomplete instances" or "integrated packages" or whatever we end up with more useful.

(Many of the "derivation" problems go away if you can simply declare an incomplete type ahead of time to introduce the proper name for the type.)

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Sat, 15 Aug 2009 10:33:43 -0400*
*Subject: Re: Ada 95 and Class attribute for none-tagged incomplete type*
*Newsgroups: comp.lang.ada*

> Why is it illegal? Doesn't seem like it ought to be on the face of it.

Umm... Because the RM says so? 7.3(4). So why does the RM say so?

Because Jean Ichbiah wanted it that way, I guess, and nobody thought it desirable or important enough to change during Ada 9X or Ada 0X projects.

> Apparently, you haven't thought to hard about the semantics of limited views. :-) *Every* private type that can be used in a limited view has (at least) three views (the incomplete view from the limited view being the third).

Good point, although at the time the rule we're talking about (allowing Private_Type'Class before the full type) was invented, there were no such things as limited views.

> I believe we (Tucker and I? On the ARG list? I don't remember)

We discussed it in a telephone meeting. No doubt also on ARG list.

Yes, I think that's a good direction to go, if we can make it work. A language design goal should be: Never require the use of pointers, except when reference semantics is desired. Ada violates this principle in several ways, and removing one of them can be a good thing.

(C violates this principle even more, and Java more still.)

## On the serialization of enumeration types

*From: xorque <xorquewasp@googlemail.com>*
*Date: Wed, 26 Aug 2009 03:22:08 -0700 PDT*
*Subject: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

I'm designing a package that uses a lot of similar but distinct enumeration types.

At some point, those types need to be encoded to be sent over the wire. The encoding rules are simple:

The enumeration values are converted to unsigned 32 bit integers with the first value as 0 and increasing sequentially with each new value. The 32 bit value is packed into big-endian byte order.

The problem is: With so many enumeration types, I now have about 300 lines of almost identical procedures (to be used as stream attributes) that just call a procedure that packs Unsigned_32 values into Storage_Element arrays.

Is there some clever way I can just write ONE 'Write attribute procedure and ONE 'Read attribute procedure and have all the defined enumeration types use those?

Freezing rules prevented me from writing a generic Packed_Enumeration_IO package ("representation item appears too late").

*From: Oliver Kellogg <okellogg@freenet.de>*
*Date: Wed, 26 Aug 2009 04:00:52 -0700 PDT*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

If your enums are represented as you describe then why do you need representations?

You could rely on the natural Ada representation (which happens to conform with your encoding rules as far as the enum values are concerned).

*From: xorque <xorquewasp@googlemail.com>*
*Date: Wed, 26 Aug 2009 04:33:37 -0700 PDT*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…] Where in the (2005) RM does it state that enumerations are encoded at big-endian 32 bit integers?

*From: Oliver Kellogg <okellogg@freenet.de>*
*Date: Wed, 26 Aug 2009 05:03:55 -0700 PDT*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

Perhaps I am misunderstanding - I thought along following lines:

```
- - file:
  enum_to_big_endian_unsigned32.ads
with Interfaces;
generic
  type Enum_Type is (<>);
function
  Enum_To_Big_Endian_Unsigned32
    (Value : Enum_Type)
    return Interfaces.Unsigned_32;

- - file:
  enum_to_big_endian_unsigned32.adb
function
  Enum_To_Big_Endian_Unsigned32
    (Value : Enum_Type)
  return Interfaces.Unsigned_32 is
    function htonl
      (host_int : Interfaces.Unsigned_32)
    return Interfaces.Unsigned_32;
    pragma Import (C, htonl, "htonl");
begin
  return htonl (
     Enum_Type'Pos (Value));
end
Enum_To_Big_Endian_Unsigned32;

- - file: enum2u32_test.adb
- - Test Main Program
with Interfaces;
with
Enum_To_Big_Endian_Unsigned32;

procedure Enum2U32_Test is

  type Color_T is (Red, Green, Blue);

  function To_U32 is new
    Enum_To_Big_Endian_Unsigned32
      (Color_T);

  Result : Interfaces.Unsigned_32 :=
    To_U32 (Green);

begin
  - - application code …
  null;
end;
```

*From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>*
*Date: Wed, 26 Aug 2009 14:05:19 +0200*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

> […]

> The enumeration values are converted to unsigned 32 bit integers with the first value as 0 and increasing sequentially with each new value. The 32 bit value is packed into big-endian byte order.

We are using a chain code for same purpose. It is better for shorter values and is not limited from above.

> The problem is: With so many enumeration types, I now have about 300 lines of almost identical procedures (to be used as stream attributes) that just call a procedure that packs Unsigned_32 values into Storage_Element arrays.

You can read/write Enumeration_Type'Pos (Value), instead, which is how we deal with this. The same will work with other discrete types.

> Is there some clever way I can just write ONE 'Write attribute procedure and ONE 'Read attribute procedure and have all the defined enumeration types use those?

No, because there is no common ancestor for these types. Another technique we are using is that there are "comm" values of tagged types rooted in the same abstract parent, while Get and Set operations are provided to get/set a type-specific non-tagged value from/to. But in this case, obviously, Get and Set must be implemented for each specific type new. Generic packages are used for that, which then leads to a geometric explosion of instances and an utter mess, as always when you deal with generics.

> Freezing rules prevented me from writing a generic

> Packed_Enumeration_IO

> package ("representation item appears too late").

I don't understand why. It works to us, however as I said above, a generic solution is always bad, unless there is no any alternative. So far Ada does not provide alternatives (like ad-hoc supertypes and interfaces etc).

*From: Jeffrey R. Carter <jrcarter@acm.org>*
*Date: Wed, 26 Aug 2009 12:44:11 -0700*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

'Pos for enumeration types gives values that satisfy this rule. They are universal_integer, and so may be used anywhere a value of your unsigned 32 bit integer type is needed.

I don't know if this helps you, however.

*From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>*
*Date: Wed, 26 Aug 2009 13:17:46 +0200*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

Not exactly the same solution, but there doesn't seem to be a problem with a generic and Seuqential_IO; I might well have misunderstood.

There is a #Gem at AdaCore's web site explaining how to use different representations for types derived from e.g. enum types, if that is any help.

```
package Enums is

  type Color is (Red, Green, Blue);
  type Smell is (Good, Neutral, Bad);
private
  for Smell use (Good => 14,
      Neutral => 23, Bad => 100);
  for Smell'size use 32;
  for Color'size use 32;
end Enums;

generic
  type E is (<>);
procedure EG(extension : String);

with Ada.Sequential_IO;
with Interfaces;  use Interfaces;
with Ada.Unchecked_Conversion;

procedure EG(extension : String) is
  package My_IO is new
```

```
      Ada.Sequential_IO(Unsigned_32);
   function To_Unsigned_32 is new
      Ada.Unchecked_Conversion
         (Source => E,
          Target => Unsigned_32);
   F : My_IO.File_Type;
begin
   My_IO.Create(F, My_IO.Out_File,
                "run." & extension);
   for K in E loop
      My_IO.Write(F, Item =>
         To_Unsigned_32 (K));
   end loop;
end EG;

with Enums, EG;
procedure Run is
   use Enums;
   procedure Color_Out is new
      EG(Color);
   procedure Smell_Out is new
      EG(Smell);
begin
   Color_Out("rgb");
   Smell_Out("snf");
end Run;
```

*From: xorque <xorquewasp@googlemail.com>*
*Date: Wed, 26 Aug 2009 04:35:17 -0700 PDT*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

Yes, I've seen that one (love the Gems series) but as I'm not just writing enum values (integers, booleans, strings, etc), I need an IO package that can handle heterogeneous data.

We're basically sending data to GNAT.Socket streams.

*From: Simon J. Wright <simon.j.wright@mac.com>*
*Date: Wed, 26 Aug 2009 22:05:24 -0700 PDT*
*Subject: Re: Generalized serialization for enumeration types*
*Newsgroups: comp.lang.ada*

[…]

This _compiles_:

```
with Ada.Streams;
generic
   type E is (<>);
procedure Xorque_Writer_G (
   Stream : access
      Ada.Streams.Root_Stream
      _Type'Class;
   V : E);
procedure Xorque_Writer_G (
   Stream : access
      Ada.Streams.Root_Stream
      _Type'Class;
   V : E) is
begin
   null;
end Xorque_Writer_G;


with Xorque_Writer_G;
package Xorque is
   type E_Base is (A, B, C);
   procedure E_Base_Writer is new
      Xorque_Writer_G (E_Base);
   type E is new E_Base;
   for E'Write use E_Base_Writer;
end Xorque;
```

## On the visibility of private part in child packages

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Wed, 3 Jun 2009 22:45:43 -0700 PDT*
*Subject: Indirect visibility of private part in child packages*
*Newsgroups: comp.lang.ada*

[…]

I've meet a tortuous question with a set of packages I'm converting from Pascal (as I use Ada now, I'm taking the opportunity to re-create a nicer design).

The question I'm facing deals with visibility of private part in child package. More precisely, an indirect visibility of a private part.

An example will better show the case:

This can be reduced to three package (specs and bodies).

The root package:

```
package P1 is
   type T1_Type is tagged private;
   - - Some primitives on T1_Type are
      specified here …
private
   type T1_Type is tagged record
      Low_Level_Data : Some_Type;
   end record; - - T1_Type
end P1;
```

The first child package (with this one, every thing's Ok):

```
package P1.P2 is
   type T2_Type is new T1_Type with
private;
   - - Some primitives on T2_Type
      are specified here …
   procedure A_Primitive (
      Item : out T2_Type);
private
   type T2_Type is new T1_Type with
null record;
end P1.P2;

package body P1.P2 is
   procedure A_Primitive (
      Item : out T2_Type) is
   begin
      Item.Low_Level_Data := …;
      - - It works, beceause P1.P2 is a
      - - child package with a view on
      - - P1's private part
   end A_Primitive;
end P1.P2;
```

The second child package (the one which turns into a subtle doubt):

```
with P1.P2;
package P1.P3 is
   type T3_Type is new P1.P2.T2_Type
      with private;
      - - Some primitives on T2_Type are
         specified here …
   procedure A_Primitive (
      Item : out T3_Type);
private
   type T3_Type is new P1.P2.T2_Type
     with null record;
end P1.P3;

package body P1.P3 is
   procedure A_Primitive (
      Item : out T3_Type) is
   begin
      Item.Low_Level_Data := …;
      - - It fails : the compiler complains
      - - there is no
      - - selector "Low_Level_Data"
      - - defined for T3_Type
   end A_Primitive;
end P1.P3;
```

As said in the last comment, it seems P1.P3 cannot see the T1_Type.Low_Level_Data member. But as P3 is a child package of P1, it has a view on P1's private part, and there is indeed no trouble about it with P2. P3 know P2.T2_Type is derived from P1.T1_Type, and P3 has a view on T1_Type privates. But it seems the compiler does not care about it.

There may be interpretation matter here : does the private specification, means "disallow access to private part" or does it means "do not provide an access to the private part". If it means "disallow access to private part", then it ill explain why P3 cannot access T1_Type.Low_Level_Data member. But if it means "do no provide an access to private" part, then P3 should still be able to access T1.Low_Level_Data, because it has an access to this as a child package of P1.

What does long experienced lawyers think about it ?

I've tried to look in section 8 of the "RM 2005", but either the answer was not there or I missed it.

Note #1: I hope my example is clear enough. If it is not, please, tell me, so that I can attempt to reword it.

Note #2: I do not want to make P3 a child package of P2, because it does not need to access any internals of T2_Type. I want to have some types, extending each others, and doing so only relying on either public interface or on the sole common private part defined in P1.

*From: Ludovic Brenta <ludovic@ludovic-brenta.org>*
*Date: Thu, 4 Jun 2009 00:50:10 -0700 PDT*
*Subject: Re: Indirect visibility of private part in child packages*
*Newsgroups: comp.lang.ada*

Maybe you can solve your problem like this:

```
package body P1.P3 is
  procedure A_Primitive
     (Item : out T3_Type) is
  begin
    T1_Type (Item).Low_Level_Data :=
       ...;
  end A_Primitive;
end P1.P3;
```

I am undecided whether the compiler is correct or wrong about the visibility of Low_Level_Data. On the one hand, P1.P3 cannot see the full declaration of T2_Type; on the other hand, it can see that T2_Type publicly inherits from T1_Type and can also see the full declaration of T1_Type.

It seems that the public (partial) view of T2_Type hides the full declaration of T1_Type. So, I think my solution would work by removing T2_Type out of the way.

Another solution, which enforces encapsulation better, would be:

```
package P1 is
  type T1_Type is tagged private;
private
  type Some_Type is …;
  type T1_Type is tagged record
    Low_Level_Data : Some_Type;
  end record;
  procedure Set (Item : out T1_Type;
                 Data : in Some_Type);
end P1;

package body P1.P3 is
  procedure A_Primitive
    (Item : out T3_Type) is
  begin
    Set (Item, Data => …); -- OK, calls
        -- inherited primitive operation
  end A_Primitive;
end P1.P3;
```

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Thu, 4 Jun 2009 04:47:26 -0700 PDT*
*Subject: Re: Indirect visibility of private part in child packages*
*Newsgroups: comp.lang.ada*

> Maybe you can solve your problem like this:

> package body P1.P3 is

>   procedure A_Primitive (Item : out T3_Type) is

>   begin

>     T1_Type (Item).Low_Level_Data :=3D ...;

>   end A_Primitive;

> end P1.P3;

I've just tried it, and it does not work better. The compiler complains there are not selector Low_Level_Data for T1_Type.

> I am undecided whether the compiler is correct or wrong about the visibility of Low_Level_Data. […]

This is exactly the same think in my mind. Perhaps the semantic is ambiguous here.

> It seems that the public (partial) view of T2_Type hides the full declaration of T1_Type. So, I think my solution would work by removing T2_Type out of the way.

T2_Type is required ;) The purpose is to have T2_Type and T3_Type, and may be later some others, beside of each other.

> Another solution, which enforces encapsulation better, would be:

> […]

I will try it soon and tell about the result. But at first sight, I think that the compiler is going to complain that Set is a primitive of T1_Type and thus that it must be defined in the public part. Furthermore, I think Set would have to be redefined for each T2_Type, T3_Type and etc.

[…]

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Thu, 04 Jun 2009 09:22:05 -0400*
*Subject: Re: Indirect visibility of private part in child packages*
*Newsgroups: comp.lang.ada*

> […]

> I've just tried it, and it does not work better. The compiler complains there are not selector Low_Level_Data for T1_Type.

Are you sure? Converting to T1_Type should work. P3 can see that T1_Type has a Low_Level_Data component.

The reason your original example didn't work is that the components of T2_Type are "nailed down" at the point where it's declared, and P3 can't see that. It can, however, see that it's derived from T1_Type, so converting should work.

> […] Furthermore, I think Set would have to be redefined for each T2_Type, T3_Type and etc.

You could make Set class-wide.

*From: Adam Beneschan <adam@irvine.com>*
*Date: Thu, 4 Jun 2009 09:06:28 -0700 PDT*
*Subject: Re: Indirect visibility of private part in child packages*
*Newsgroups: comp.lang.ada*

[…]

Here's why it doesn't work:

The "operation" of providing a selector Low_Level_Data for T1_Type is made visible in the private part of P1.

When T2_Type is declared as derived from T1_Type, the operation of providing the selector Low_Level_Data is inherited from T1_Type, but it only becomes visible where the original operation becomes visible. Since the original operation is visible in the private part of P1, the inherited operation becomes visible in the private part of P1.P2, which is the first place that the private part of P1 is visible.

(It's not relevant that T2_Type is a private extension; I believe it would work the same if T2_Type were fully declared in the visible part.)

When T3_Type is declared as derived as T2_Type, the operation of providing the selector Low_Level_Data is inherited from T2_Type, but it only becomes visible where the original operation (on T2_Type!!) becomes visible, which is… nowhere, because there isn't anywhere in P1.P3 that is able to access P1.P2's private part (and according to the above paragraph, that's where the operation on T2_Type is made visible---in P1.P2's private part). That's why Low_Level_Data isn't available for T3_Type.

So the compiler is correct to reject your original program. However, I think Ludovic's fix (using a type conversion, actually a view conversion) should work, and it's a compiler bug if that is rejected.

One could argue that this rule is anomalous, and that if a type is visibly derived from a grandparent (or great-grandparent, etc.) type in a situation like this, then operations that are indirectly inherited from the grandparent (etc.) type should be visible if they're visible for the grandparent, even if the inheritance comes indirectly through a non-visible part of some other package. There could be a privacy problem if the operation could be overridden in the non-visible part, but that isn't possible here---there's nothing P1.P2 can do to change the meaning of the Low_Level_Data selector. This issue is related to AI05-125, but although a couple of us think there's a flaw in the language, others believe that it can't be fixed without making things even more confusing. In any event, it's not all that important here since the view conversion is an adequate workaround, assuming your compiler isn't broken.

> I've tried to look in section 8 of the "RM 2005", but either the answer was not there or else, I missed it.

I think 7.3.1 and particularly 7.3.1(4) is the important rule here, although I have to admit that that section uses a lot of slightly fuzzy generalities that I've had problems interpreting occasionally.

## Randomness tests in Ada

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Thu, 16 Jul 2009 12:41:59 -0700 PDT*
*Subject: Randomness tests*
*Newsgroups: comp.lang.ada*

Does someone have some randomness checking sources in Ada?

I am evaluating a fast random generator (published long time ago in the Ada letters), alternative to GNAT's implementation.

Probably the most obvious test is to check that it is well uniformly distributed.

*From: jonathan <johnscpg@googlemail.com>*
*Date: Fri, 17 Jul 2009 15:25:56 -0700 PDT*
*Subject: Re: Randomness tests*
*Newsgroups: comp.lang.ada*

[…]

You can find several basic tests at

http://web.am.qub.ac.uk/users/j.parker/miscellany

in the subdirectory: disorderly

see for example:

gcd_4bytes_1.adb

gcd_6bytes_1.adb

rank_tst_2.adb

bday_tst_1.adb

gorilla_tst_demo_2.adb

(To use gcd_6bytes_1.adb, concatenate 2 of the 24 bit words generated by Marsaglia's Universal generator.)

The greatest-common-divisor test, gcd_6bytes_1.adb and gorilla tests are from the Marsaglia, Tsang updated diehard suite. To find their paper google for "Some difficult-to-pass tests of randomness", Marsaglia, Tsang.

The birthday test is a variant of Marsaglia's birthday test. I include the birthday test (bday_tst_1.adb) because the present GNAT generator fails this. (It has to. All short period generators fail this test.)

I include Marsaglia's rank test (rank_tst_2.adb) because it easily breaks the Mersenne Twister.

(It has to. The Mersenne Twister is full-period and linear; they all fail this test catastrophically.) I would still prefer the Mersenne Twister over the Marsaglia Universal generator though .. it does quite a bit more bit-mixing between outputs than the Marsaglia Universal generator.

The site given above contains my own version of the way I think Random Number Generators ought to be done: disorderly.ads. Generator Disorderly is really just a non-linear 61-bit version of Marsaglia's KISS generator. KISS appeared about a decade after his Universal generator. The KISS generator is a combination generator that uses linear component generators. Disorderly includes a non-linear component generator: X_{n+1} =3D X_n**2 mod M.

The X_{n+1} =3D X_n**2 mod M component in Disorderly was inspired by GNAT's generator. The GNAT generator is a combination generator made from 2 of these non-linear component generators.

(If you use just the least significant bit of each word output by the GNAT generator, then its called the Blum-Blum-Schub generator.)

A fast linear version of Disorderly is included; seemed to be about 3 times faster than the GNAT generator .. produces 61 bits per call.

Here is the start of the README:

1. directory Disorderly contains

- a package of new Random Number Generators (package Disorderly.Random) along with some test/demo routines.
- a package of Random Deviates (package Disorderly.Random.Deviates) with the following distributions:

  Uniform, Normal (Gaussian), Exponential, Lorentzian (Cauchy), Poissonian, Binomial, Negative Binomial, Weibull, Rayleigh, Student_t, Beta, Gamma, Chi_Squared, Log_Normal, Multivariate_Normal.

  procedure Deviates_Demo_1 tests and demonstrates usage of random deviates (variates).

The procedure tests package Disorderly.Random.Deviates.

procedure Basic_Deviates_Demo_1 tests and demonstrates usage of random deviates: Disorderly.Basic_Rand.Deviates Uses Disorderly.Basic_Rand and demonstrates usage of random deviates (variates).

*From: Cristoph Grein <christoph.grein@eurocopter.com>*
*Date: Fri, 17 Jul 2009 01:10:41 -0700 PDT*
*Subject: Re: Randomness tests*
*Newsgroups: comp.lang.ada*

Do you mean Marsaglias_Generator:

The algorithm was developed by George Marsaglia, Supercomputer Computations Research Institute, Florida State University

(ref. to Ada LETTERS, Volume VIII, Number2, March/April 1988).

I have an implementation of it. It produces the results required for a correct implementation.

*From: Cristoph.Grein <christoph.grein@eurocopter.com>*
*Date: Fri, 17 Jul 2009 12:05:52 -0700 PDT*
*Subject: Re: Randomness tests*
*Newsgroups: comp.lang.ada*

[…]

> Christoph, Gautier,
>
> did you look at the open source
>
> GNAT:ada.numerics.[float][discrete] random ?

Of course I know and use those. I just wanted to know which Ada Letters random generator Gautier wanted to test.

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Fri, 17 Jul 2009 13:41:03 -0700 PDT*
*Subject: Re: Randomness tests*
*Newsgroups: comp.lang.ada*

> Please let us know your conclusions.

So - the mentioned package is indeed the "universal" random generator from Dr. George Marsaglia, which seems to have appeared in the Ada

I've just "repimped" U_Rand with generics, no more global variables but a type Generator, and some Ada 95 A.N.Float_Random-style "renames", which facilitates the switch between random packages, like in this example:

```
 -- *** Choice of the floating-point type
used for the whole Portfolio Model:
 subtype Real is Long_Float;
 package Real_U_Rand is new
   U_Rand(Real);

 -- *** Choice of a random generator:
A.N.F_R, or U_Rand (faster), or...:

 package RRand renames
   -- Ada.Numerics.Float_Random;
  Real_U_Rand;
```

The funny thing is that RRand can be plugged in its turn into another generic package

```
package GRA is new
  Ada.Numerics.Generic_Real_Arrays
    (Real);

package RCopulas is new Copulas(
  Real,
  RRand.Uniformly_Distributed,
  RRand.Generator,
  RRand.Random,
  GRA
);
```

In case someone is interested, mail me.

Anytime soon I'll open a SourceForge project with various random variable goodies.

*From: Jeffrey R. Carter <spam.jrcarter.not@nospam.acm.org>*
*Date: Thu, 16 Jul 2009 21:41:37 GMT*
*Subject: Re: Randomness tests*
*Newsgroups: comp.lang.ada*

[…]

An interesting test is the "parking-lot" test. Consecutive pairs of numbers from the generator are treated as (x, y) pairs of coordinates and plotted on a graph.

Poor generators create line segments; good ones don't. (The test gets its name from the 1st generator it was applied to, which created short, parallel, diagonal line segments that reminded people of the lines in a parking lot.)

Apparently even generators that do well on other tests can fail this one.

The generator in Turbo Pascal did especially poorly on this test.

I know of one application in which consecutive triples of numbers were treated as (x, y, z) 3-D coordinates. A generator which did well on other tests, including the 2-D parking lot test, failed the 3-D version.

Also note that the PragmAda reusable components include the "universal" random number generator, which is supposed to be a very good generator, is portable, and gives identical results on all platforms.

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Thu, 16 Jul 2009 18:53:41 -0700 PDT*
*Subject: Re: Randomness tests*
*Newsgroups: comp.lang.ada*

[…]

Thanks for the explanations about the "parking-lot" method.

[…]

I was precisely considering that algorithm (the famous u_rand).

## Record type as generic parameter

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Thu, 13 Aug 2009 02:52:52 -0700 PDT*
*Subject: Record type as generic parameter*
*Newsgroups: comp.lang.ada*

[…]

Let's say I have an enumerated type called Scenario, and I want to fill some data structure with the following information: for each scenario, I have a certain number of items (of a defined type), which is varying according to the scenario. That information is stored in an array with pairs (scenario, item). An item might by appearing in several scenarios, it is why I don't have simply an array(Item) of Scenario. Everything fine up to now. Now, I would like to make it generic, since I have several sets of scenario - i.e. different enumerated types like Scenario. Then, my favourite Ada compiler doesn't want a record as generic parameter:

```
generic
  type Scenario is range <>;
  type Association is record
    scen: Scenario;
    it  : Item;
  end record;
  scenario_definition:
    array(Positive range <>) of
    Association;
  procedure Fill_Scenarios;
```

Any nice workaround?

A not-so-nice one is to have two arrays, one array of Scenario, one array of Item.

Ideally I would keep the type like Association above.

```
XY_scenario_definition:
  constant array(Positive range <>) of
    XY_Association:=
    (
      (Scen_abc, Item_56),
      (Scen_abc, Item_345),
      (Scen_uvw, Item_12),
    ..
```

The data are easier to pump from a database directly as Ada sources, they are then easier to read, and there is no chance of messing positions as with the two arrays solution.

*From: Jacob Sparre Andersen <sparre@nbi.dk>*
*Date: 13 Aug 2009 12:30:38 +0200*
*Subject: Re: Record type as generic parameter*
*Newsgroups: comp.lang.ada*

[…]

A private type. With functions for accessing the fields, if necessary.

*From: Gautier de Montmollin <gdemont@users.sourceforge.net>*
*Date: Thu, 13 Aug 2009 06:34:31 -0700 PDT*
*Subject: Re: Record type as generic parameter*
*Newsgroups: comp.lang.ada*

> Any reason why you cannot make a generic package that would *export*

> Association and Scenario_Definition

No!

> (and procedure Fill_Scenarios),

Yes!

> rather than importing them?

So first of all, thanks for the very nice idea!

The "Yes!" is because the exporting generic package is in (or withed by) a very low-level, satellite package, DB_Data.Geo, that basically consists in data pumped (once for all) from a database in form of big enumerated types and constant arrays. So it cannot "see" the definitions needed for Fill_Scenarios.

But the "No!" is valid as well, considering the types used for defining these scenarios.

So the solution is:

1) In the low-level package DB_Data.Geo.Links:

```
generic
  type Scenario is (<>);
package Scenario_framework is
  -- pairs:
  type Association is record
    scen: Scenario;
    it  : Item;
  end record;
  -- list of pairs:
  type Scenario_definition is
    array(Positive range <>) of
    Association;
end Scenario_framework;

package Dept_Xyz is new
  Scenario_framework
  (DB_Data.Geo.Dept_Xyz_Scenario);

  dept_xyz_scenario_definition:
    constant
     Dept_Xyz.Scenario_definition:=
     (
       (Scen_abc, Item_56),
       (Scen_abc, Item_345),
      (Scen_uvw, Item_12),
```

2) In the high-level package:

```
generic
   type Scenario is (<>);
   with package Framework is new
      Scenario_framework(Scenario);
   def: Framework.Scenario_definition;
 procedure Fill_Scenarios;
```

One instantiation is then:

```
procedure Fill_Dept_Xyz_Scenarios is
    new Fill_Scenarios(
        Dept_Xyz_Scenario,
        Dept_Xyz,
        dept_xyz_scenario_definition
      );
```

[…]

## Warnings when overriding Initialize and Finalize

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Fri, 10 Jul 2009 07:16:47 -0700 PDT*
*Subject: Got warnings when overriding Initialize and Finalize*
*Newsgroups: comp.lang.ada*

[…]

I got some warnings which still stick to me when I want to override Initialize and Finalize on controlled types.

Here is a reduced example of the matter:

```
with Ada.Finalization;
package Test is
  type T is tagged limited private;
private
  type T is new Ada.Finalization.
     Limited_Controlled with null record;
   overriding procedure Initialize
     (Object : in out T);
   overriding procedure Finalize
     (Object : in out T);
end Test;
```

This give me warnings like “warning: declaration of "Initialize" hides one at line xxx ” where xxx is the line of “ type T is new Ada.Finalization.Limited_Controlled with null record; ”. The same warning appears for Finalize.

If there is no more private part, there is no more warnings. Ex …

```
with Ada.Finalization;
  type T is new
     Ada.Finalization.Limited_Controlled
     with null record;

  overriding
  procedure Initialize (Object : in out T);
  overriding
  procedure Finalize (Object : in out T);
end Test;
```

… does not produce any warnings.

If I do the following, there is no warning as well :

```
with Ada.Finalization;
package Test is
  type T is new
     Ada.Finalization.Limited_Controlled
     with private;

  overriding
  procedure Initialize (Object : in out T);
  overriding
  procedure Finalize (Object : in out T);
  private
   type T is new
     Ada.Finalization.Limited_Controlled
     with null record;
end Test;
```

Is it mandatory to declare overriding of Initialize and Finalize in the public part and thus to declare the T is an Ada.Finalization.Limited_Controlled in the public part as well ?

But the Annotated RM contains this small excerpt :

AARM 7.6 17.h.2/1 says:

```
package P is
  type Dyn_String is private;
  Null_String : constant Dyn_String;
  …
  private
   type Dyn_String is new
     Ada.Finalization.Controlled with …
   procedure Finalize
     (X : in out Dyn_String);
   procedure Adjust
     (X : in out Dyn_String);

   Null_String : constant
     Dyn_String :=
     (Ada.Finalization.Controlled
      with …);
  …
end P;
```

So what to think about these warnings?

Is there something I am not seeing?

*From: Adam Beneschan <adam@irvine.com>*
*Date: Fri, 10 Jul 2009 08:34:17 -0700 PDT*
*Subject: Re: Got warnings when overriding Initialize and Finalize*
*Newsgroups: comp.lang.ada*

[…]

This looks like a compiler glitch to me.

There is a case, involving *untagged* types, where overriding an operation in the private part can lead to some unexpected results:

```
package Pak2 is
  type T2 is new Pak1.T;
   - - inherits operation Op
  private
   overriding procedure Op (X : T2);
end Pak2;
```

Now, calling Op on an object of type T2 may give you either the inherited one or the overriding one, depending on whether the private part of Pak2 is visible at that point. It may be that the compiler, with this case in mind, displays a warning any time there's an override in the private part of a package; but it seems like it's going overboard in this case. The compiler needs to tailor its warnings a little better. That's just my guess as to why you're seeing the warnings. But there isn't anything wrong with your original code.

*From: Yannick Duchêne <yannick_duchene@yahoo.fr>*
*Date: Fri, 10 Jul 2009 09:12:04 -0700 PDT*
*Subject: Re: Got warnings when overriding Initialize and Finalize*
*Newsgroups: comp.lang.ada*

> Now, calling Op on an object of type T2 may give you either the inherited one or the overriding one, depending on whether the private part of Pak2 is visible at that point.

Indeed, this example is a bad practice.

> It may be that the compiler, with this case in mind, displays a warning any time there's an override in the private part of a package

… and it does not take care it is a tagged type so

Your idea of the reason why is clever.

*From: Randy Brukardt <randy@rrsoftware.com>*
*Date: Fri, 10 Jul 2009 20:10:25 -0500*
*Subject: Re: Got warnings when overriding Initialize and Finalize*
*Newsgroups: comp.lang.ada*

[…]

For the record, I agree with Adam. Claw is full of overridings like this; our rule was that overridings should be done in the private part as there is no reason for the client to know whether the routines are overridden or just inherited. Looks like a bogus warning to me (and Adam's explanation makes sense to me, too).

*From: Robert A Duff <bobduff@shell01.TheWorld.com>*
*Date: Tue, 14 Jul 2009 10:54:53 -0400*
*Subject: Re: Got warnings when overriding Initialize and Finalize*
*Newsgroups: comp.lang.ada*

[…]

For what it's worth, the latest version of GNAT Pro does not give these bogus warnings. I don't know if the fix has made it into public version(s) yet.

# Conference Calendar

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

The information in this section is extracted from the on-line *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.

## 2009

October 04-09 ACM/IEEE 12th **International Conference on Model Driven Engineering Languages and Systems** (MoDELS'2009), Denver, Colorado, USA. Topics include: Development of domain-specific modeling languages, 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 current model-based engineering standards, Experience with model-based engineering tools, etc.

☺ October 06 2nd **International Workshop on Model Based Architecting and Construction of Embedded Systems** (ACES-MB'2009). Topics include: model-oriented counterparts of specific design and implementation 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 05-06 2nd **International Conference on Software Language Engineering** (SLE'2009), Denver, Colorado, USA. Topics include: the engineering of artificial languages used in software development including general-purpose programming languages, domain-specific languages, modeling and meta-modeling languages, data models, and ontologies.

October 07-09 14th **International Real-Time Ada Workshop** (IRTAW'2009), Portovenere, Italy. In cooperation with Ada-Europe.

☺ October 11 5th **Workshop on Programming Languages and Operating Systems** (PLOS'2009), Big Sky, MT, USA. Topics include: critical evaluations of new programming language ideas in support of OS construction; type-safe languages for operating systems; language-based approaches to crosscutting system concerns, such as security and run-time performance; language support for system verification; the use of OS abstractions and techniques in language runtimes; etc.

☺ October 12-14 IMCSIT2009 - 2nd **Workshop on Advances in Programming Languages** (WAPL'2009), Mragowo, Poland. Topics include: Compiling techniques; Domain-specific languages; Formal semantics and syntax; Generative and generic programming; Languages and tools for trustworthy computing; Language concepts, design and implementation; Metamodeling and modeling languages; Model-driven engineering languages and systems; Practical experiences with programming languages; Program analysis, optimization and verification; Program generation and transformation; Programming tools and environments; Proof theory for programs; Specification languages; Type systems; etc.

October 13-16 16th **Working Conference on Reverse Engineering** (WCRE'2009), Lille, France. 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.

October 15-16 3rd **International Symposium on Empirical Software Engineering and Measurement** (ESEM'2009), Lake Buena Vista, Florida, USA. Topics include: Reports on the benefits derived from using certain technologies; Empirically-based decision making; Industrial experience in process improvement; Quality measurement and assurance; Evidence-based software engineering; Effort and cost estimation, defect rate and reliability prediction; etc.

☺ October 25-29 — 24th Annual **Conference on Object-Oriented Programming, Systems, Languages, and Applications** (OOPSLA'2009), Orlando, Florida, USA. Topics include: the intersection between programming languages and software engineering; key programming models and programming methods and related software engineering ideas, technologies, tools, and applications; critical evaluation of accepted practices, proposals for new programming models, exploration and extension of well-established models, and other novel approaches to building systems; etc.

- ☺ October 25 — 8th **"Killer Examples" workshop**. Topics include: examples that expose bad practice and so lead to better appreciation of good practice, as obtained by following sound object-oriented principles.

- October 25 — 1st **Workshop on Evaluation and Usability of Programming Languages and Tools** (PLATEAU'2009). 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, 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 25-29 — **Onward! 09**. Topics include: different ways of thinking about, approaching, and reporting on programming languages and software engineering research.

- October 26 — 3rd **Workshop on Assessment of Contemporary Modularization Techniques** (ACoM.09). Topics include: Lessons learned from assessing new modularization techniques; Empirical studies and industrial experiences; Comparative studies between new modularization techniques and conventional ones; etc.

♦ Nov 01-05 — ACM Annual **International Conference on Ada and Related Technologies** (SIGAda'2009), St. Petersburg, Tampa Bay area, Florida, USA. Sponsored by ACM SIGAda, in cooperation with SIGCAS, SIGCSE, SIGPLAN, Ada-Europe, and Ada Resource Association.

☺ Nov 02-03 — 14th **International Workshop on Formal Methods for Industrial Critical Systems** (FMICS'2009), Eindhoven, the Netherlands. Topics include: Design, specification, code generation and testing based on formal methods; Verification and validation methods that address shortcomings of existing methods with respect to their industrial applicability; Tools for the development of formal design descriptions; Case studies and experience reports on industrial applications of formal methods, focusing on lessons learned or identification of new research directions; Impact of the adoption of formal methods on the development process and associated costs; Application of formal methods in standardization and industrial forums; etc.

November 02-07 — 16th **International Symposium on Formal Methods** (FM'2009), Eindhoven, the Netherlands. Theme: "Theory meets practice". Topics include: every aspect of the development and application of formal methods for the improvement of the current practice on system developments; of particular interest are papers on tools and industrial applications; etc.

- November 02 — **Workshop on Verified Software: Theories, Tools, and Experiments** (VSTTE'2009). Topics include: Specification and verification techniques, Tool support for specification languages, Automation in formal verification, etc.

- November 04 — 8th **International Workshop on Parallel and Distributed Methods in Verification** (PDMC'2009). Topics include: multi-core/distributed model checking, slicing and distributing the state space, parallel/distributed theorem proving and constraints solving, tools and case studies, industrial applications, etc.

- November 06 — 2nd **International Conference on Teaching Formal Methods** (TFM'2009). Topics include: experiences of teaching FMs, both successful and unsuccessful; educational resources including the use of books, case studies and the internet; the advantages of FM-trained graduates in the workplace; changing attitudes towards FMs in students, academic staff and practitioners; etc.

| | |
|---|---|
| November 06 | NIST SAMATE **Static Analysis Tool Exposition Workshop** (SATE'2009), Washington, D.C., USA. Topics include: Contribution of static analysis to software security assurance; Integration of, or tradeoffs between, static and dynamic analysis; Issues in scaling static analysis to deal with large systems; Flaw catching vs. sound analysis; Benchmarks or reference datasets; Formal descriptions of weaknesses and vulnerabilities; User experience drawing useful lessons or comparisons; Synergies of pre- and post-production assurance; Case studies on real applications; etc. Deadline for submissions: October 2, 2009 (papers) |
| November 16-19 | 20th **International Symposium on Software Reliability Engineering** (ISSRE'2009), Mysuru-Bengaluru, India. Topics include: Reliability, availability, and safety of software systems; Validation, verification, and testing; Software quality and productivity; Software security; Fault tolerance, survivability, and resilience of software systems; Open source software reliability engineering; Supporting tools and automation; Industry best practices; etc.; Empirical studies of any of the above topics. |
| ☺ Nov 23-27 | 7th IEEE **International Conference on Software Engineering and Formal Methods** (SEFM'2009), Hanoi, Vietnam. Topics include: formal methods technology transfer; software specification, verification and validation; component-based development; programming languages and type theory; program analysis; real-time, hybrid and embedded systems; safety-critical and fault-tolerant systems; software architectures and their description languages; light-weight formal methods; CASE tools and tool integration; applications of formal methods and industrial case studies; etc. |
| December 04-12 | 5th **International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering** (CISSE'2009), Internet on-line event. Includes 4 e-conferences. Topics include: Programming Models and tools, Parallel and Distributed processing, Embedded Systems and Applications, Programming Languages, Object Based Software Engineering, Parallel and Distributed Computing, Real Time Systems, Multiprocessing, etc. Deadline for paper submissions: October 12, 2009. |
| ☺ Dec 08-11 | 15th IEEE **International Conference on Parallel and Distributed Systems** (ICPADS'2009), Shenzhen, China. Topics include: Parallel and Distributed Applications and Algorithms, Multi-core and Multithreaded Architectures, Resource Management and Scheduling, Security, Dependable and Trustworthy Computing and Systems, Real-Time Systems, etc. |
| December 10 | Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day! |
| December 16-19 | 16th IEEE **International Conference on High Performance Computing** (HiPC'2009), Kochi(Cochin), India. Topics include: Parallel and Distributed Algorithms, Parallel Languages and Programming Environments, Scheduling, Fault-Tolerant Algorithms and Systems, Scientific/Engineering/Commercial Applications, Embedded Applications, Compiler Technologies for High-Performance Computing, Software Support, etc. Deadline for early registration: November 16, 2009. |

## 2010

| | |
|---|---|
| ☺ January 20-22 | 37th ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2010), Madrid, Spain. Topics include: all aspects of programming languages and systems, with emphasis on how principles underpin practice. |
| | ☺ January 19 **Workshop on Programming Languages meets Program Verification** (PLPV'2010). 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 and/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 8, 2009 (papers). |
| January 25-27 | 5th **International Conference on High Performance and Embedded Architectures and Compilers** (HiPEAC'2010), Pisa, Italy. Topics include: Compilation techniques for embedded processors; Compilation and runtime support for multi- and many-core architectures; Tools and techniques for simulation and performance analysis; Tools for analysis, design, testing and implementation of embedded systems; etc. Deadline for submissions: November 13, 2009. |

> ☺ January 23 2nd **Workshop on GCC Research Opportunities** (GROW'2010). Topics include: current challenges in research and development of compiler analyses and optimizations based on the free GNU Compiler Collection (GCC). Deadline for submissions: November 13, 2009.

☺ February 03-04 2nd **International Symposium on Engineering Secure Software and Systems** (ESSoS'2009), Pisa, Italy. Topics include: security architecture and design for software and systems, systematic support for security best practices, programming paradigms for security, processes for the development of secure software and systems, etc. Deadline for submissions: October 24, 2009 (tutorials).

February 15-18 4th **International Conference on Complex, Intelligent and Software Intensive Systems** (CISIS'2010), Krakow, Poland. Includes track on: Software Engineering for Distributed Systems.

> ☺ Feb 15 **International Workshop on Multi-Core Computing Systems** (MuCoCoS'2010). Topics include: multi-core embedded systems; programming languages and models; applications for multi-core systems; performance modeling and evaluation of multi-core systems; design space exploration; tool-support for multi-core systems; compilers, runtime and operating systems; etc.

February 17-19 18th Euromicro **International Conference on Parallel, Distributed and network-based Processing** (PDP'2010), Pisa, Italy. Topics include: Parallel Computer Systems (embedded parallel and distributed systems, fault-tolerance, multi/many core systems, ...); Models and Tools for Parallel Programming Environments: Advanced Applications (numerical applications with multi-level parallelism, real time distributed applications, distributed business applications, ...); Languages, Compilers and Runtime Support Systems (parallel languages, object-oriented languages, dependability issues, scheduling, ...); etc.

March 09-11 16th **French-speaking Conference on Object-Oriented Languages and Models** (LMO'2010), Pau, France. Deadline for submissions: October 10, 2009 (papers).

March 09-12 23rd IEEE-CS **Conference on Software Engineering Education and Training** (CSEET'2010), Pittsburgh, PA, USA. Theme: "Bridging the Gap between Academia and Industry in Software Engineering Education and Training". Topics include: Curriculum and teaching materials, Software engineering professionalism, Internship and projects for students and graduates, Case studies of educational or training practices, Industry-academia collaboration models, etc. Deadline for submissions: October 1, 2009 (research papers, experience reports, short papers), October 15, 2009 (workshops, panels, tutorials).

☺ March 10-13 41st ACM **Technical Symposium on Computer Science Education** (SIGCSE'2010), Milwaukee, Wisconsin, USA.

March 15-18 14th **European Conference on Software Maintenance and Reengineering** (CSMR'2010), Madrid, Spain. Topics include: Experience reports and empirical studies on maintenance, reengineering, and evolution; Description of education-related issues to evolution, maintenance and reengineering; Mechanisms and techniques for reengineering systems as services; etc. Deadline for submissions: October 9, 2009 (abstracts), October 16, 2009 (full papers), October 23, 2009 (workshops, industry track submissions, tool demonstrations, doctoral symposium, European Projects track).

March 20-28 **European Joint Conferences on Theory and Practice of Software** (ETAPS'2009), Paphos, Cyprus. Events include: FOSSACS, Foundations of Software Science and Computation Structures; FASE, Fundamental Approaches to Software Engineering; ESOP, European Symposium on Programming; CC, International Conference on Compiler Construction; TACAS, Tools and Algorithms for the Construction and Analysis of Systems. Deadline for submissions: October 1, 2009 (abstracts), October 8, 2009 (full papers).

> March 20-28 10th **Workshop on Language Descriptions, Tools and Applications** (LDTA'2010). Topics include: applications of and tools for meta programming in a broad sense, such as Program analysis, transformation, generation and verification; Reverse engineering and reengineering; Refactoring and other source-to-source transformations; Language definition and language prototyping; Debugging, profiling and testing; etc. Deadline for submissions: November 27, 2009 (abstracts), December 04, 2009 (papers).

March 22-26 25th ACM **Symposium on Applied Computing** (SAC'2010), Sierre and Lausanne, Switzerland.

☺ Mar 22-26 **Track on Object-Oriented Programming Languages and Systems** (OOPS'2010). Topics include: Language design and implementation; Type systems, static analysis, formal methods; Integration with other paradigms; Components and modularity; Distributed, concurrent or parallel systems; Interoperability, versioning and software adaptation; etc.

☺ Mar 22-26 **Track on Software Engineering** (SE'2010). Topics include: technologies, theories, and tools used for producing highly dependable software more effectively and efficiently; such as Safety and Security; Dependability and Reliability; Fault Tolerance and Availability; Architecture, Framework, and Design Patterns; Standards; Maintenance and Reverse Engineering; Verification, Validation, and Analysis; Formal Methods and Theories; Component-Based Development and Reuse; Empirical Studies, and Industrial Best Practices; etc.

☺ Mar 22-26 **Track on Real-Time Systems** (RTS'2010). 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 22-26 **Track on Programming Languages** (PL'2010). 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 22-26 **Track on Software Verification and Testing** (SVT'2010). Topics include: development of technologies to improve the usability of formal methods in software engineering, tools and techniques for verification of large scale software systems, real world applications and case studies applying software verification, static and run-time analysis, correct by construction development, software certification and proof carrying code, etc.

March 22-26 17th IEEE **International Conference and Workshops on the Engineering of Computer Based Systems** (ECBS'2010), Oxford, UK. Topics include: Component-Based System Design; Design Evolution; Distributed Systems Design; ECBS Infrastructure (Tools, Environments); Education & Training; Embedded Real-Time Software Systems; Integration Engineering; Model-Based System Development; Modelling and Analysis of Complex Systems; Open Systems; Reengineering & Reuse; Reliability, Safety, Dependability, Security; Standards; Verification & Validation; etc. Deadline for submissions: November 1, 2009 (abstracts), November 10, 2009 (papers). Deadline for early registration: February 22, 2010.

March 24-26 15th IEEE **International Conference on the Engineering of Complex Computer Systems** (ICECCS'2010), Oxford, UK. Topics include: Verification and validation, Reverse engineering and refactoring, Design by contract, Safety-critical & fault-tolerant architectures, Real-time and embedded systems, Tools and tool integration, Industrial case studies, etc. Deadline for submissions: October 23, 2009 (abstracts), October 30, 2009 (papers).

April 06-09 3rd IEEE **International Conference on Software Testing, Verification and Validation** (ICST'2010), Paris, France. Topics include: Verification & validation, Quality assurance, Empirical studies, Inspections, Tools, Embedded software, Novel approaches to software reliability assessment, etc. Deadline for submissions: October 2, 2009 (full papers).

☺ April 13-16 5th **European Conference on Computer Systems** (EuroSys'2010), Paris, France. Topics include: various issues of systems software research and development, such as systems aspects of Dependable computing, Distributed computing, Parallel and concurrent computing, Programming-language support, Real-time and embedded computing, Security, etc. Deadline for submissions: October 23, 2009.

| | |
|---|---|
| ☺ April 19-23 | 24th IEEE **International Parallel and Distributed Processing Symposium** (IPDPS'2010), Atlanta, Georgia, USA. Topics include: 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. Deadline for submissions: December 14, 2009 (PhD forum). |
| ☺April 19 | 15th **International Workshop on High-Level Parallel Programming Models and Supportive Environments** (HIPS'2010). Topics include: all areas of parallel applications, language design, compilers, run-time systems, and programming tools; such as New programming languages and constructs for exploiting parallelism and locality; Experience with and improvements for existing parallel languages and run-time environments; Parallel compilers, programming tools, and environments; Programming environments for heterogeneous multicore systems; etc. Deadline for submissions: December 3, 2009. |
| April 26-29 | 22nd **Annual Systems and Software Technology Conference** (SSTC'2010), Salt Lake City, Utah, USA. |
| ☺ May 02-08 | 32nd **International Conference on Software Engineering** (ICSE'2010), Cape Town, South Africa. 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 5, 2009 (tutorials, education papers, software engineering in practice track, software engineering education track); November 26, 2009 (doctoral symposium Papers); January 7, 2010 (research demonstration papers, new and emerging results papers). |
| May 31 – June 02 | 10th **International Conference on Computational Science** (ICCS'2010), Amsterdam, The Netherlands. Topics include: recent developments in methods and modelling of complex systems for diverse areas of science, advanced software tools, etc. Deadline for submissions: January 1, 2010 (full papers). Deadline for early registration: March 31, 2010. |
| June 14-15 | 2nd USENIX **Workshop on Hot Topics in Parallelism** (HotPar'2010), Berkeley, CA, USA. Topics include: the broad impact of multicore computing in all fields, including application design, languages and compilers, systems, and architecture. Deadline for position paper submissions: January 24, 2010. |
| ♦ June 14-18 | 15th **International Conference on Reliable Software Technologies - Ada-Europe'2010**, Valencia, Spain. Sponsored by Ada-Europe, in cooperation with ACM SIGAda (approval pending). Deadline for submissions: November 16, 2009 (papers, tutorials, workshops), January 11, 2010 (industrial presentations). |
| June 21-23 | AMAST2010 - 10th **International Conference on Mathematics of Program Construction** (MPC'2010), Québec City, Canada. Topics of interest range from algorithmics to support for program construction in programming languages and systems, such as type systems, program analysis and transformation, programming-language semantics, security, etc. Deadline for submissions: December 7, 2009 (abstracts), December 14, 2009 (full papers). |
| June 21-25 | 10th **International Conference on Application of Concurrency to System Design** (ACSD'2010), Braga, Portugal. Topics include: (Industrial) case studies of general interest, gaming applications, consumer electronics and multimedia, automotive systems, (bio-)medical applications, internet and grid computing, ...; Synthesis and control of concurrent systems, (compositional) modelling and design, (modular) synthesis and analysis, distributed simulation and implementation, ...; etc. Deadline for submissions: January 10, 2010 (papers). |
| June 26-30 | 15th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2010), Ankara, Turkey. |
| ☺ Jun 28 – Jul 02 | 48th **International Conference Objects, Models, Components, Patterns** (TOOLS Europe'2010), Malaga, Spain. Deadline for submissions: November 30, 2009 (workshops). |
| December 10 | Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day! |

## SIGAda 2009 Advance Program

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

**Hilton St. Petersburg Bayfront Hotel**
**St. Petersburg (Tampa Bay area), Florida, USA**
**November 1-5, 2009**

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

### Special Keynote Presentations

**Echo: A New Approach to Formal Verification Based on Ada The Technology, and Experience in Security and Medical Devices**
**John Knight**
*University of Virginia*
*Computer Science Department*

**Ada Through the Eyes of Developing Large, Mature, Reliable Systems**
**Richard Schmidt**
*Lockheed Martin*
*Information Systems & Global Services - Civil Group*

**A Look at Ada from Both Sides Now:**
**Government and Defense Contractor Perspectives**
**J.C. Smart**
*Raytheon*
*Intelligence and Information Systems*

**Corporate Sponsors – Platinum**

AdaCore
The GNAT Pro Company

**Corporate Sponsors – Silver**

Ellidiss Software
TNI Europe Limited

LOCKHEED MARTIN

**VISIT http://www.sigada.org/conf/sigada2009 TODAY**
**FOR UP-TO-THE-MINUTE TUTORIAL, PROGRAM, AND EXHIBIT INFORMATION**
**AND ONLINE REGISTRATION**

## SIGAda 2009 VENUE & HOTEL

**Hilton St. Petersburg Bayfront**
333 First Street South
St. Petersburg, Florida 33701 (USA)
Tel: 1-727-894-5000
Fax: 1-727-823-4797

The SIGAda 2009 Conference will be held at the Hilton St. Petersburg Bayfront Hotel. This is a beautiful spot located directly on Tampa Bay, Florida. The weather in November is typically in the high 70's F (20 degrees C) with minimal rain fall. It is easily accessible from either the Tampa or Clearwater/St Petersburg airports. And it is about 1-1/2 hours away from the Orlando International Airport and its associated theme parks. The Tampa Bay area also has the Busch Gardens theme park, the Salvador Dali museum, and numerous beaches on the Gulf coast.

The hotel has reserved a block of rooms for the SIGAda 2009 conference. The conference rate is $95 for single or double occupancy rooms, $105 for triple occupancy rooms, and $115 for quadruple occupancy rooms. A 12% tax will be added per night. All reservations must be guaranteed by credit card. Reservations must be received by October 1, 2009. For further information, see http://www.sigada.org/conf/sigada2009/hotel-rates.html

## EXHIBITORS

SIGAda 2009 will include vendor participation, featuring presentations on their products and services. For specific information, please contact the Exhibits Chair, **Alok Srivastava**, Northrup Grumman, *Alok.Srivastava@AUATAC.com*.

## GRANTS TO EDUCATORS

As in past years, SIGAda is offering grants to educators to attend the conference. Grants cover the registration and tutorial fees; travel funds are not covered, but members of the GNAT Academic Program may be eligible for travel funds from AdaCore. Applications must be sent by e-mail, no later than October 16, 2009. Grantees will receive instructions for on-line registration, which will be accepted through October 28, 2009. Grant program details are available from the conference website or **Prof. Michael B. Feldman**, *mfeldman@gwu.edu*.

## WORKSHOPS / BOFS

Focused workshops are important in shaping Ada technology to better meet the needs of the Ada community. Workshops are free for those registered for the conference. Workshop descriptions are listed at the SIGAda 2009 website. Additional workshops or Birds-of-a-Feather (BoF) sessions are welcome. Workshops have a focused objective and result in a report to be published in *ACM Ada Letters.* BoFs are informal discussion groups. If you would like to propose a Workshop or BoF, please contact the Workshops Chair, **Bill Thomas**, *BThomas@MITRE.Org*

## CONFERENCE TEAM

*Conference Chair*
**Greg Gicca**
AdaCore
gicca@adacore.com

*Program Chair*
**Jeff Boleng, Lt Col, USAF**
US Air Force Academy
jeff.boleng@usafa.edu

*Workshops Chair*
**Bill Thomas**
The MITRE Corporation
BThomas@MITRE.org

*Exhibits Chair*
**Alok Srivastava**
Northrop Grumman
Alok.Srivastava@AUATAC.Com

*Local Arrangements Chair*
**Currie Colket**
colket@acm.org

*Tutorial Chair*
**Richard Riehle**
Naval Postgraduate School
rdriehle@nps.edu

*Webmaster & Proceedings Chair*
**Clyde Roby**
Institute for Defense Analyses
ClydeRoby@ACM.Org

*Publicity Co-Chair*
**Michael Feldman**
George Washington University (ret.)
mfeldman@gwu.edu

*Publicity Co-Chair*
**Ron Price**
Softcrafts
softcrafts@aol.com

*SIGAda Chair*
**Ricky E. Sward**
The MITRE Corporation
rsward@mitre.org

*SIGAda and Conference Treasurer*
**Geoff Smith**
Lightfleet Corporation
gsmith@lightfleet.com

*SIGAda Vice Chr, Meetings/Conferences*
**Alok Srivastava**
Northrop Grumman
Alok.Srivastava@AUATAC.Com

*SIGAda International Representative*
**Dirk Craeynest**
K.U. Leuven (Belgium)
dirk.craeynest@cs.kuleuven.be

*Registration Chair*
**Thomas A. Panfil**
US Department of Defense
Phone and FAX +1 301-498-7313
Office Phone: +1 410 854-5818
tapanfil@acm.org

# Preliminary Call for Papers
# 15th International Conference on Reliable Software Technologies – Ada-Europe 2010

**14-18 June 2010, Valencia, Spain**

http://www.ada-europe.org/conference2010.html

**Conference Chair**

*Jorge Real*
Universidad Politécnica de Valencia, Spain
jorge@disca.upv.es

**Program Co-Chairs**

*Jorge Real*
Universidad Politécnica de Valencia, Spain
jorge@disca.upv.es

*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**

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

**Industrial Chair**

*Erhard Plödereder*
University of Stuttgart, Germany
ploedere@informatik.uni-stuttgart.de

**Publicity Chair**

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

UNIVERSIDAD POLITECNICA DE VALENCIA

In cooperation with
ACM SIGAda
(approval pending)

## General Information

The 15th International Conference on Reliable Software Technologies – Ada-Europe 2010 will take place in Valencia, Spain. Following its traditional 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.

## Schedule

| | |
|---|---|
| 16 November 2009 | Submission of regular papers, tutorial and workshop proposals |
| 11 January 2010 | Submission of industrial presentation proposals |
| 1 February 2010 | Notification of acceptance to all authors |
| 1 March 2010 | Camera-ready version of regular papers required |
| 10 May 2010 | Industrial presentations, tutorial and workshop material required |
| 14-18 June 2010 | Conference |

## Topics

The conference has successfully established itself as an international forum for providers, practitioners and researchers into reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers representing industry, academia and government organizations active in the promotion and development of reliable software technologies. To gather experience on the latest periodic revision of the Ada language standard, contributions that present and discuss the potential of the revised language are especially welcome.

All prospective 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 Architecture and Engineering, Formal Methods, Re-engineering and Reverse Engineering, Reuse, Software Management Issues.
- **Software Architectures**: 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 Technology.
- **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, Trust and Safety, Languages Vulnerabilities.
- **Embedded Systems**: Multicore Architectures, Architecture Modeling, HW/SW Co-Design, Reliability and Performance Analysis.
- **Mainstream and Emerging Applications**: Manufacturing, Robotics, Avionics, Space, Health Care, Transportation, Energy, Games and Serious Games, etc.
- **Experience Reports**: Case Studies and Comparative Assessments, Management Approaches, Qualitative and Quantitative Metrics.
- **Ada and Education**: Where does Ada stand in the software engineering curriculum; how learning Ada serves the curriculum; what it takes to form a fluent Ada user; lessons learned on 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
*Roderick Chapman*, Praxis High Integrity Systems, 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
*Raymond Devillers*, Université Libre de Bruxelles, Belgium
*Franco Gasperoni*, AdaCore, France
*Michael González Harbour*, Universidad de Cantabria, Spain
*José Javier Gutiérrez*, Universidad de Cantabria, Spain
*Andrew Hately*, Eurocontrol CRDS, Hungary
*Peter Hermann*, Universität Stuttgart, Germany
*Jérôme Hugues*, ISAE Toulouse, France
*Hubert Keller*, Institut für Angewandte Informatik, Germany
*Albert Llemosí*, Universitat de les Illes Balears, Spain
*Kristina Lundqvist*, Mälardalen University, Sweden & MIT, USA
*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
*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*, University of Newcastle upon Tyne, UK
*Jean-Pierre Rosen*, Adalog, France
*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
*Santiago Urueña*, Grupo de Mecánica de Vuelo, Spain
*Tullio Vardanega*, Università di Padova, Italy
*Francois Vernadat*, LAAS-CNRS & INSA Toulouse, France
*Daniel Wengelin,* Saab, Sweden
*Andy Wellings*, University of York, UK
*Jürgen Winkler*, Friedrich-Schiller Universität, Germany
*Luigi Zaffalon*, University of Applied Sciences, Switzerland

## Industrial Committee

*Guillem Bernat*, Rapita Systems, UK
*Roderick Chapman*, Praxis High Integrity Systems, UK
*Dirk Craeynest*, Aubay Belgium & K.U.Leuven, Belgium
*Pierre Dissaux,* Ellidiss Technologies, France
*Franco Gasperoni*, AdaCore, France
*Hubert Keller*, Forschungszentrum Karlsruhe GmbH, Germany
*Ismael Lafoz*, EADS CASA, Spain
*Ahlan Marriott*, White-Elephant GmbH, Switzerland
*Erhard Plödereder*, Universität Stuttgart, Germany
*José Simó*, Universidad Politécnica de Valencia, Spain
*Alok Srivastava*, Northrop Grumman, USA
*Rei Stråhle*, Saab Systems, Sweden

## 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 1 March 2010. For format and style guidelines authors should refer to the following URL: *http://www.springer.de/comp/lncs/authors.html*. Failure to comply and to register for the conference 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 may deliver value and insight, but do not fit the selection process for regular papers. Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation to the *Conference Chair* by 11 January 2010. The *Industrial Program Committee* will review the proposals and make the selection. The authors of selected presentations shall prepare a final short abstract and submit it to the *Conference Chair* by 10 May 2010, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the Ada User Journal, which will host the proceedings of the Industrial Program of the Conference.

## Call for Tutorials

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

## Call for Workshops

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

## Call for Exhibitors

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

## Grants for Students

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

# 14th International Conference on Reliable Software Technologies

## Ada-Europe 2009

**Brest, France,**
**June 8-12, 2009**

## "Workshop on Vulnerabilities"

**Program**

09:00 – 10:30
- *Overview and Update of the ISO WG23 Technical Report on Vulnerabilites*, J. Benito/E. Ploedereder
- *Expunging the C/C++ Bias*, J.P. Rosen
- *Ada and Programming Language Vulnerabilities*, S. Michell

10:30 – 11:00 Break

11:00 – 12:30
- *Ada and Programming Language Vulnerabilities (cont)*, S. Michell
- *Argument for Language Subsetting*, T. Vardanega

12:30 – 14:00 Lunch

14:00 – 15:30
- *Concurrency and Vulnerabilities*, A. Burns

15:30 – 16:00 Break

16:00 – 18:00
- *Vulnerabilities Enumeration*, R. Chapman
- *Wrap-Up*, J. Tokar

# Ada Europe 2009 – Workshop on Software Vulnerabilities

***Joyce L. Tokar, PhD***

*Pyrrhus Software, PO Box 1352, Phoenix, AZ, 85001-1352, USA.; Tel: +1 602373 0713; email: tokar@pyrrhusoft.com*

## 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. A workshop was conducted at Ada Europe 2009 to define the Ada Annex to the ISO WG 23 TR and to identify breaks and gaps in the TR with respect to the programming language Ada. This article provides and overview of the workshop and introduces the position papers that were presented at the workshop.*

*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 [1]. Presently, there is a large interest world-wide in software vulnerabilities and how they enable software applications to be infiltrated and corrupted. There are several activities underway to identify and enumerate the collection of software vulnerabilities in existing programming languages. The work being conducted, for the International Organization of Standards (ISO), under the auspices of the Programming Languages Subcommittee (SC22), done by Working Group 23 (WG23), is one such effort.

ISO WG23 has developed a Technical Report (TR) [2] that captures the current view of software vulnerabilities. Annexes to this document are being developed to identify if the vulnerabilities defined in the TR exist in various programming languages. This workshop focused on the content of the Technical Report and its applicability to the Ada and SPARK programming languages.

Several position papers were presented on the status of the vulnerabilities in the Ada programming language. The workshop initiated work on the Ada and SPARK Annexes to the ISO WG23 Technical Report on Vulnerabilities. The workshop also identified missing vulnerabilities in the TR.

## 2 Status of the WG 23 Technical Report on Vulnerabilities

The workshop began with a presentation by John Benito, Convener of WG23. John provided an update on the status of the TR. It has been through the first technical ballot within SC22. There were a lot of comments from various groups which will need to be addressed before reballoting the document.

WG23 is now soliciting Programming Language Groups for Annexes to the TR. The C Annex is being written as well as a FORTRAN Annex. WG23 is very interested in receiving an Ada Annex and a SPARK Annex. WG23 is planning to wait for a couple of Annexes to become available before submitting the final document for approval. These Annexes are intended to help identify which vulnerabilities are language specific and which are common to all languages. WG23 will review the Annexes for content prior to publication. Expect the collection of annexes to be available in 2010.

## 3 Workshop Position Papers

The majority of the workshop attendees developed position papers for the workshop providing insights on key areas of software vulnerabilities and the relationship with the programming language Ada. The full set of papers is provided in this journal.

JP Rosen initiated the workshop position paper presentations with a discussion of the C/C++ bias of the Technical Report. He postured that the TR would be more beneficial to the software community at large if this bias was expunged.

Steve Michell continued the position paper presentation with a summary of the objectives of the WG23 TR, some of the areas that are missing from the TR, such as Concurrency, and a collection of examples of how the vulnerabilities apply to Ada.

The presentation then went in a new direction with a presentation by Tullio Vardanega of the benefits of programming language subsets to eliminate vulnerable features.

Alan Burns complemented Steve Michell's presentation with a detailed discussion of concurrency and software vulnerabilities.

Rod Chapman closed the position paper session by highlighting the benefits and weaknesses of enumerating vulnerabilities.

## 4 Software Vulnerabilities in Ada – Developing the Ada and SPARK Annexes

Following the position paper discussion, the workshop moved on to initiate the development of the Ada and SPARK Annexes to the TR.

### 4.1 Ada Annex

The focus of this session was to get the description in the Ada Annex to correspond to the issues raised in the vulnerabilities section. The focus of the annex entry needs to clarify how the vulnerability is provoked.

The introduction to the Ada Annex needs to include some discussion on the specifics of the Ada programming language environment such as failing to compile, erroneous compilation, and structured name spaces. Additionally, it needs to provide an explanation of how the Ada compiler provides some of the mechanisms for checking for vulnerabilities; other languages require separate tools.

WG23 had provided an outline of the suggested format for the Programming Language Annexes. The annex template suggests that for each vulnerability identified in the TR, there should be a corresponding section in the annex.

The workshop participants worked together to establish an understanding of this format, how to apply it in the development of the Ada annex, and what modifications should be suggested to WG23. In the end, the general format of an entry section was as follows:

- *Language-specific terminology and features* – this section will deal with terminology differences between the body of the document and the terminology used in the language standard. This sub-section may also explain language specific features and terminology that are applicable to the subsequent sub-sections.
- *Description of application vulnerability* – this section will provide a brief description why Ada is susceptible to the vulnerability along with some examples of the vulnerability in an Ada application.
- *Mechanism of Failure* – the consequences of the given vulnerability in the program application are identified in this section.
- *Avoiding the vulnerability or mitigating its effects in Ada* – this section specifies what the programmer can do to avoid or mitigate the vulnerability.

Upon completing the work on the format of the Ada Annex, the workshop attendees were then assigned sections of the TR to work on to develop the corresponding sections for the Ada Annex. During the workshop, six examples were written to be submitted to WG23 for evaluation. The remaining sections of the TR will be addressed by a Rapporteur Group within WG9 (the ISO/IEC JTC 1/SC 22 working group on Ada).

### 4.2 SPARK Annex

The workshop then considered the development of the SPARK Annex. The first question considered was whether SPARK could be included as an Annex given that it is a proprietary language developed by Praxis. Rod Chapman, from Praxis, explained that SPARK is Freely Licensed Open Source Software (FLOSS), the definition of SPARK is publicly available, and the latest version is always publicly available.

SPARK is sufficiently distinct from a programming language design point of view from Ada to warrant its own annex in the TR. It is designed, taught, and verified differently from Ada. It has different characteristics than Ada from a vulnerabilities perspective

The workshop attendees agreed that the SPARK annex should be developed in conjunction with the Ada annex.

## References

[1] NIST Special Publication 268, “Source Code Security Analysis Tool Functional Specification Version 1.0,” May 2007.

[2] ISO/IEC PDTR 24772.2, Information Technology — Programming Languages — Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use, May 2009.

# On Removing Programming Language Bias from the Vulnerabilities Document

***J-P Rosen***

*Adalog, 19-21 rue du 8 mai 1945, 94110 ARCUEIL, France; Tel: +33 1 41 24 31 40; email:rosen@adalog.fr*

## Abstract

*ISO/SC22/WG23 is currently working on a document that identifies vulnerabilities in programming languages. The document is structured as a core report which is supposed to be independent of any programming language, and annexes related to the applicability of each vulnerabitlity in specific languages. Unfortunately, the core exhibits in places a bias, generally towards the C/C++ family of languages. This paper identifies those places in the report where the wording or intent was biased by the features of certain programming languages, and suggests improvements to remove them.*

*Keywords: Ada,C, C++, vulnerabilities.*

## 1 Introduction

ISO/SC22/WG23 is a working group of ISO which has been formed with the goal of producing a technical report (TR) that identifies vulnerabilities in programming languages. The official title of the TR is *Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use* 0. Since the goal of this TR is clearly to provide guidance in selecting a programming language based on the vulnerabilities that each language may exhibit, it is structured as a general core study which does not refer to any programming language in particular, with language annexes that describe how the vulnerability applies, or not, to the given language. Since the document originates from members of the C/C++ community (although there is strong participation from the Ada community), it is unavoidable that some bias from these languages has crept in the core document. For the sake of simplicity, we'll use the name "C*" to designate the C family of programming languages.

This paper identifies places where this bias has been introduced, and suggests improvements to remove them.

## 2 Typical C-isms

As noted above, it is very hard to keep the core document totally language agnostic. For example, the description of vulnerabilities often include examples, such examples are very useful, but they have to be given with some programming language. Moreover, vulnerabilities that exist in only one language have to be addressed in the core document, since the language annexes are not supposed to be adding new vulnerabilities.

However, there are cases where the description of the vulnerability reflects a C* approach (as opposed to C* syntax). Conversely, there are vulnerabilities in some languages that do not exist in C* due to the absence of features that correspond to the vulnerability. Not addressing these vulnerabilities is another hint of a C* bias.

For example, *Pointer Arithmetic* [RVG/6.22][1] addresses (rightly) the vulnerabilities caused by arithmetic operations on pointers. However, the recommendations mention pointer arithmetic only as a way of indexing arrays. Although other languages may provide pointer arithmetic, C is the only language where pointer arithmetic is connected to the indexing of arrays.

Another arguable statement is found in *Argument Passing to Library Functions* [TRJ/6.48]. There is no definition of what "library functions" are, but it seems that the intent is to refer to standard libraries provided with the language. The description states:

> *Libraries that supply objects or functions are in most cases not required to check the validity of parameters passed to them.*

Although such a statement might be applicable to some C* libraries, there is no reason to think that it is a general principle that applies to all languages. There is a general vulnerability connected to subprograms that do not check their arguments, but there is no reason to limit this to "libraries".

## 3 Lack of generality

Some vulnerabilities are related to functionalities that exist in several programming languages, but whose scope and features vary greatly among languages. For example, generics/templates exist in Ada, C++, Eiffel... However, the report describes mainly the C++ view:

"Many languages provide a mechanism that allows objects and/or functions to be defined parameterized by type, and then instantiated for specific types" (*Templates and Generics* [SYM/ 6.25.1])

[1] Each description of a vulnerability is identified with an arbitrary three-letter code. This is intended to make them independent of any renumbering of clauses that may happen during the preparation of the document, but makes it harder to retrieve the place where it is defined. We therefore refer to the vulnerabilities by their code, followed by the corresponding clause number in the version of the document 0 that was current as of June 8th, 2009.

This formulation is clearly too restrictive for Ada, where other entities (such as packages) can be generic, and where the possible parameter kinds include subprograms, constants, variables, and even other packages (through formal instantiations).

Another example can be found in *Likely Incorrect Expression* [KOA/6.32]. This vulnerability is mainly concerned with the unintended use of "=" in place of "==" in expressions. While it is true that this problem is haunting every C programmer, it has no equivalent in other languages. On the other hand, the description only mentions in passing the confusion between "&" and " &&", which does have equivalents in other languages ("and" and "and then" in Ada). Note that this vulnerability should be kept separate from the issue of order of evaluation, which is addressed by *Side-effects and Order of Evaluation* [SAM/6.31].

## 4 Left-out features

The introduction of [1] states that, due to the limited amount of resources, vulnerabilities related to some subjects were deliberately postponed. These subjects include:

- Object-oriented language features
- Concurrency
- Numerical analysis
- Scripting languages
- Some issues related to inter-language operability

Although it is understandable that the subject of vulnerabilities is gigantic, and that it is not possible to address them all, the choice of left-out feature is another indication of language bias: all of these features are either not provided by C (even though some of them, and notably object orientation, are provided by other languages of the C family), or related to domains where C is not particularly fit (like numerical analysis). To Ada users, for example, addressing concurrency would seem a much more important topic than syntactic ambiguity!

## 5 Abstracting the vulnerabilities

It should be understood that the C* bias found in the description of some vulnerabilities does not invalidate the value of the vulnerability; the issue is more on separating the general, high level problem that it addresses (which belongs to the core document) from how it shows in some specific language (which belongs to the language annex). This requires an effort for abstracting the vulnerability.

For example, *String Termination* [CJM/6.16] describes the vulnerability caused by forgetting the null character that terminates a string. The general vulnerability is about using a sentinel value to mark the end of a data structure; there is nothing specific to strings, not even to arrays, here.

Strangely enough, the document distinguishes *Boundary Beginning Violation* [XYX/6.17], *Unchecked Array Indexing* [XYZ/6.18], *Unchecked Array Copying* [XYW/6.19], and *Buffer Overflow* [XZB/6.20]. All these are variants of a single vulnerability: accessing an array outside of its bounds. The origin of this distinction is that in C, it is common practice to allocate arrays in the direction where the stack is growing; therefore, addressing below the array may ruin the return address, while addressing above it does not. This is not even connected to a particular language, but to a specific (although common) implementation technic.

Similarly, there are subtle distinctions between *Type System* [IHN/6.11], *Numeric Conversion Errors* [FLC/6.15], *Pointer Casting and Pointer Type Changes* [HFC/6.21], *Sign Extension Error* [XZI/6.29], and *Type-breaking Reinterpretation of Data* [AMV/6.46]. They are all occurrences of problems with conversions; the only possible distinction could be between semantic-preserving conversions (regular conversions in Ada) and non-semantic-preserving conversions (Unchecked_Conversion in Ada).

The same phenomenon appears with vulnerabilities related to bad pointers: *Pointer Arithmetic* [RVG/6.22], *Null Pointer Dereference* [XYH/6.23], *Dangling Reference to Heap* [XYK/6.24], and *Dangling References to Stack Frames* [DCM/6.40].

In the last two examples, we have clearly single vulnerabilities that can appear, in the C* languages, in various forms; the core should contain only the abstract formulation (incorrect pointer value), leaving the variants to the language annex.

## 6 Cross-references clauses

The standard vulnerability template includes a "Cross-reference" clause to provide links to other documents addressing the given concerns. All vulnerabilities have links to C or C++ standards, and only those, although Ravenscar and Spark (but not the HRG document) are mentioned in the bibliography.

This is a clear indication that the selection of rules was made from C* documents; although a good starting point, documents from other languages should have been considered right from the start. Otherwise, only C* vulnerabilities will be addressed, especially considering that at this point, it could be argued that it is too late to add new vulnerabilities to the document.

## 7 Conclusion

There is no doubt that the vulnerabilities identified in the document are real, and do happen in various programming languages, including the C* family of languages. However, the formulation of some of them, and the selection of vulnerabilities, show a strong C* influence in some cases.

We suggest in this paper some improvements to make the formulation more general and applicable to other languages, and identify the parts that should be moved to language specific annexes; we hope that, by following theses advices, the generality and overall quality of the document could be improved.

## References

[1] ISO/IEC PDTR 24772.2, as of 2009-05-29

[2] Alan Burns, Brian Dobbing and Tullio Vardanega (June 2004). "Guide for the use of the Ada Ravenscar Profile in high integrity systems". ACM SIGAda Ada Letters XXIV (2): 1–74. Now part of annex D of ISO/IEC 8652:1995 with cor. 1 and amdt 1 (Programming language Ada).

[3] John Barnes: "High Integrity Software: The SPARK Approach to Safety and Security"

[4] ISO/IEC TR 15942:2000, Guidance for the Use of Ada in High Integrity Systems.

# Ada and Programming Language Vulnerabilities

***Stephen Michell***
*Maurya Software Inc,Ottawa, Ontario, Canada, email: stephen.michell@maurya.on.ca*

## Abstract

*ISO/IEC/JTC 1/SC 22/WG 23 has requested that other SC22 language working groups prepare annexes to their in-development document ISO/IEC PDTR 24772 Guidance to Avoiding Vulnerabilities in Programming Languages Through Language Selection and Use. WG 9 is working on an anex to this document for the Ada programming language. Although Ada is a well designed language, and with good programming practices, most vlnerabilities discussed in PDTR 24772 do not arise in practice, still there are mechanisms in Ada where many of these vulnerabilities can occur. This paper addresses six of the vulnerabilities and proposes writeups suitable for an Ada annex.*

*Keywords: template, journal, Ada.*

## 1 Introduction

Vulnerabilities are an extreme problem in the interconnected world of millions of computers connected in co-operative and no-co-operative ways. Lax operating system design, network protocol design, language design, application design, and end-user refusal to enable and follow even basic security practices are resulting in the anarchy that is being experienced on the internet. It is imperative that we do everything possible to limit the damage being experienced currently, and to begin to rectify the situation by identifying what vulnerabilities exist, how they are being misused to launch attacks on our infastructure, and how such attacks can be mitigated through improved computer systems design at all levels.

One common attack vector is to identify how an application is weak because of exposed application vulnerabilities that originate from the way that humans program using less than perfect programming languages. These programming languages have vulnerabilities that permit the program to function in ways that were not intended by the designers or developers, and that are not easily detectable by review or tests.

ISO/IEC/JTC1/SC22/WG23, the Programming Languages Vulnerabilities Working Group, was created to help identify, document, and propose corrective action for vulnerabilities that exist in programming languages. WG23 is producing an International Technical Report 24772 entitled "Guidance to Avoiding Vulnerabilities in Programming Languages Through Language Selection and Use"' [1]. This report does a good job of identifying vulnerabilities across many programming languages. The document is missing, however, some important vulnerability classes and all of the annexes that will explain how each vulnerability is addressed by an individual programming language.

WG23 intends to include such annexes, but is relying upon programming language committees to prepare and forward these annexes to WG23 for inclusion. ISO/IEC/JTC 1/SC 22/WG 9 Ada Working Group is gathering resources in order to build such an annex.

Of all of the programming languages that are in common usage, Ada [2] is arguably the most robust language in building applications that will operate in the way that the programmer intended, even in the presents of faults and attacks. However, Ada is not perfect as many of the vulnerabilities identified by WG23 can be present in programs written in Ada, given bad choices in the use of features, bad choices in combination of features, and inadequate reviews.

This paper summarizes the current state of ISO/IEC PDTR 24772, proposes a number of vulnerabilities that should be added to those currently in the document, and presents possible Ada writeups for vulnerabilities currently identified in ISO/IEC/PDTR 24772 that could be put in an Ada-Specific Annex. Proposed writeups are included with this document as Annexes.

## 2 Problem Statement

Every computer system being developed, whether it is targetted for deployment on the open internet, targetted for a private network or designed forstanda-alone mode, must be aware of the risks that are present in the development environment, the acquisition environment, and the ultimate deployment environment. Common attack vectors are not only from direct attacks from over a network connection that targets executing programs or systems, but can include attacks on development systems that deposit malicious code for later activation, and mistakes made during the development of the application that result in unacceptable behaviour.

One of the strengths that ISO/IEC/JTC 1/SC 22/WG23's document PDTR 24772 has over other vulnerability assessment systems such as the Common Weakness Enumeration is that this document considers other attack vectors and does not rely exclusively on the "over the internet"' style of attack. It uses analysis as well as "found in the wild" approaches to identify, categorize and analyze vulnerabilities.

WG23, during the development of its technical report, has identified to date 48 programming language vulnerabilities to be considered. Most of these show up often in language systems written in languages other than Ada, but an honest Ada developer must acknowledge that many of these vulnerabilities can happen in a system programmed in Ada if good choices are not made. We will examine specific issues with some of the issues that will be submitted in this work.

## 3 General Positive Statements About Ada

Ada is a very strongly typed language. This means that many of the worst mistakes found in weakly typed programming languages are completely avoided, unless the programmer takes obvious steps to disable some of these features, such as using 'Unchecked_Conversion to defeat the checking of conversions of unrelated types, using Unchecked_Deallocation to defeat Ada's memory management features, disabling runtime check suppression which may permit out-of-bounds access to data, or combining the program with libraries that cannot be analysed by the compiler.

Ada's first line of defense is the integral static analysis in the compiler. Attempts to abuse the program in source code violate the language rules and are caught in that phase. This checking cannot be disabled or forgotten, as it often is when the analysis tools are an accessory to the language translator, as must be the case with weakly typed languages.

Ada's second line of defense is the runtime checks that enforce the boundaries and typing system that was designed into the application and the language. These checks occur whenever the compiler cannot show statically that the rules are being followed, and ensure that the boundaries on data access and manipulation are consistent with the language rules.

There are good guides for safely using Ada. In particular, following the Ada Quality and Style Guide [3] and International Technical Report 14592, “Guidance for the Use of the Ada Programming Language in High Integrity Systems” [4] will eliminate or all issues.

Another other positive statement about Ada is that it is a language that has been designed for ease of reading and comprehension. This means that domain-specific technical people can reasonably read Ada code and determine the correctness of the algorithm, especially if the units have been coded as recommended in the Ada Quality and Style Guide.

## 4 Early Submission of an Ada Annex

Another question that arises is whether or not WG9 should submit an Annex to the version of the TR that will likely go for DTR voting in late 2009 or early 2010. As a committee member of WG23 and contributor to the TR, my expectation is that there will be no completed annex submissions from C, C++, COBOL and there may indeed be no other language submissions. It would therefore be counterproductive for Ada to submit its Annex, only to have it the only Annex or one of the only Annexes in the document. This could create the perspective that Ada's issues are more serious than those of other languages, when in fact the opposite is true.

## 5 Example Analysis, [CCB] Enumeration Issues

As an example submission for consideration for an Ada specific analysis for an Ada annex to PDTR 24772, a summary of issues for “Enumeration issues” is presented. This is section 6.15 [tag CCB] of the current ISO/IEC/PDTR 24772.

Ada's enumeration types are a strong part of Ada's design. Ada provides mechanisms to guarantee full coverage of any case statement or aggregate that is built using enumeration types. Poor programming practices, however, can lead to behaviours as described in the TR, as follows:

- Using the “others” clause in case statements or aggregates defeats a compiler's coverage analysis, but there will be no branches outside of the case alternatives. Addition of new literals to the enumeration list and failing to put the new alternative in a branch of the case statement or aggregate results in unexpected transfers to the default handler, or assignment of default value.
- Using knowledge of the representations of an object of an enumeration with non-default representations creates the potential that non-representable values can be coereced into the object, or that arrays indexed by the underlying type can contain ``holes''.itle information is set across the whole page but unjustified.

## 6. Example Analysis,[STR] Bit Representation

As an example submission for consideration for an Ada specific analysis for an Ada annex to PDTR 24772, a summary of issues for “Bit Representation issues” is presented. This is [tag STR] of the current ISO/IEC/PDTR 24772.

Ada provides two distinct mechanisms to manage explicitly programmed bits in a system - records with mapped fields, and modular types.

Records with mapped fields are statically checkable by the compiler, and Ada provides facilities for placing, naming, and manipulating bit fields , as well as ways to specify the endianness of data. Mistakes can be made, especially in placement and layout, but it has been found that such mistakes are obvious quickly, usually in test.

Modular types mimic the “C” unsigned type, and, while safer than in more weakly typed languages, have many of the challenges of representation, arithmetic, and logical operations that are discussed in the TR.

## 7 Writeups

The following writeups have been prepared, and are submitted to the workshop for consideration for inclusion in a future Ada-specific annex to TR 24772. They are not included in this document.

- Ada.3.8 Choice of Clear Names [NAI]
- Ada.3.11 Identifier Name Reuse [YOW]
- Ada.3.12 Type System [IHN]
- Ada.3.13 Bit Representation [STR]
- Ada.3.14 Floating Point Arithmetic [PLF]
- Ada.3.15 Enumeration Issues [CCB]

The following sections are included in the Annex, however, since they reflect the work that meets the format and content requested by WG 23.

- Ada 3.5 Deprecated Language Features [MEM]
- Ada 3.6 Preprocessor Directives [NMP]
- Ada 3.10 Identifier Name Reuse [YOW]
- Ada 3.47 Memory Leak [XYL]

## 8. Conclusions

This paper has shown some of the issues that need to be considered in developing a language-specific annex to ISO/IEC PDTR 24772. Many more issues must be addressed and agreed to by the Ada community via ISO/IEC/JTC 1/SC 22/ WG 9 the Ada Working Group, before the annex is ready for inclusion in the TR. It is not expected that the Ada annex will be available before the first version of the PDTR is published.

## Bibliography

[1] "The Ada Programming Language Reference Manual", ISO/IEC 8652:1995 with Technical Corrigendum 1:2001 and Amendment 1:2007, International Standards Organization, Geneva, Switzerland, available from http://ada-auth.org/arm.html

[2] International Preliminary Draft Technical Report 24772 "Guidance to Avoiding Vulnerabilities in Programming Languages Through Language Selection and Use", International Standards Organization, Geneva, Switzerland.

[3] Software Productivity Consortium, "Ada Quality and Style Guide", available from http://www.adaic.org/ docs/95style/html/cover.html

[4] International Preliminary Draft Technical Report 15942, "Guidance for the Use of Ada in High Integrity System", International Standards Organization, Geneva, Switzerland

## Annex - Writeups Ada Annex Items for PDTR 24772

The writeups in this Annex reflect the work that was done by the author in concert with ISO/IEC/JTC 1 /SC 22/WG 9. After the submission of the original writeups to the workshop, WG 9 assigned vulnerability writeups to various individuals, but also recommended section numbering, title and content changes to WG 23. WG 23 accepted some of the changes but made subsequent chages. The writeups in this annex reflect the work that was assigned to the author written in the format that was requested by WG 23.

## Ada.3.5 Deprecated Language Features [MEM]

### Ada.3.5.0 Status and history

July 2009 written by Stephen Michell

### Ada.3.5.1 Ada specific terminology and features

Deprecated: Ada has a number of features that have been declared "deprecated".

> These are documented in Annex L of the Ada Reference Manual.

Pragma restrictions:

> Ada provides "pragma Retrictions" to let developers remove from use language features that are considered problematic by the developer(s). In particular, 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.

### 3.5.2 Description of Vulnerability in Ada

If deprecated language features are used (i.e. the pragma restriction is not applied), then the mechanism of failure for the vulnerability is as described in the parent document.

If the pragma Restrictions(No_Obsolescent_Features) is used then the vulnerability does not exist.

### Ada.3.5.3 Avoiding the vulnerability or mitigating its effects in Ada

Use pragma Restrictions(No_Obsolescent_Features) to prevent the use of any deprecated features.

### Ada.3.5.4 Implications for standardization in Ada

None.

## Ada.3.6 Pre-Processor Directives [NMP]

### Ada.3.6.0 Status and History

200908: created by S. Michell

The vulnerability is not applicable to Ada. Ada does not have a preprocessor.

## Ada.10 Identifier Name Reuse [YOW]

### Ada.3.10.0 Status and history

200905: Submitted By Stephen Michell

20090609: Stephen Michell, JP Rosen – Ada Europe Vulnerabilities Workshop

### Ada.3.10.1 Terminology and features

Homograph: Two declarations are *homographs* if they have the same name, and do not overload each other according to the rules of the language.

Overriding: A subprogram *overrides* another if they have identical names and signatures, except that the controlling operand of one is a derived type of the overridden subprogram.

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.

### Ada.3.10.2 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 6.10.

### Examples of the problem:

```
package body P is
   I : Integer;   -- static object called I
   procedure Calculate(X : in out Float;
                       I : in Integer) is -- parameter called I
   begin
         X := X * float(I*I);  -- want to multiply
                               --  static I * parameter I
                               -- static I hidden, wrong
                               -- product, no diagnostic
         I := I + 1;           -- want to increment static
                               -- I, hidden by parameter I
                               -- get compiler diagnostic
                               -- since we cannot assign
                               -- to an in parameter.
   end Calculate;
   …
end P;
```

Here, the parameter called I hides the static variable I because they have exactly the same type and the parameter *hides"* the outer name. This can be corrected by writing

```
package body P is
   I : Float;   -- static object called P.I
   procedure Proc is
         I : Integer -- local variable called P.Proc.I
   begin
         I := 1; -- removal of local I causes diagnostic
                 -- since must say I := 1.0 for float
   end Proc;
end P;
```

Consider an example with more nested scope:

```
procedure P is
   type Age is range 0..125;
   I : Age;
begin
   <<INNER>> declare
         I : Integer := 0;
   begin
         P.Inner.I := P.Inner.I+1 -- increment local I
                                  -- removal of Inner.I
                                  -- causes diagnostic
         P.I := P.I+1;            -- increment outer I
         -- suceeds even if
         --  inner.I removed
   end Inner;
end P;
```

In this example, P.I and P.Inner.I are *expanded names*.

The failure associated with hiding due to nested scopes applies to Ada. 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.3.7).

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.3.10.3 Avoiding the vulnerability or mitigating its effects

This vulnerability can be avoided or mitigated in Ada in the following ways:

[1] A good way to be guaranteed to keep names separated is to always use the *expanded name.* This guarantees that, even if the simple name could produce a conflict, there is never any doubt as to usage in the mind of the human reader. Indeed, high integrity system guidelines recommend that distinct and representative names be used of items, and that each usage of a name be distinct.

### Ada.3.10.4 Implications for standardization

Ada could define a pragma Restrictions identifier No_Hiding that forbids the use of a declaration that result in a local homograph.

### Ada.3.10.5 Bibliography

None

## Ada.3.47 Memory Leak [XYL]

### Ada.3.47.0 Status and history

200907 written by Stephen Michell

### Ada.3.47.1 Ada-specific terminology and features

Access type:

A type in Ada that references a value of another type. An Access type is called a pointer in other languages.

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. The advantage of a storage pool over the use of the standard heap is that it can be sized exactly to the requirements of the application (to permit 10 items only to be allocated for example). If an application has a storage leak in a storage pool that is sized close to the maximum number of items expected to be in use simultaneously, then storage leaks will be detected early in the test cycle and not in operation.

Furthermore, exceptions raised due to memory failures in a storage pool will not adversely affect storage allocation from other storage pools or from the heap. Storage pools also eliminate the fragmentation problem of generalized storage systems where sufficient space may exist, but there is no allocatable block of sufficient size for an allocation.

Controlled type:

A controlled type is a specialized type in Ada where an implementer can tightly control the assignment, allocation and return of objects of the type. If an access type is a controlled type, then referencing counting techniques can be applied to eliminate storage leaks for that type, i.e. When the reference count is non zero for an object that is dereferencing the object, then the object is not returned to the pool; when the reference count is zero after dereferencing, then the object is returned to the pool or heap.

Pragma Restrictions(No_Allocators):
Pragma Restrictions(No_Local_Allocators):
Pragma Restrictions(No_Implicit_Heap_Allocations):
Pragma Restrictions(No_Unchecked_Deallocations):

These Ada restrictions prevent the application from using any allocators. The first prevents all use of allocators. The second prevents the use of allocators after the main program has commenced. The third prevents the use of allocators that would use the heap, but permits allocations from storage pools. The final one (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.3.47.2 Description of  Vulnerability in Ada

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 the effects are mitigated by the localization of memory references, meaning that the failure in a single storage pool will not affect memory in other storage pools, and the sizing of memory close to the expected maximum will guarantee that any memory leaks will be detected during test.For objects that are objects of a controlled type that uses referencing counting, the vulnerability does not exist.

Ada does not use a garbage collector.

### Ada.3.47.3 Avoiding the vulnerability or mitigating its effects in Ada

- Use storage pools where possible.
- Use controlled types and referencing counting to implement explicit storage management systems that cannot have storage leaks.

### Ada.3.47.4 Implications for standardization in Ada

Future Standardization of Ada should consider:

Implementing a generic reference counting storage management for objects.

# An Argument for Language Subsetting

***Tullio Vardanega***

*University of Padua, Department of Pure and Applied Mathematics, via Trieste 63, 35121 Padua, Italy.*
*Email: tullio.vardanega@math.unipd.it*

## Abstract

*ISO/JTC1/SC22/WG23 is currently busy with the production of a Technical Report entitled "Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use". Language-specific annexes to that document are called from language experts in the communities that share the concerns addressed by WG23. This initiative is an opportunity for the Ada community to reflect on two dimensions of interest: Ada's ability (and the possible strengthening of it) to facilitate the creation of language profiles that may help the user circumvent vulnerabilities; and Ada's stance in comparison to other mainstream languages from the standpoint of support tools that add to subsetting in getting proven applications. This short position paper discusses those two opportunities.*

*Keywords: Ada.*

## 1 Introduction

In the context of programming languages, vulnerabilities can be looked at from two distinct angles:

- as *design faults*, where the presence or absence of certain, possibly fundamental features of the language – which arise from explicit or implicit design choices – makes the achievement of a provably correct implementation of a solution exceedingly difficult and onerous if not altogether impossible; concurrency, distribution, object orientation are among the most fundamental challenges of language design where important faults may occur;
- as *obstacles to use*, when the language includes constructs and exhibits features that may require either the programmer or the programming environment, or perhaps even both, exceeding capabilities to master the resulting complexity and its inherent exposure to obscure effects.

## 2 Vulnerabilities as design faults

A general classification of the former area of problem, transversal across programming languages, is extremely arduous to attain, if at all possible, because it is bound to stir animosity across language camps and also because it lacks universally agreed reference terms in the positive direction. It is in fact so much easier to criticize a programming language for what it does or doesn't do, but it is immensely more difficult and perhaps even futile to sanction what the perfect language should do.

In the ideal world, one should consider attacking this particular fact of the problem of programming language vulnerabilities before facing the other. In practice however, pragmatism and realism alike suggest that one should make do with programming languages as they are in their fundamental nature, and help their user see and understand their defects and circumvent their weaknesses. This does of course not stop the insightful from willing to help out language standards to get better. One should reckon however that language standards must necessarily only evolve at the pace that does not undermine the user community, their legacy and their investments.

That is seemingly the stand taken by the language-neutral strand of the work carried out by ISO/JTC1/SC22/WG23, chartered to produce a Technical Report entitled "Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use" [1].

The inherent pragmatism of the chosen approach inevitably diminishes the influence that this work may have on the advancement of the addressed programming languages, since it would not directly expose their design faults. Moreover, so long as economic countermeasures exist against the identified vulnerabilities, no major effort to remove those design faults will likely be deemed viable.

A deeper level of introspection is also difficult and perhaps even naïve to expect from the language-specific annexes that are to be produced by the relevant expert bodies to accompany the core document product of WG23.

## 3 Vulnerabilities as obstacles to use

As for the latter angle of the problem, the analysis of the incidence of *obstacles to use* in a specific language, the effort looks attractive and strategically important for Ada because it should be expected that Ada ought to come out very well positioned in that regard.

Two ISO reports that were produced in the past by the Ada community should be regarded, in both approach and contents, as a very useful basis for the production of the Ada-specific annex of the WG23 Technical Report. Those well-known reports were:

1. ISO/IEC TR 24718:2005, Guide for the Use of the Ada Ravenscar Profile in High Integrity Systems [2]
2. ISO/IEC TR 15942:2000, Guide for the Use of the Ada Programming Language in High Integrity Systems [3]

Interestingly, although different in vision and intent, both documents take the view that *language subsetting* is the only practical route to attaining traceable and reproducible correlation between the external attributes of software in use and the internal attributes of the corresponding program code, and therefore, ultimately, of the features in use of the implementation language.

I would argue that language subsetting should be left to neither the user nor the application-domain cognizant, since its very definition poses problems intrinsically analogous to language design, which is instead the specialized job of *language designers*.

The question therefore arises as to what approach should be taken to identifying language features and constructs that should be avoided to mitigate risks or contain verification effort: enumerating language features that might incur vulnerability when used at the application level in fact does not in itself produce a language subset, as there is no intrinsic coherence in the sequence of vulnerabilities adopted in the WG23 TR (while there was some in ISO TR 15942:2000, and a lot of it in ISO TR 24718:2005).

## 4 Programming environments

Another angle to consider in that line of work is that the programming language and the programming environment (support and analysis tools) should complement one another in the avoidance of vulnerabilities at application level. The existence of mature analysis techniques that may help the user single out problem areas in the source code was a distinct concern in both ISO TR 15942:2000 and ISO TR 24718:2005, and it should arguably continue to be so in the development of the Ada-specific annex of the WG23 TR.

An important by-product of considering that particular angle is that its discussion in an ISO document might motivate the birth or perhaps the rejuvenation of good support tools for Ada, with the SPARK environment as the ideal model and the term of reference.

A good argument, which I encountered in a recent discussion thread of WG23 and that I fully agree with, is that when it comes to avoiding vulnerabilities, the language itself is only one factor: a less well-defined language for which there are good tools may in fact be no worse than a better-defined language for which tools are less well developed. Ada (in the general sense, thus above SPARK) seems to be in the latter category when one considers the density of good support tools. Yet, it must be certainly acknowledged that, by virtue of the very motivations behind the Ada language specification, the Ada compilers are all most excellent support tools in the way of preventing software defects from making into executable programs. While this is certainly true, I would however argue that compilers in their conventional sense need not and cannot be the sole support tool apt to facilitate the production of safe, secure and predictable applications. The wisdom to this notion stems from the observation that the verification of a program does not require and should perhaps not even suggest the production of an executable (as in very definition of static analysis) which is instead the fundamental job of a compiler. That the Ada compiler may have or may easily acquire the knowledge needs to support a score of static analyses is an evidence I have often been exposed to. I have also seen however numerous cases in which that wealth of knowledge is ditched without the user getting to seeing it. It is for this reason therefore that I feel that an answer to the demand for additional support tools, in the tune of “Ada has an all powerful compiler and does therefore not need any further external support tools” would miss out the target.

## References

[1] ISO/IEC PDTR 24772.2, Information Technology — Programming Languages — Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use, May 2009

[2] ISO/IEC. Information technology - programming languages - guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report TR 24718, ISO/IEC, 2005.

[3] ISO/IEC. Information technology - programming languages - guide for the Use of the Ada Programming Language in High Integrity Systems. Technical Report TR 15942, ISO/IEC, 2000.

# Concurrency Vulnerabilities

*A. Burns, A.J. Wellings*

*Real-Time Systems Group, Department of Computer Science, University of York, UK.*

## 1 Introduction

The current draft of the ISO/IEC document on Language Vulnerabilities [5] contains little on the subject of concurrent language features. And yet concurrency is a significant issue in the design and implementation of systems. Sequential programming languages ignore such issues and assume that the underlying operating system will deal with the management of threads/tasks and their inevitable interactions and synchronisations.

There are a number of motivations for wanting to write concurrent programs and therefore to have the notion of concurrency in the programming language. Concurrency does however bring with it a number of new vulnerabilities, and language features that support concurrent programming need to be assessed in the same way that sequential features are currently being scrutinized.

Note that the decision not to use a concurrent programming language does not remove these vulnerabilities; many will be present in the operating system (OS) and the API used by the sequential program to access to the concurrency features of the OS. Indeed many of the vulnerabilities will be more extreme as they cannot easily be mitigated by semantic restrictions in the language.

In this short position paper three topics are addressed: concurrency, communication and synchronisation, and scheduling. For each topic a series of issues are considered.

Note the concurrent entity that is termed a task, a thread, a process or sometimes an event in the many different concurrent programming languages is called a task in the following descriptions.

## 2 Concurrency

Many different concurrency models can be found in programming languages. There are distinguished by issues such as: static or dynamic task creation, hierarchical task structures, and the degree to which one task can influence/interfere with the behaviour of other tasks.

### Static task creation

The simplest task structure available is one in which there is a fixed number of tasks that are created at the time of program instantiation. All tasks then exist for the duration of the program, which may be unbounded. Vulnerabilities from this simple model include the following.

1. Not all tasks start their execution (e.g. they may fail during activation).
2. Premature silent termination of a task or tasks.
3. Tasks executing with inappropriate initialisation parameters.
4. Overflow of task-local data (task attributes).

The first two vulnerabilities result in the program executing with only partial functionality. If tasks are relatively independent of each other then this situation may not be apparent to the tasks that are actually executing. A task may fail for a number of reasons including functional problems or execution issues such as stack overflow.

A common pattern for a real-time task is for it to be release periodically (with period T) and for its execution urgency to be influenced by its deadline (D) – both of these measures being of some appropriate time. Such a task could be instantiated with inappropriate values for its parameters.

The task may still function perfectly, but at the wrong rate (or it may be more likely to miss a deadline). Where tasks use attributes stored in the TCB (task control block) during execution then an overflow of data may result in another task's TCB being corrupted. Similar issues apply to stack usage.

A multi-tasking program is likely to behave differently on a multiprocessor platform than it does on a single CPU. Tasking always introduces some level of nondeterminacy; this is not of itself a vulnerability as the correct behavior of a program should be invariant over this

### Dynamic and hierarchical task creation

If tasks can be created during the program's execution then many different program architectures can be constructed. Task termination becomes a normal event, and dependencies between tasks based on creation and termination are possible. Additional vulnerabilities from these features include the following.

5. Memory exhaustion due to dynamic object creation.
6. Memory exhaustion due to memory leakage.
7. Tasks indefinitely waiting for other tasks to terminate.
8. Tasks subject to errors propagating from child task creation.

All dynamically created objects, whether tasks or not, require memory and hence are subject to finite memory constraints. In very dynamic programs where many tasks are recreated and then terminate, it is important to ensure that terminated tasks can (and do) relinquish all memory allocated to them.

A common pattern in dynamic task programs is for one task to create another, and to subsequently wait for its termination. Creating a task may open up a vulnerability, and waiting for another task to terminate will clearly lead to indefinite postponement if that task does not in fact complete.

### Inter-task influence/interference

Task can usually communication data and synchronise their executions via the language features that are described in section 3. But there are languages features that allow other forms of influence/interference. Some of these are via the scheduling facilities, and are covered in section 4. Here we cover abort, asynchronous exceptions, and asynchronous transfer of control (ATC). All of these are used to get the 'immediate' attention of the designated task. Polling for a state change is inappropriate. Vulnerabilities from these features include the following.

9. Rogue task aborting correctly behaving task (rather than visa versa).
10. Task (or program scope) terminated whilst holding locks/resources.
11. Task being in an inappropriate state to handle ATC or asynchronous exception.

The abort feature is one of the most controversial in that the motivation for its inclusion (to remove a rogue task) is mirrored by its main drawback (rogue task removing others).

All task terminations can cause problems if the task is not in the correct state for termination. But this is especially true when termination is imposed from outside. If the task is not terminating but its control is being influenced from outside then again there is the problem of this influence taking effect when the task is vulnerable, for example while updating a shared complex data structure.

## 3 Communication and Synchronisation

Two general forms of communication are possible: synchronous and asynchronous. Synchronisation can be explicitly supported or programmed. Three combinations are considered in turn.

### Asynchronous communication via shared variable

Vulnerabilities from these features include the following.

12. Unintentional use of unprotected shared variables.
13. Mutual update problem.
14. Race conditions.
15. Livelocks.
16. Memory caching.

Shared variable are a well known error-prone language feature. As a result no language relies only on such variables.

But some languages do allow their use and require the programmer to ensure that the intended behaviour is delivered. Race conditions covers all situations where non-determinacy allows (perhaps rarely) the ordering of accesses to shared variables to be incorrect with respect to the intended behaviour of the program. Livelocks occur when tasks loop checking for the value of shared variable to change – but the tasks responsible for making these changes are also blocked in these busy loops. Memory management may allow tasks to store copies of shared variables in local registers. This may lead to updates not been propagated as expected.

### Asynchronous communication with synchronisation support

A wide range of support features are available in different languages; for example, semaphores, signals, monitors and protected objects. Vulnerabilities from these features include the following.

17. Race conditions.
18. Deadlocks.
19. Indefinite postponements.
20. Protocol failures.

Once a task can be suspended then deadlocks and unbounded suspension become possible. Parts, or all, of the program can fail to make adequate progress. If a low-level primitive such as a signal or semaphore is used with shared variables to support a protocol such as readers/writers then errors can lead to rare race conditions and protocol failures.

### Synchronous communication (eg. rendezvous)

Simple CSP-like primitives and extended rendezvous are supported in different languages. Vulnerabilities from these features include the following.

21. Race conditions.
22. Deadlocks.
23. Indefinite postponements.

Although a similar list to before, these vulnerabilities are less severe with synchronous communications facilities. Indeed modeling and proof systems can be used to show the absence of these problems within programs, but only if the rendezvous is a simple one.

## 4 Scheduling and Real-Time Issues

In this section there are a number of issues to consider. First we will cover time and clock primitives, then scheduling and related topics.

### Clocks and time

For real-time systems it is necessary to have access to a clock, measure time intervals and suspend a task for an interval of time. Vulnerabilities from these features include the following.

24. Drift between system clock and 'real-time'.
25. Drift between clocks on a distributed platform.
26. Inappropriate incorporation (or not) of leap seconds and time zone changes.
27. Mismatch between delay/sleep intent and clock granularity.

These are all well known timing/clock issues [2, 3].

### Asynchronous and synchronous task control

These language features allow one task to control the executable state of another task. Vulnerabilities from these features include the following.

28. Suspended tasks not being continued subsequently.
29. Tasks being suspended whilst holding locks/ resources.
30. Race conditions.

Similar arguments to those for banning the use of abort can be applied to this level of task control. Being indefinitely suspended is almost the same as being aborted.

### Fixed priority scheduling

In this section the most common form of task dispatching is considered. If the language does not directly support such a policy then all scheduling must be under direct user control using the synchronous/asynchronous task control methods discussed above (and the dynamic priority scheme covered below). For fixed priority scheduling the vulnerabilities include the following.

31. Priority inversion.
32. Starvation.
33. Assumptions of scheduling analysis not been met by the program, for example execution times, blocking times, minimum times between sporadic tasks, intensity of interrupts, overheads of run-time, garbage collection overheads etc. requirements.
34. Excessive asynchronous traffic (interrupts/events) generated.

Priority inversion occurs through the use of a synchronisation primitive that does not take priority into account. Starvation occurs when there is not enough processing time available for the low priority tasks to make adequate progress.

### Program control over scheduling parameters or policy

If a scheduling scheme such as fixed priority scheduling is supported by the language then it is usual to allow the program to exercise control over some of the scheduling parameters, such as the assignment (static or dynamic) of priorities to tasks and the periods and deadlines of the tasks. Vulnerabilities from these features include the following.

35. Loss of liveness (some tasks fail to make progress).
36. Loss of timeliness (some task failing to meet a deadline).

## 5 Conclusions

This short paper has attempted to highlight the many different vulnerabilities that exist with concurrent programming languages. There are a large number (and variety) of language features that support various aspects of concurrent programming. Not all can be used safety and there are many vulnerabilities that the above review has highlighted. For many of these vulnerabilities it is possible to define language-level mitigation. Some are simply to not allow the use of (non-essential) features. Others point to safe usage patterns. As noted in the introduction, the decision not to use a concurrent programming language does not remove these vulnerabilities; many will be present in the operating system (OS) and the API used by the sequential program to gain access to the concurrency features of the OS.

It is possible to take an extensive set of language features, such as those provided by Ada tasking, and define a subset (and other restrictions) so that a profile is defined that has adequate expressive power and a minimum of vulnerabilities. One candidate for this would be the Ravenscar profile for Ada [1, 4]. Similar profiles need to be defined for other languages. We note that Java has started to undertake this process under the auspices of the Java Community Process (JSR 302).

## References

[1] A. Burns, B.Dobbing, and T. Vardanega. Guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report YCS-2003-348, University of York, Department of Computer Science, 2003.

[2] A. Burns and A. J. Wellings. Real-Time Systems and Programming Languages. Addison Wesley Longman, 4th edition, 2009.

[3] G.F. Coulouris, J. Dollimore, and T. Kindberg. Distributed Systems, Concepts and Design. Addison Wesley, 4th edition, 2005.

[4] ISO/IEC. Information technology - programming languages - guide for the use of the Ada Ravenscar Profile in high integrity systems. Technical Report TR 24718, ISO/IEC, 2005.

[5] ISO/IEC. Information technology - programming languages - guidelines to avoiding vulnerabilities in language selection and use. Technical Report PDTR 24772 – draft, ISO/IEC, 2009.

# The Pros and Cons of Enumerating Programming Language Vulnerabilities

***Roderick C. Chapman***

*SPARK Team, Praxis High Integrity Systems, 20 Manvers St,. Bath BA1 1PX, UK.; Tel: +44 1225 466991; email: sparkinfo@praxis-his.com*

## Abstract

*There are many on-going efforts to enumerate so-called "Vulnerabilities" in programming languages. This position paper considers the benefits, limitations and potential dangers of such efforts.*

*Keywords: Vulnerabilities, CWE, Programming Languages*

## 1 Introduction

The recent concern over security in software has given rise to several notable efforts to enumerate so-called "Vulnerabilities" in programming languages and/or the way such languages should or shouldn't be used, such as the CWE and CVE efforts, the ISO SC22 OWGV group, and the SANS "Top 25" list. These vulnerabilities range from simple well-known programming mistakes to subtle application- and domain-specific properties. Some are specific to particular languages, some are generic, and some seem to have nothing to do with programming languages at all.

This paper tries to lay out a "playing field" for describing vulnerabilities, and goes on to point out some pros and cons of the enumeration efforts.

## 2 A playing field for static analysis

This section lays out a simple scheme for categorizing security-related vulnerabilities in software, mostly according to the ease with which they can be described and checked. These things are important—if we want automated tools to check for the absence of such things, then we must be willing to specify *precisely* what a particular vulnerability is and is not. We must also consider the soundness (false-negative rate), completeness (false-positive rate), and efficiency with which tools can check for such properties—asking tool vendors to solve undecideable or NP-hard problems is unlikely to be productive.

This paper sets out five categories:

### Level 1 – Elementary Mistakes and Discipline

At this level, we have basic rules that avoid common programming errors, enforcement of a subset language, and so on. These can be enforced very efficiently in a manner that is both sound and complete.

Example: Syntactic coding standard rules, such as "don't use templates", "don't use tasking" and so on.

### Level 2 – Absence of Undefined Behaviour

Many common imperative languages, such as C, C++, Java and Ada, list a set of behaviours that is said to be "undefined" or "erroneous." These problems are particularly annoying, since they typically fall into the category of "things we'd like to get rid of but are too hard for a compiler to solve." Compilers can do pretty much anything they like with such things, and are under no obligation to document their behaviour or even to be consistent across compilations, compiler versions, optimization levels and so on.

For code-analysis tools, these are a serious headache. The absence of such issues is a clear goal, but in doing so we are consciously asking verification tools to solve problems that compilers can't, but the same "laws of physics" (most notably Rice's Theorem[1]) still apply, making this something of a tall order. Existing approaches fall into two camps: language subsetting (i.e. making the problem simpler until an acceptable solution can be achieved), or admission of unsound behaviour (i.e. admit it's a tough problem, and do the best you can…)

Example: Elimination of uninitialized variables in unsubsetted C.

### Level 3 – Type Safety

"Type Safety"[2] is a concept used by programming language designers. An accessible definition owes to Wright and Felleisen:

- (Type-) preservation - "Well typedness" of programs remains invariant under the transition rules (i.e. evaluation rules or reduction rules) of the language.
- Progress - A well typed program never gets "stuck", i.e., never gets into an undefined state where no further transitions are possible.

Robin Milner put it somewhat more briefly: "Well-typed programs never go wrong."

In terms of specific languages, type-safety can be thought of as encompassing the absence of exceptions, buffer-overflows, numeric under- or over-flow, the injection of mal-formed input data and so on.

In some ways, type-safety mops up a great many common vulnerabilities – for example, all input-data validation vulnerabilities can be treated as just special cases of type-safety in languages that provide facilities to describe such conditions.

Example: Absence of buffer-overflow. Typically, these properties are very hard to statically check soundly and efficiently, and with an acceptably low false-positive rate. These are definitely “non-trivial” program properties so are subject to the limits imposed by Rice’s Theorem. Can be attacked by analysis techniques such as model-checking, abstract interpretation, or theorem-proving.

### Level 4 – Application- and Domain-Specific Properties

At level 4, things finally start to get interesting. Here we have properties that are derived from the security policy and requirements of *your* application and its environment, not from some pre-defined list supplied by a committee or from a book.

Example: Security Property 3 from the Tokeneer[3] system: “Whenever the door is in an insecure state, then the alarm is ringing.”

These properties can be extremely difficult to state, especially in a form that is sufficiently formal for a tool to check. Functional properties can be checked by theorem-proving, model-checking and so on. Some properties (e.g. absence of covert channels) remain beyond the state-of-the-art.

### Level 5 – Stuff we haven’t even thought of yet

We must acknowledge the existence of attacks and vulnerabilities that we don’t even know about yet. Worse, there may be vulnerabilities known to attackers, but unknown to us – the developers.

(Retrospective) example: Many smartcards in the late 1990s turned out to be susceptible to Differential Power Analysis (DPA). At the time, this seemed fine, since DPA hadn’t been invented (or at least wasn’t widely known.)

## 3 Observations and concerns

Returning to the issue of enumeration of vulnerabilities, I’d like to offer a few observations:

**Observation 1**: For any given programming language, there is a finite and well-known set of vulnerabilities at levels 1, 2 and 3. These can be enumerated since the set is finite.

**Observation 2**: Level 4 issues are application- and domain-specific. They depend on your application, its environment (including threats and attackers), and your security policy. As such, the set of these things is so large as to be effectively infinite. Therefore, efforts to enumerate such vulnerabilities in general will never terminate.

**Observation 3**: Many existing enumerations, such as the SANS “Top 25” include level 4 properties. For example, all so-called “SQL injection” attacks fall into this category. To check for the absence of such problems, a tool needs a sufficiently precise description of what exactly constitutes a “valid” SQL query, which is clearly dependent on the application, its domain, the state of the database, and the overall system security policy. Overly generic description of the vulnerability (“i.e. warn about all SQL strings passed to the database”) just leads to an overtly high false-alarm rate.

**Observation 4**: Most Level 4 properties actually have nothing to do with programming languages.

Reconsider the example Level 4 property shown above from Tokeneer: “Whenever the door is in an insecure state, then the alarm is ringing.”

If your program fails to enforce that property, is that a “Vulnerability” of C, C++, Ada, or Java?!?! Obviously, this is a ridiculous question, so why do we expect a general “Programming Language X Vulnerability Checking Tool” to verify such a thing?

Some tools can be specialized by describing properties that we want to be verified. Two main camps exist – some tools allow such additional properties to be expressed as user-defined small “programs”, “rules”, or “checkers” which extend the basic set of properties which are “built in” to the tool. The Coverity Extend tool is such an example. In the second camp, some languages allow desired properties to be expressed as assertions or contracts which can be checked by a tool. The most notable examples in this second camp include Eiffel, SPARK, and JML for Java. This leads to my next observation:

**Observation 5**: Our ability to express and verify Level 4 properties *critically* depends on the programming language chosen. The ability to express such a property depends on the presence of and the expressive power of the “rule” or “assertion” language, and then there are the soundness, completeness and efficiency of the tool’s checking algorithm to consider.

## 4 The Pros and Cons of Enumerating Vulnerabilities

So, to turn to some pros and cons of enumeration:

### Pros

- An enumerated list gives the developer a tangible list of items to check, as a starting point for further verification.
- A list gives tool vendors a bare-minimum set of things that should be checked for a given language.
- Compliance with a list can be used as a means to compare the capabilities of different languages and tools.

### Cons

- Given that level 4 properties are innumerable, we must accept that any finite list will be incomplete at the point of publication and will rapidly become obsolete.
- Tool vendors will have a strong incentive to claim “100% compliance” (whatever that means) with a list of vulnerabilities, so once all the vendors claim they “tick all the boxes” the efficacy of such a list as a basis for users to compare tools and languages will rapidly fall to zero.

- Users might get the impression that "ticking all the boxes" is the *end* of the verification process, not the beginning. We see the absence of such vulnerabilities as a basic set of hygiene rules that should be observed before the real verification (i.e. level 4 properties), somewhat like a surgeon washing his or her hands before starting the real work.
- Tools and languages which are *unsound* for basic properties are dangerous, since they give a false sense of security for verification at higher levels. What's the point of claiming verification of type-safety if a tool is unsound for a more basic property like variable initialization? Saying this doesn't matter is like a surgeon who claims to be so clever and skilled that they don't need to wash their hands (or at least it doesn't matter if they only wash one hand but not the other…)
- There is a trend that vulnerability descriptions should be both language-independent and informal. This is tempting for those writing the guidance since it widens the "market" for the rules to a wide range of tool vendors and users. On the other hand, this is rather useless for tool builders, who inescapably need fully formal and language-specific rules to implement. Given informal descriptions, tool vendors will implement whatever seems "good enough" or (more likely) will claim that whatever their tool already does is "good enough" to tick-the-box and claim compliance. This leads to a "Tower of Babel" effect – all the tools claim "100% compliance" yet all differ in their behaviour! This is not a conjecture—it is *exactly* the state of the market for MISRA C checking tools, and has not improved much for a decade - arguably because the editors of MISRA C Version 2 rejected the use a simple formal notation that would have radically improved the precision of the standard. I am dismayed to see the same problem emerging for security vulnerability checking tools. If language designers want tools that are sound and precise, then fully formal and language-specific specifications of properties are a must.

## 5 Conclusions

I believe that enumeration of vulnerabilities has a strong role as a *starting point* for software verification, not an end-point, but users must be educated to appreciate this. I hope that enumeration will also give rise to a scientific comparison of the capabilities of particular tools and languages, and will not degenerate into a mere "ticking all the boxes" exercise that will leave users none the wiser.

I hope this paper will serve as a starting point for discussion within OWGV and other groups.

## References

[1] http://en.wikipedia.org/wiki/Rice's_theorem

[2] http://en.wikipedia.org/wiki/Type_safety

[3] http://www.adacore.com/tokeneer

# 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 #46: Incompatibilities between Ada 83 and Ada 95

**Robert Dewar, AdaCore**

*Date: 29 September 2008*

**Abstract:** Part 1, Unconstrained arrays in generics

**Let's get started…**

For the most part, Ada 95 is entirely upwards compatible with Ada 83, meaning that correctly written portable Ada 83 code can be compiled with an Ada 95 compiler, and no changes are required.

However, there are some cases of incompatibilities which have to be addressed when dealing with legacy code. You can of course compile in Ada 83 mode using the pragma Ada_83, or the equivalent compiler switch -gnat83 (/83 if you are using VMS). However, this means you can't use any Ada 95 features as you continue development, so it is better to fix these incompatibilities if possible.

In this series of Gems we will address these incompatibilities. Some are really easy, others trickier. In this first installment, we address one of the most common cases.

Consider the following declaration,

```
generic
  type T is private;
package GP is
  …
end GP;
```

In Ada 83 you could instantiate this generic with an unconstrained type, for example:

```
package NP is new GP (String);
```

This is no longer allowed in Ada 95, since this represents a so-called "contract-model violation". If you try to compile the instantiation in Ada 95 mode, you will get an error:

```
8.  package NP is new GP (String);
    >>> actual for "T" must be a definite subtype
```

String is an indefinite subtype, meaning basically that it is unconstrained. In contrast, a definite subtype means a subtype whose bounds are constrained, so something like

```
subtype S50 is String (1 .. 50);
package NP is new GP (S50);
```

would be legal, but unconstrained String cannot be used.

The contract model for generics says that if a generic compiles error free, then all possible instantiations must compile error free. But if you allow the above instantiation, then whether this is legal or not depends on whether the generic has a declaration of a variable of type T. If it does, then the instantiation is illegal, since you can't have variables of type String without specified bounds.

Let's assume that we don't have any illegal instantiations in the legacy code. That means that, in a case like the above, in fact there are no declarations of variables of type T. In that case, all you have to do is to change the generic as follows:

```
generic
  type T(<>) is private;
package GP is
  …
end GP;
```

The use of (<>) here means that instantiation with an unconstrained subtype is allowed, and consequently the generic will check to ensure that no variables of type T are declared in the template, as you can see from this example:

```
1. generic
2.   type T (<>) is private;
3. package GP is
4.   Var : T;
           |
   >>> unconstrained subtype not allowed (need
       initialization)
5. end;
```

So, following the Ada 95 rules, the contract-model violation is now avoided. Either the generic allows unconstrained subtypes or it does not, and the generic is properly checked when it is compiled.

In summary, we do have a real incompatibility between Ada 83 and Ada 95 here, but it fixes a real hole in the language, and it is easy to fix up old Ada 83 code to meet the new Ada 95 rules.

## Gem #63: The Effect of Pragma Suppress

**Gary Dismukes, AdaCore**

*Date: 21 April 2009*

**Abstract:** The features of Ada have generally been designed to prevent violating the properties of data types, enforced either by compile-time rules or, in the case of dynamic properties, by using run-time checks. Ada allows run-time checks to be suppressed, but not with the intent of allowing programmers to subvert the type system.

**Let's get started…**

One of Ada's key strengths has always been its strong typing. The language imposes stringent checking of type and subtype properties to help prevent accidental violations of the type system that are a common source of program bugs in other less-strict languages such as C. This is done using a combination of compile-time restrictions (legality rules), that prohibit mixing values of different types, together with run-time checks to catch violations of various dynamic properties. Examples are checking values against subtype constraints and preventing dereferences of null access values.

At the same time, Ada does provide certain “loophole” features, such as Unchecked_Conversion, that allow selective bypassing of the normal safety features, which is sometimes necessary when interfacing with hardware or code written in other languages.

Ada also permits explicit suppression of the run-time checks that are there to ensure that various properties of objects are not violated. This suppression can be done using pragma Suppress, as well as by using a compile-time switch on most implementations (in the case of GNAT, with the -gnatp switch).

In addition to allowing all checks to be suppressed, Pragma Suppress supports suppression of specific forms of check, such as Index_Check for array indexing, Range_Check for scalar bounds checking, and Access_Check for dereferencing of access values. (See section 11.5 of the Ada Reference Manual for further details.)

Here’s a simple example of suppressing index checks within a specific subprogram:

```
procedure Main is
  procedure Sort_Array (A : in out Some_Array) is
    pragma Suppress (Index_Check);  - - eliminate check
                                     - - overhead
  begin
    …
  end Sort_Array;
end Main;
```

Unlike a feature such as Unchecked_Conversion, however, the purpose of check suppression is not to enable programs to subvert the type system, though many programmers seem to have that misconception.

What’s important to understand about pragma Suppress is that it only gives permission to the implementation to remove checks, but doesn’t require such elimination. The intention of Suppress is not to allow bypassing of Ada semantics, but rather to improve efficiency, and the Ada Reference Manual has a clear statement to that effect in the note in RM-11.5, paragraph 29:

*There is no guarantee that a suppressed check is actually removed; hence a pragma Suppress should be used only for efficiency reasons.*

There is associated Implementation Advice that recommends that implementations should minimize the code executed for checks that have been suppressed, but it’s still the responsibility of the programmer to ensure that the correct functioning of the program doesn’t depend on checks not being performed.

There are various reasons why a compiler might choose not to remove a check. On some hardware, certain checks may be essentially free, such as null pointer checks or arithmetic overflow, and it might be impractical or add extra cost to suppress the check. Another example where it wouldn’t make sense to remove checks is for an operation implemented by a call to a run-time routine, where the check might be only a small part of a more expensive operation done out of line.

Furthermore, in many cases GNAT can determine at compile time that a given run-time check is guaranteed to be violated. In such situations, it gives a warning that an exception will be raised, and generates code specifically to raise the exception. Here’s an example:

```
X : Integer range 1..10 := …;
..
if A > B then
  X := X + 1;
  ..
end if;
```

For the assignment incrementing X, the compiler will normally generate machine code equivalent to:

```
Temp := X + 1;
if Temp > 10 then
  raise Constraint_Error;
end if;
X := Temp;
```

If range checks are suppressed, then the compiler can just generate the increment and assignment. However, if the compiler is able to somehow prove that X = 10 at this point, it will issue a warning, and replace the entire assignment with simply:

```
raise Constraint_Error;
```

even though checks are suppressed. This is appropriate, because (1) we don’t care about the efficiency of buggy code, and (2) there is no “extra” cost to the check, because if we reach that point, the code will unconditionally fail.

One other important thing to note about checks and pragma Suppress is this statement in the Ada RM (RM-11.5, paragraph 26):

*If a given check has been suppressed, and the corresponding error situation occurs, the execution of the program is erroneous.*

In Ada, erroneous execution is a bad situation to be in, because it means that the execution of your program could have arbitrary nasty effects, such as unintended overwriting of memory. Note also that a program whose “correct” execution somehow depends on a given check being suppressed might work as the programmer expects, but could still fail when compiled with a different compiler, or for a different target, or even with a newer version of the same compiler. Other changes such as switching on optimization or making a change to a totally unrelated part of the code could also cause the code to start failing.

So it’s definitely not wise to write code that relies on checks being removed. In fact, it really only makes sense to suppress checks once there’s good reason to believe that the checks can’t fail, as a result of testing or other analysis. Otherwise, you’re removing an important safety feature of Ada that’s intended to help catch bugs.

## Gem #68: Let’s SPARK! - Part 1

**Yannick Moy, AdaCore**

*Date: 29 June 2009*

**Abstract:** In this Gem and the next one, we present a simple walk-through of SPARK's capabilities and its integration with GPS. In this first Gem, we show how to set up a SPARK project and prove that your SPARK programs are free from uninitialized variable accesses and that they execute without run-time errors.

**Let's get started…**

With Praxis and AdaCore now teaming up to offer an integration of SPARK technology inside GPS (see http://www.adacore.com/home/products/sparkpro/), many GPS users will be interested in trying out the proof capabilities of SPARK on their own Ada programs. Of course it's a little more involved than that. SPARK is not only a set of tools for verifying high-assurance systems, but also incorporates a language that must be learned.

The SPARK language is made up of two parts, one of which is a subset of Ada (whose features will obviously already be familiar to Ada programmers!), meant to facilitate code understanding and proofs, and the other part being a specification language, meant to express properties of programs. Quite conveniently, the SPARK specifications (a.k.a. SPARK annnotations, to distinguish them from Ada specifications) are expressed within stylized Ada comments of the following form:

```
--# <some annotation here>
```

so SPARK annotations don't interfere with normal Ada compilation.

This Gem and the next one demonstrate various of the properties you can prove with SPARK, and how these relate to the SPARK annotations written by the user. As a simple example, we will take a procedure which searches linearly for a value in an array, and returns the index, if any, at which the value is found.

Here is the specification file search.ads:

```
package Search is
  type IntArray is array (Integer range <>) of Integer;
  procedure Linear_Search
    (Table : in IntArray;
     Value : in Integer;
     Found : out Boolean;
     Index : out Integer);
end Search;
```

And the body file search.adb:

```
package body Search is
  procedure Linear_Search
    (Table : in IntArray;
     Value : in Integer;
     Found : out Boolean;
     Index : out Integer)
  is
     I : Integer := 0;
  begin
     Found := False;
     while I <= Table'Last loop
        if Table(I) = Value then
           Found := True;
           Index := I;
           exit;
        end if;
        I := I + 1;
     end loop;
  end Linear_Search;
end Search;
```

Let's first get set up for using SPARK:

1. Copy the files search.ads and search.adb into a directory named search.
2. Open GPS with a default project in the same directory.
3. Expand file search.adb.
4. Select SPARK/SPARKMake from the menu. This generates a file search.idx.
5. Copy the following code to a file called search.cfg:

```
package Standard is
  type Integer is range -2**31 .. 2**31-1;
end Standard;
```

6. From the Menu, select Project/Edit Project Properties.
7. Go to the page Switches/Examiner.
8. Select the following options:

```
Index File          : search.idx
Configuration File  : search.cfg
Analysis            : Data Flow only
Generate VCs        : yes (check it)
```

That's it!

Now open search.adb and select SPARK/Examine File. The SPARK Examiner runs and outputs the following messages:

```
Flow Error 602 - The undefined initial value of Index
      may be used in the derivation of Index
Warning 402 - Default assertion planted to cut the loop
Note - Information flow analysis not carried out
```

The Flow Error indicates that, although Index is an out parameter, it is not initialized on all paths through Linear_Search. One could argue that our intent here is to access Index only when Found is set to True. SPARK considers initialization errors too serious to allow such subtleties, and requires that all paths through the procedure must initialize all out parameters. Let's comply and initialize Index:

```
begin
  Found := False;
  Index := 0;
```

After we rerun the Examiner, we are left with a Note that is simply a reminder of our choice of options, and a Warning that we will explain in the next Gem.

Now, we can continue with proving that our procedure is free from run-time errors, such as integer overflow and out-of-bounds array accesses. Select SPARK/Simplify All. The SPARK Simplifier runs. To see the result of this tool's execution, select SPARK/POGS. This opens a file search.sum, which summarizes all the proofs in the following table:

```
VCs for procedure_linear_search :
-----------------------------------------------------------------------------------
     |       |                    |  ---Proved In--  |       |       |
#    | From  | To                 | vcg | siv | plg | prv | False | TO DO |
-----------------------------------------------------------------------------------
1    | start | rtc check @ 13     |     | YES |     |     |       |       |
2    | start |    assert @ 15     |     | YES |     |     |       |       |
3    | 15    |    assert @ 15     |     | YES |     |     |       |       |
4    | 15    | rtc check @ 16     |     |     |     |     |       |  YES  |
5    | 15    | rtc check @ 18     |     | YES |     |     |       |       |
6    | 15    | rtc check @ 21     |     |     |     |     |       |  YES  |
7    | 15    |    assert @ finish | YES |     |     |     |       |       |
8    | 15    |    assert @ finish | YES |     |     |     |       |       |
-----------------------------------------------------------------------------------
```

Each line corresponds to a Verification Condition (VC), which must be proved in order to guarantee that the program is free from run-time errors. Each column corresponds to the result of the proof attempt. If YES appears in one of the 4 "Proved In" columns, the proof was successful. If YES appears in the "False" column, there is something definitely wrong. If "YES" appears in the "TO DO" column, we don't know: either the program is wrong, or it is too complex to prove.

Since column "TO DO" is not empty here, not all proofs were successful. In the next Gem, we will give you more details about failed proofs. The reason for the failures here is that program Linear_Search is incorrect, meaning that run-time errors can be raised. To see this, just complete the program with the following code in main.adb, and build the executable.

```
with Search;
use Search;
procedure Main is
  Table : IntArray(1..10) := (others => 0);
  Found : Boolean;
  Index : Integer;
begin
  Linear_Search(Table, 0, Found, Index);
end Main;
```

Now run the executable, and it raises an exception:

```
raised CONSTRAINT_ERROR : search.adb:16
      index check failed
```

This is because we are passing an array to Linear_Search that starts at index 1 in its parameter Table, whereas Linear_Search loop assumes that the array starts at index 0. SPARK correctly assumes that we can pass in such an array to Linear_Search, and thus it fails to prove that Linear_Search is free from run-time errors.

Let's correct the code of Linear_Search:

```
procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer) is
begin
   Found := False;
   Index := 0;

   for I in Integer range Table'Range loop
     if Table(I) = Value then
        Found := True;
        Index := I;
         exit;
     end if;
   end loop;
end Linear_Search;
```

Let's do it again: Examine, Simplify All, POGS …

This time, columns "False" and "TO DO" are empty, meaning that all proofs were successful.

```
VCs for procedure_linear_search :
-----------------------------------------------------------------------------------
     |       |                    |  ---Proved In--  |       |       |
#    | From  | To                 | vcg | siv | plg | prv | False | TO DO |
-----------------------------------------------------------------------------------
1    | start | rtc check @ 11     |     | YES |     |     |       |       |
2    | start | rtc check @ 13     |     | YES |     |     |       |       |
3    | start |    assert @ 13     |     | YES |     |     |       |       |
4    | 13    |    assert @ 13     |     | YES |     |     |       |       |
5    | 13    |    assert @ 13     |     | YES |     |     |       |       |
6    | 13    | rtc check @ 14     |     | YES |     |     |       |       |
7    | 13    | rtc check @ 16     |     | YES |     |     |       |       |
8    | 13    |    assert @ finish | YES |     |     |     |       |       |
9    | 13    |    assert @ finish | YES |     |     |     |       |       |
-----------------------------------------------------------------------------------
```

Thus, we have proved that procedure Linear_Search is free from run-time errors.

In the next Gem, after the summer break, we will prove that procedure Linear_Search actually respects a given contract.

## Gem #69: Let's SPARK! - Part 2

**Yannick Moy, AdaCore**

*Date: 7 Sepetmber 2009*

**Abstract:** In this Gem and the previous one, we give you a simple walkthrough of SPARK's capabilities and its integration with GPS. In the previous Gem, we showed how to set up a SPARK project and prove that your SPARK programs are free from uninitialized variable accesses and that they execute without run-time errors. In this Gem, we show how to prove that your SPARK programs respect given contracts.

**Let's get started…**

In the last Gem, we proved that procedure Linear_Search was free from uninitialized variable accesses and run-time errors, which are safety properties of Linear_Search.

Now we can try to prove a specific behavioral property of Linear_Search, expressed as a contract between Linear_Search and its callers. A contract will consist of a precondition that callers of Linear_Search are responsible for establishing, before calling Linear_Search, and a postcondition that Linear_Search must establish, before returning to the caller. If not present, a default "true" pre- or postcondition is assumed.

Let's prove that when Linear_Search returns with Found = True, the value of Table at Index is Value. This can be expressed in SPARK as a postcondition on procedure Linear_Search that is located after its declaration in search.ads:

```
procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer);
--# post Found -> Table(Index) = Value;
```

Notice the implication symbol ->, which is only defined in SPARK annotations.

Let's call the SPARK tools, as we did in the previous Gem: Examine, Simplify All, POGS …

This time, the column "TO DO" is not empty [Table in the end of the page – editor note].

If we right-click on this line and select SPARK/Show Simplified VC, GPS opens a file linear_search.siv, which shows that the unproved VC corresponds precisely to the postcondition we just added:

```
C1:   found -> element(table, [index]) = value .
```

This is the conclusion (C) that the prover tries to prove with the set of hypotheses (H) above it. If we look at the hypotheses, we see that the conclusion cannot indeed be proved. This has to do with the warning which we already saw in the previous Gem:

```
Warning 402 - Default assertion planted to cut the loop
```

To prove a property of a procedure with a loop, we cannot unroll the loop an unbounded number of times. Therefore, SPARK "cuts" the loop with a loop invariant, which is a property that the loop maintains. Unless you provide such a loop invariant, SPARK assumes by default that nothing is maintained through the loop. If you provide one, SPARK will prove three things:

1) the loop invariant holds when control enters the loop for the first time

2) the loop invariant is maintained during an arbitrary run through the loop

3) the loop invariant is sufficient to prove the postcondition

What is missing in our case is the information that Found remains False throughout the loop. Let's add it, with the following syntax in search.adb:

```
for I in Integer range Table'Range loop
   --# assert Found = False;
```

Let's do it again: Examine, Simplify All, POGS …

Notice that the warning about a default assertion disappeared.

This time, the "TO DO" column is empty, so we have successfully proved Linear_Search's postcondition!

```
VCs for procedure_linear_search :
----------------------------
      |       |                     |  ---Proved In--  |       |       |
#     | From  | To                  | vcg  | siv  | plg  | prv  | False  | TO DO |
----------------------------
1     | start | rtc check @ 11      |      | YES  |      |      |        |       |
2     | start | rtc check @ 13      |      | YES  |      |      |        |       |
3     | start |    assert @ 13      |      | YES  |      |      |        |       |
4     | 13    |    assert @ 13      |      | YES  |      |      |        |       |
5     | 13    | rtc check @ 14      |      | YES  |      |      |        |       |
6     | 13    | rtc check @ 16      |      | YES  |      |      |        |       |
7     | 13    |    assert @ finish  |      | YES  |      |      |        |       |
8     | 13    |    assert @ finish  |      |      |      |      |        |  YES  |
----------------------------
```

Finally, let's see how SPARK deals with global variables, by adding a counter to Linear_Search, which is incremented by one each time the call succeeds. We need to state explicitly that Counter is part of the state of this package, which we do using an "own" annotation below. We also need to state explicitly that Counter is initialized using an "initializes" annotation. The following declaration should go into search.ads:

```
package Search
--# own Counter;
--# initializes Counter;
is
   Counter : Natural := 0;
```

Now, we increment Counter in Linear_Search's body in search.adb:

```
      if Table(I) = Value then
         Counter := Counter + 1;
```

If we try to run the Examiner at this point, it flags an error:

```
      Semantic Error 1 - The identifier Counter is either
            undeclared or not visible at this point
```

This is because reads and writes of global variables are part of a subprogram specification in SPARK. Since Linear_Search does not declare in its specification that it reads or writes a global variable, it is not allowed to do so in its body. So let's add a SPARK annotation to state that Linear_Search reads and writes Counter.

```
procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer);
--# global in out Counter;
--# post Found -> Table(Index) = Value;
```

Let's do it again: Examine, Simplify All, POGS …

We get a new unproved VC in the column "TO DO", which corresponds to the following conclusion:

```
      C1:    counter <= 2147483646 .
```

This time, it is because SPARK has detected a problem during the proof that the update of Counter does not overflow. Indeed, it could overflow! The solution is to add a precondition to Linear_Search, that promises that it will never be called in a state where Counter is the largest integer value.

```
procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer);
--# global in out Counter;
--# pre  Counter < Integer'Last;
--# post Found -> Table(Index) = Value;
```

As before, we must modify the loop invariant. Here, we just have to repeat this information in the loop invariant:

```
   for I in Integer range Table'Range loop
      --# assert Found = False and Counter < Integer'Last;
```

Let's do it again: Examine, Simplify All, POGS …

Everything is proved!

Notice that the promise made by the precondition will have to be proved by callers of Linear_Search …

Finally, let's add to the SPARK annotation of Linear_Search that Counter is incremented by one when Linear_Search returns Found = True. This can be expressed as a postcondition relating the value of Counter at procedure entry, denoted Counter~, and the value of Counter at procedure exit, denoted Counter:

```
procedure Linear_Search
  (Table : in IntArray;
   Value : in Integer;
   Found : out Boolean;
   Index : out Integer);
--# global in out Counter;
--# pre  Counter < Integer'Last;
--# post Found -> (Table(Index) = Value and Counter =
     Counter~ + 1);
```

Notice that in the precondition we simply use Counter to denote the value at procedure entry, because the precondition is precisely evaluated at procedure entry.

As before, we update the loop invariant to state that the current value of Counter is the same as the one at procedure entry:

```
   for I in Integer range Table'Range loop
      --# assert Found = False and Counter < Integer'Last
      --#        and Counter = Counter~;
```

Let's do it a last time: Examine, Simplify All, POGS …

Everything is proved!

Remember though that we can only prove a contract we wrote, which may be very different from saying abstractly that procedure Linear_Search is "correct". Who knows what the correct behaviour of Linear_Search means for the human who programmed it?

Let's recap what we have seen so far. SPARK is a language that combines a strict subset of Ada with annotations written inside stylized Ada comments. We have seen various kinds of SPARK annotations: preconditions introduced by pre; postconditions introduced by post; loop invariants introduced by assert; frame conditions introduced by global, own, and initializes. The SPARK tools allow us to check that a procedure is free from uninitialized variable accesses, that it executes without run-time errors, and that it respects a given contract, written next to its declaration, that callers can rely on.

In later Gems we will explore in more depth the capabilities of SPARK and its integration with GPS. In the meantime, you can learn more about SPARK in the SPARK tutorial at http://www.adacore.com/home/products/sparkpro/tokeneer/discovery/.

# 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

attn. Rei Stråhle
Saab Systems
S:t Olofsgatan 9A
SE-753 21 Uppsala
Sweden
Phone: +46 73 437 7124
Fax: +46 85 808 7260
Email: Rei.Strahle@saabgroup.com
*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*