ADA USER JOURNAL

Volume 31
Number 1
March 2010

Contents

	Page
Editorial Policy for *Ada User Journal*	2
Editorial	3
Quarterly News Digest	5
Conference Calendar	29
Forthcoming Events	36
Special Issue: Ada User Journal 30th Anniversary	
B. Tooby "*Opinion: The Word 'Coding' Considered Harmful*"	41
E. Schonberg "*Origins and history of GNAT*"	42
J. Barnes "*We don't know nothing*"	45
B. Dobbing "*The Ravenscar Tasking Profile for High Integrity Real-Time Programs*"	47
J. Barnes "*The SPARK way to Correctness is Via Abstraction*"	55
S. T. Taft "*Object-Oriented Programming Enhancements in Ada 200Y*"	65
P. Leroy "*Memories of a Language Designer*"	74
Ada-Europe Associate Members (National Ada Organizations)	76
Ada-Europe 2009 Sponsors	Inside Back Cover

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

Welcome to the special issue commemorating the 30th Anniversary of the Ada User Journal.

In this past year we celebrated this anniversary with some initiatives: a special paper about the history of the Journal was published in the first issue of Volume 30, one year ago; and a poster-based history presentation was offered at both the Ada-Europe 2009 and the SIGAda 2009 conferences.

We are now dedicating the March 2010 issue, the official 30th anniversary issue, to re-print selected papers from the Journal's 30-year history. The intent of this initiative is to offer our readership a sample of the papers that may be considered to have had the most impact and relevance at the time of publication, in the several incarnations of the Ada User Journal (Ada UK News, Ada-Europe News, Ada User and Ada User Journal).

In order to evaluate and select the papers to re-print, a small group of former editors and distinguished readers of the Journal was set up. These "**Guest Editors of the 30th Anniversary Issue**" are *Albert Llemosi*, *Andy Wellings*, *Dan Simpson*, *Dirk Craeynest*, *Ian Pyle*, *Jean-Pierre Rosen*, *John Barnes*, *Juan António de la Puente*, *Michael González Harbour* and *Tullio Vardanega*. We are grateful to them for their invaluable collaboration in the preparation of this issue.

Resulting from this process, limited only by the Journal's size limits, we re-print seven papers, spanning almost three decades, ranging from small points of view to large technical papers. Some of them had to be re-set from the available (paper only) copies and others regenerated from the sources. They may look different, but have the same ageless contents (unless errors were introduced in the editing process for which I apologize).

The first paper (in chronological order), "*Opinion: The Word 'Coding' Considered Harmful*" by Brian Tooby, was originally printed in Ada User, Vol. 7 N. 3, in September 1986, and provides an interesting view of the role that was then expected to be played by the Ada specification and the Ada body in supporting the modular, computer-aided, design of programs.

In the second paper, Edmond Schonberg presents the "*Origins and history of GNAT*", originally printed in the Ada-Europe News, Issue 20, in March 1995. This is the first paper of an issue entirely dedicated to the GNAT compiler, undoubtedly one of the tools which helped keep Ada alive and well. This particular paper provides information on how GNAT came to be, and of its connection to the development of Ada 95.

The following paper, by John Barnes, is "*We don't know nothing*", and it was originally printed in Ada User, Vol. 17 N. 4, in December 1996. The paper provides a humorous, but to the point and valid, description of the different language design perspectives of Ada and C/C++.

The paper "*The Ravenscar Tasking Profile for High Integrity Real-Time Programs*", by Brian Dobbing, was printed in the Ada User Journal, Vol. 19 N. 4, in January 1999. It provides an advance description of the Ravenscar profile, which provided Ada with a subset for building concurrent high-integrity applications, an important effort resulting from the International Real-Time Ada Workshop (IRTAW) series.

The paper, "*The SPARK way to Correctness is Via Abstraction*", also by John Barnes, appeared in the Ada User Journal, Vol. 22 N. 4, in December 2001. This paper provides an overview of SPARK, one of the most significant technologies developed around Ada for safety critical systems.

The following paper, "*Object-Oriented Programming Enhancements in Ada 200Y*", by Tucker Taft, appeared in the Ada User Journal, Vol. 24, N. 2, in June 2003. It presents a detailed description of the most important object-oriented features being developed in the 2005 revision of the language.

Last, but definitely not least, the issue re-prints a paper by Pascal Leroy, "*Memories of a Language Designer*", published in the Ada User Journal, Vol. 27 N. 3, in September 2006. In this paper, the author, at that time the Chairperson of the Ada Rapporteur Group, offers his behind-the-scene reflections on the 2005 revision process.

I would like to congratulate the authors of these papers, also thanking them for their contributions to the Journal.

Continuing with the contents of this issue, I wish to underline the name change of the former News section to better reflect the nature of its contents, which I know that you will continue to find useful. The issue also provides a calendar of relevant events, where I note the several Ada-related and focused events of 2010, two of which (Ada-Europe and SIGAda) are presented in more detail in the Forthcoming Events section. The Ada Gems and Ada User Guide sections are not included due to space restrictions. However they will come back in the next issues of the Journal.

I also refer the readers to the latest additions to the Online Archive of the Ada User Journal, which now provides the full contents of issues since March 2001. Its contents are slowly being extended when electronic sources of older issues become available. If you have in your possession some of these files, or you know of their whereabouts, please contact us. Also, Ada-Europe is missing a copy of the proceedings of the 1st International Conference on Reliable Software Technologies – Ada-Europe 1996, which took place in Montreux, Switzerland. If you have a spare copy, then please consider donating it to Ada-Europe.

As a final personal note, I would like to add that it was very movitating to organize this celebration year. I hope that you find the results as rewarding as I did, and that you will continue supporting the Journal, both by reading and disseminating it, and by continuing to provide its worthwhile contents.

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

An Invitation to Join Ada-Europe

What is Ada-Europe?

Ada-Europe is an international organization, set up to promote the use of Ada. It aims to spread the use and the knowledge of Ada and to promote its introduction into academic and research establishments. Above all, Ada-Europe intends to represent European interests in Ada and Ada-related matters.

In its current form, Ada-Europe was established in 1988. As there is no European legal framework to govern such organizations, it was established according to Belgian Law. Currently, the member organizations are: Ada-Belgium, Ada-Denmark, Ada-Deutschland, Ada-France, Ada-Spain, Ada in Sweden and Ada in Switzerland. Individual members of these organizations can become indirect members of Ada-Europe. Direct membership is available to individuals in countries without national member organization.

What does Ada-Europe do?

The best-known of Ada-Europe's activities is its annual conference. These conferences usually attract 100 to 150 participants. They involve three days of lectures and presentations, and provide the perfect opportunity to discuss new information and exchange experiences with fellow Ada users. As well as the usual conference features, you have the opportunity to attend an additional two days of tutorials dealing with specialist Ada matters. The conference also hosts an exhibition, where Ada-related products are presented.

Ada-Europe offers a framework for setting up working groups and task groups to discuss and investigate technical aspects of using Ada on a European basis. It provides grants for Ada-related conferences and activities.

The members of Ada-Europe also receive the quarterly Ada User Journal, produced by Ada-Europe. This journal contains Ada-related papers, experience reports, details of past, present and future Ada events and activities, and reviews of new publications and products. The journal is usually distributed via the national member organizations, but can also be mailed directly at additional postage costs.

A reduced registration fee at the annual Ada-Europe conference is an additional benefit to direct and indirect members registered with Ada-Europe by their national organizations. On a semi-regular basis, Ada-Europe "surprises" its individual members with useful material: in 2006 for example, the then recently published Ada 2005 Reference Manual was such a surprise benefit of Ada-Europe membership.

How to become a member of Ada-Europe?

Individuals

If you want to become a member of Ada-Europe, please join your national Ada organisation and become an indirect member of Ada-Europe. In some countries, indirect membership in Ada-Europe is automatically part of your national membership; in other countries, it is an optional element of your national membership.

As benefits you will receive:

- a free copy of the quarterly Ada User Journal, distributed via the national Ada organisations
- a reduced registration fee at the annual Ada-Europe conference (exceeding the cost of your indirect membership)
- in some years, a "surprise" distribution (in 2008 it was the "Ada 2005 Rationale" published by Springer)

Your benefits run from April to March of the following year.

If your country does not have a national Ada organisation, you can contact the Secretary of Ada-Europe to become a direct member of Ada-Europe. Your benefits are the same as for indirect members, except that the Journal is shipped directly to you.

Institutions

National Ada organisations are the primary promoters of corporate memberships. In case a national Ada organisation exists in your country, it can offer its corporate members to designate individuals as indirect members of Ada-Europe at the Ada-Europe individual indirect membership fee (plus any fees that your national organization charges).

In case no national organisation exists in your country, corporate membership may be established directly with Ada-Europe.

Further information

For further information please refer to Ada-Europe’s website at http://www.ada-europe.org, or contact the General Secretary of Ada-Europe.

National organizations contacts are available on the last page.

Quarterly News Digest

Marco Panunzio

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

Contents

Ada-related Organizations 5
Ada-related Events 5
Ada Semantic Interface Specification (ASIS) 6
Ada and Education 6
Ada-related Resources 8
Ada-related Tools 8
Ada-related Products 13
Ada and GNU/Linux 15
Ada and Microsoft 16
References to Publications 17
Ada Inside 17
Ada in Context 19

Ada-related Organizations

New ARA sponsorship levels

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 16 Dec 2009 18:34:55 -0600
Subject: New ARA sponsorship levels
Newsgroups: comp.lang.ada

The Ada Resource Association has announced new sponsorship levels, including an inexpensive "contributor" sponsorship. The new levels allow any organization of any size to help support the evolution and marketing of Ada. Sponsors get increased visibility on the Ada IC website along with other benefits.

More information on the new sponsorship levels and their benefits can be found at:

http://www.adaic.com/ara/sponsor.html

Show your support for Ada: sponsor the ARA today!

Randy Brukardt

Disclaimer: The vast majority of my funding for Ada standardization activities comes from the ARA. So I'm more than a little interested in the organization having additional sponsors.

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 — Call for Industrial Presentations

From: Dirk Craeynest <dirk@asgard.cs.kuleuven.be>
Date: Sun, 3 Jan 2010 19:30:51 +0100 CET
Subject: FINAL CfIP, Conf. Reliable Software Technologies, Ada-Europe 2010
Newsgroups: comp.lang.ada, fr.comp.lang.ada,comp.lang.misc

FINAL Call for Industrial Presentations

15th International Conference on

Reliable Software Technologies - Ada-Europe 2010

14 - 18 June 2010, Valencia, Spain

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

*** DEADLINE Monday 11 JANUARY 2010 ***

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.

In addition to the usual Call for Papers, 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 Jorge Real (jorge@disca.upv.es) 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.

In addition to the award for best regular paper, Ada-Europe will also offer an honorary award for the best presentation, considering both regular and industrial presentations.

Schedule

11 January 2010: Submission of industrial presentation proposals

01 February 2010: Notification of acceptance to all authors

10 May 2010: Industrial presentations required

14-18 June 2010: Conference

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

Please circulate widely.

ACM SIGAda 2010 — Call for Technical Contributions

From: Michael Feldman <mfeldman@seas.gwu.edu>
Date: Wed, 03 Feb 2010 16:18:51 -0600
Subject: Call for Technical Contributions -- ACM SIGAda 2010
Newsgroups: comp.lang.ada

Call for Technical Contributions -- ACM SIGAda 2010

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

Hyatt Fair Lakes Hotel

Fairfax, Virginia (USA) (near Washington, DC)

October 24-28, 2010

Submission Deadline: June 25, 2010

Sponsored by ACM SIGAda

ACM's Special Interest Group on the Ada Programming Language in cooperation with SIGBED, SIGCAS, SIGCSE, SIGPLAN, Ada-Europe, and the Ada Resource Association

http://www.acm.org/sigada/conf/sigada2010

[see the Forthcoming Events section in this issue —lmp]

Ada Semantic Interface Specification (ASIS)

ASISEyes

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Tue, 12 Jan 2010 20:33:49 -0800 PST
Subject: ASISEyes : show you the ASIS interpretation of an Ada source
Newsgroups: comp.lang.ada

[…]

Here is ASISEyes, which as its name suggest, is a little application which integrates well in GPS (possibly others IDE also) and whose purpose is to make you see, side by side, the ASIS view (interpretation) of an Ada source, beside the Ada source.

This is an aid in two main areas:

1) Learning and understanding ASIS (how does ASIS view this and that ?)

2) Debugging aid when an ASIS application seems to fail (what was the semantic context returned by ASIS when this great ASIS application break at that point of that source) ?

The distribution comes in two archives : one ZIP for Windows users and one tar.gz for Unix-like users

http://www.les-ziboux.rasama.org/download/asiseyes-for-gps.zip

http://www.les-ziboux.rasama.org/download/asiseyes-for-gps.tar.gz

As it is an ASIS application, it must be built for your environment, compiler and ASIS implementation.

There are some limitations and usages notes which are documented in the README.txt file and the other README-gps.txt file. The "gps" directory contains files required for the integration into GPS.

Please, read the notes which comes with this directory.

[…]

Ada and Education

Seminar on Ada and SPARK at K.U.Leuven

From: <Dirk.Craeynest@cs.kuleuven.be>
Date: Sat, 6 Feb 2010 18:16:30 +0100 CET
Subject: Tue Feb 23 Seminar - Ada and SPARK for education and research
Newsgroups: comp.lang.ada, fr.comp.lang.ada, comp.lang.misc, be.education, be.comp.programming

The Computer Science Department of the K.U.Leuven

and the Ada-Belgium organization

are pleased to announce the seminar

T e c h n o l o g y U p d a t e :

A d a a n d S P A R K

f o r e d u c a t i o n a n d r e s e a r c h

State-of-the-art programming language technology with Ada

Formal specifications made practical with SPARK

organized with support from AdaCore and Altran Praxis

on Tuesday, February 23, 2010, 14:00-18:00

at the K.U.Leuven, Department of Computer Science

Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium

http://distrinet.cs.kuleuven.be/events/AdaEvent/

Introduction

Ada is a state-of-the-art programming language especially suitable for large, long-lived applications where safety, security, and reliability are critical. Due to its approach of detecting errors as soon as possible it is also generally usable for all types of applications. SPARK is a formally-defined programming language based on Ada, intended to be secure and to support the development of high-integrity software.

This event is primarily intended for the educational and research community, and will present experts from academia and industry who believe that using Ada and SPARK in education and research is fundamental to form the software engineers of tomorrow. Why Ada? Because they believe that Ada is the right choice for a range of courses including elementary programming, data structures, software engineering and for more advanced courses and research in compiler construction, real-time systems, robotics, cryptography, etc. Ada and SPARK embody the best contemporary ideas in software technology, and students exposed to these languages at an early stage of their career become more skilled and principled programmers.

The event will focus on the technical advantages of these programming languages, the tools and support available for academics, as well provide an insight into their academic and industrial use through real-life case studies.

Schedule

- 13:30-14:00 Arrival
- 14:00-14:50 "What's New in the World of Ada", Robert Dewar, AdaCore, New York, USA
- 14:50-15:20 "Ada in Industry, an Experience Report", Philippe Waroquiers, EUROCONTROL/CFMU, Brussels, Belgium
- 15:20-15:40 Break
- 15:40-16:10 "Ada in Research and Education, an Experience Report", Erhard Plödereder, University Stuttgart, Germany
- 16:10-17:00 "SPARK - The Libre Language and Toolset for High-Assurance Software", Rod Chapman, Altran Praxis, Bath, UK
- 17:00-18:00 Networking drink

All presentations will be in English.

Presentations

"What's New in the World of Ada"

Robert Dewar, AdaCore, New York, USA

This talk will briefly review the history and main features of Ada, its usage in academic and industrial projects, and will then cover new developments in the Ada language and Ada language tools. New features of Ada 2012 will be discussed as well as the current status of their implementation in GNAT. The talk will also discuss interesting new tools that are available for Ada development, including CodePeer, the new static analysis system being developed jointly by AdaCore and SofCheck, and Couverture, a novel approach to coverage analysis, suitable for both certified critical systems, and mainstream application development.

Robert Dewar is co-founder, President and CEO of AdaCore and is a Professor of Computer Science at the Courant Institute of New York University. He has been involved with Ada for over 20 years and, as co-director of both the Ada-Ed projects and the GNAT project, led the team that developed the first validated Ada compiler at NYU. Robert was one of the authors of the requirements document for the Ada revision, and served as a distinguished reviewer for both Ada 83 and Ada 95. He has co-authored several renowned compilers including the SPITBOL (SNOBOL) compiler, the Realia COBOL compiler for the PC (now marketed by Computer Associates), and the Alsys Ada compiler. He has also written several real time operating systems for Honeywell Inc. Among his many publications, Robert is a principal author (with Professor Edmond Schonberg) of GNAT, the GNU Ada Compiler. A talented public speaker, he is frequently invited to share his thoughts in public on computers and on open-source software.

"Ada in Industry, an Experience Report"

Philippe Waroquiers, EUROCONTROL/ CFMU, Brussels, Belgium

The presentation will give details about how Ada is used at the CFMU to develop ETFMS (Flow Management system) and IFPS (Flight Plan processing system). IFPS processes all the flight plans for of the flights departing from, landing in, or crossing Europe. ETFMS balances the traffic load with the capacity, ensuring an efficient usage of the airspace capacity while maintaining safety.

Philippe Waroquiers works in the Engineering division of EUROCONTROL/CFMU. The CFMU (Central Flow Management Unit) is the operational unit of EUROCONTROL, the European Organization for the Safety of Air Navigation. Philippe is involved in the functional specification, architecture and development of its mission critical systems.

"Ada in Research and Education, an Experience Report"

Erhard Plödereder, University Stuttgart, Germany

The University of Stuttgart uses Ada as the programming language of choice for the introductory courses in Computer Science and Software Engineering. The talk will expand on the reasons for this decision and the discussions about it. It will attempt to separate winning arguments from the grist of many. It will also address issues on the road to teaching computer science students a general understanding of programming languages, in order to enable them to pick up future languages easily and without prejudice. Finally, a large on-going research project using Ada will be briefly described.

Prof. Dr. Erhard Plödereder is Head of the Department of Programming Languages and Compilers at the University of Stuttgart, Germany. Presently he is also serving as Dean of the Faculty of Computer Science, Electrical Engineering and Information Technology. A former chair of IFIP WG2.4 and of several ISO Rapporteur Groups, Ada-Europe President, and long-term researcher in program analysis, he has a keen interest in programming languages, their strengths and weaknesses, as a teacher, a user, and a researcher.

"SPARK - The Libre Language and Toolset for High-Assurance Software"

Rod Chapman, Altran Praxis, Bath, UK

This presentation introduces SPARK - a language specifically designed to support the development and verification of high-assurance software. This presentation covers the concepts behind SPARK, the language design and the capabilities of the verification tools. It will also cover the uses of SPARK in teaching software engineering and will look at current and potential research topics for the academic community, as well as recent and on-going industrial projects.

The presenter will be Dr. Roderick Chapman of Altran Praxis. Rod has been involved with the design of both safety- and security-critical software with Praxis for many years, including significant contributions to many of Praxis' key-note projects such as SHOLIS, MULTOS CA, Tokeneer, and the development of the SPARK language and verification tools. Rod is a well-known conference speaker. He has presented papers, tutorials and workshops at many international events including SSTC, NSA HCSS, and ACM SIGAda. He was the opening key-note speaker at Ada Europe 2006. Rod is a Chartered Engineer, a Fellow of the BCS, and an SEI-certified PSP Instructor.

Participation

Attendance is free, but registration is necessary.

To register, please provide your name, email address and affiliation, either by email to <adaspark2010@cs.kuleuven.be>, or via the web form at <http://distrinet.cs.kuleuven.be/events/AdaEvent/registration.php>.

For directions to the Computer Science Department of the K.U.Leuven, see <http://distrinet.cs.kuleuven.be/events/AdaEvent/route.html>.

Please circulate among your contacts in education and research who may be in the neighborhood at that time, or live or work close-by.

From: Dirk Craeynest <Dirk.Craeynest@cs.kuleuven.be>
Date: Tue, 2 Mar 2010 10:24:00 +0100 CET
Subject: [ada-france] Presentations on-line - Ada&SPARK for Education&Research
Mailing list: ada-france.org

[…]

All presentations at this half-day Seminar, held at the university in Leuven last week, are available now on the web sites of Ada-Belgium and the Distrinet research group: see URLs above.

For each presentation a PDF version can be downloaded; some are also available in other formats (ODP or PPTX):

- "What's New in the World of Ada", Robert Dewar, AdaCore, New York, USA
- "Ada in Industry, an Experience Report", Philippe Waroquiers, EUROCONTROL/CFMU, Brussels, Belgium
- "Ada in Research and Education, an Experience Report", Erhard Plödereder, University Stuttgart, Germany
- "SPARK - The Libre Language and Toolset for High-Assurance Software", Rod Chapman, Altran Praxis, Bath, UK

The seminar went very well, with a good mix of people from academia, research and industry, not only from Belgium, but also from the U.K., the Netherlands, Germany and France. Many participants told us they really appreciated the event; some of the feedback we received:

"… was really interesting being there."

"… a most interesting Ada meeting at the K.U.Leuven."

"I did find all the presentations very interesting as well as the informal discussions with the people present."

Thanks again to all presenters for their collaboration, to AdaCore for the many Ada books we handed out, to the participants for their interest, as well as for all the efforts many speakers and participants went through to come to Leuven at a time when both European high speed rail and air travel was disrupted.

Enjoy the on-line presentations!

[find the presentations at http://distrinet.cs.kuleuven.be/events/AdaEvent/abstracts.html —mp]

Ada-related Resources

Ada, data structures and Big-O notation

From: Rick Duley <rickduley@gmail.com>
Date: Wed, 17 Feb 2010 15:20:02 -0800 PST
Subject: Ada-based Primer in Big-Oh Notation
Newsgroups: comp.lang.ada

In preparing a tutorial on binary trees I am looking for an introductory article on Big-Oh Notation. The NIST article at http://www.itl.nist.gov/div897/sqg/dads/HTML/bigOnotation.html is, as expected, excellent - if you are comfortable with mathematics.

The students at whom this tutorial is aimed struggle with mathematics, and would be much more comfortable with Rob Bell's article at http://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/.

Unfortunately, the examples in Rob's article are not in Ada. Can anyone point me to a tutorial at a similar level to Rob's in which Ada examples are used?

From: John McCormick <mccormick@cs.uni.edu>
Date: Thu, 18 Feb 2010 06:57:40 -0800 PST
Subject: Re: Ada-based Primer in Big-Oh Notation
Newsgroups: comp.lang.ada

[…]

Section 5.4 of my book, Ada Plus Data Structures (Dale and McCormick, 2007), discusses various ways to compare implementations. That discussion includes an introduction to Big-O and textural descriptions of the common orders of magnitude. After this introduction, the Big-O of each algorithm developed in the book is discussed. As the material is spread out throughout the book, organizing it into a single tutorial would take some effort.

This is our students' first exposure to Big-O. They see it again in several other classes. It seems to take several spirals through the concept adding more details each time for Big-O to sink in.

It would not be difficult to translate Rob's examples into Ada.

[…]

Ada-related Tools

Simple Components for Ada v3.7

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 26 Dec 2009 12:58:51 +0100
Subject: ANN: Simple components for Ada v3.7 released
Newsgroups: comp.lang.ada

The current version provides implementations of smart pointers, sets, maps, directed graphs, directed weighted graphs, stacks, tables, string editing, unbounded arrays, expression analyzers, lock-free data structures, synchronization primitives (events, race condition free pulse events, arrays of events, re-entrant mutexes, deadlock-free arrays of mutexes), pseudo-random non-repeating numbers, symmetric encoding and decoding, IEEE 754 representations support; strings editing and tables management.

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

The focus of this release is an implementation of directed graphs (this includes trees). The nodes of directed graphs are supported of any type including limited and indefinite types. The directed weighted graphs are provided as well. The weights can be of any indefinite type. Use of weighted graphs is illustrated on the example of the suffix tree.

[see also "Simple Components 3.4, 3.5 and 3.6" in AUJ 30-3 (Sep 2009), p.142 —mp]

On the status of AdaCL and synchronization of containers in Ada

From: Michael Rohan <michael@zanyblue.com>
Date: Sun, 21 Feb 2010 14:09:00 -0800 PST
Subject: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

[…]

I remembered a previous post on a command line handling library, searched and found AdaCL. The Orto package seems to be something, on the surface, that might be really useful but it depends on the Charles library which appears to be a pre-Ada2005 container library. The date associated with the latest release on the download page is 2007-12-09. Is this project "dead"?

From: Björn Persson <bjorn@xn--rombobjrn-67a.se>
Date: Mon, 22 Feb 2010 02:51 +0100
Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

[…]

Orto isn't dead, but it's dormant while I'm focusing on other projects. I can't speak for Charles or for the rest of AdaCL, but Orto and EAstrings are "alive" and if you find any bugs in them then I want to hear about it.

I had intended to switch from Charles to Ada.Containers, but I changed my mind when I learned that Ada.Containers can't even be read by multiple tasks at once. As far as I know the Charles containers can be accessed by multiple tasks as long as none of them modifies the container. The internal data structure in Orto? Which makes use of Charles? isn't modified after the parsing is done, so if you parse the command line before you start any additional tasks, you can then freely look up parameters in multiple tasks.

I believe Charles still works, but I could probably use some other container library if there is a compelling reason to stop using Charles.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Wed, 24 Feb 2010 17:48:18 -0600
Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

[…]

For the record, we've studied this several times and have always concluded that hidden synchronization is dangerous. That is, synchronization should be explicit. Beyond that, it is impossible to come up with a reasonable definition of what should be locked -- it really depends on the use of the containers.

In the case of the containers, task safety of iterators and similar features is something that defies a reasonable definition. The problem gets worse if you include features used together (such as using First and Next to create a loop of some sort). We'd probably need to make the locks visible in order for them to be useful.

It's easy to wrap container operations in a protected object, and that is always allowed (such operations are not potentially blocking). That allows tailoring the locking for the actual usage, and even hiding the actual container to prevent abuse.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Thu, 25 Feb 2010 10:22:16 +0100
Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

[…]

Indeed, aren't there advantages in hiding data stores like Ada.Containers behind some facade, whether the use is sequential or not: everything else always smells of exposed internal data structures, or lack of abstraction.

There need to be good reasons for using List, Set, etc. as is, I think. Just like there should be good reasons to expose arrays.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 25 Feb 2010 07:23:22 -0500

Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

> For the record, we've studied this several times and have always concluded that hidden synchronization is dangerous. […]

I agree with this, but I think the OP was implying that you needed locking even for read-only access of Ada.Containers from multiple tasks; is that true? I don't see why it should be; each task declares its own cursors, which don't interfere with each other.

Of course, there's nothing enforcing the read-only, so this is not very safe.

From: Alex R. Mosteo <alejandro@mosteo.com>
Date: Thu, 25 Feb 2010 15:16:17 +0100
Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

[…]

I think that the downward closure subprograms update some flags inside the container. E.g., in GNAT doubly linked lists, within the Query_Element:

```
procedure Query_Element
  (Position : Cursor;
   Process : not null access procedure
            (Element : Element_Type));
```

Pretty read-only, it seems, but inside you find:

```
declare
  C : List renames
     Position.Container.all'
     Unrestricted_Access.all;
  B : Natural renames C.Busy;
  L : Natural renames C.Lock;

 begin
  B := B + 1;
  L := L + 1;
(…)
```

So, basically, yes, even certain read-only uses are not thread-safe (Element would be, at least in GNAT implementation).

Never thought of this before, but even wrapping a call to that in a protected function would be dangerous, since protected functions are concurrent? And that's a procedure with only "in" arguments, which would be callable from such a function. Am I right here?

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Thu, 25 Feb 2010 12:19:39 -0800 PST
Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

I believe you are: LRM 9.5.1(1), "protected functions provide concurrent read-only access to the data".

My normal mode of operation is to wrap the use of the Container (OK, Booch Component!) in a package and use a lock to ensure single-threaded access, with the Container outside any PO.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Wed, 24 Feb 2010 20:54:49 +0100
Subject: Re: Status of AdaCL: Ada Class Library
Newsgroups: comp.lang.ada

> The date associated with the latest release on the download page is 2007-12-09. Is this project "dead"?

Not dead as such - just not extended for a while. But if you find a bug add it to the tracker and I'll have a look.

GNAT 4.3.4 for OpenBSD/amd64

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Wed, 27 Jan 2010 23:57:20 +0200
Subject: GNAT 4.3.4 binaries for OpenBSD/amd64
Newsgroups: comp.lang.ada

[…] I recently got an amd64(x86_64) system and put OpenBSD there. There was no native GNAT for OpenBSD/amd64, so I had to create one by cross-compiling the compiler on i386.

To spare others from the same operation, I made my GCC and GNAT packages available at

http://tkoskine.iki.fi/openbsd/amd64/index.html

They work on the latest OpenBSD 4.6-current (or 4.7-beta) and installation happens with pkg_add:

pkg_add http://tkoskine.iki.fi/openbsd/amd64/gcc-4.3.4.tgz

pkg_add http://tkoskine.iki.fi/openbsd/amd64/gnat-4.3.4.tgz

I am sure there are some bugs left, but the packages can be used for bootstrapping if nothing else.

Makefile rules ("port" in OpenBSD terms) to create packages can be found from GNUAda repository:

http://gnuada.svn.sf.net/viewvc/gnuada/trunk/OpenBSD/current/lang/gcc/

AVR-Ada 1.1

From: Rolf Ebert <rolf.ebert_nospam_@gmx.net>
Date: Sat, 27 Feb 2010 08:07:53 -0800 PST
Subject: [Ann] new AVR-Ada release
Newsgroups: comp.lang.ada

We are proud to announce a new release of AVR-Ada, one of the first GCC-based Ada compilers targeting 8-bit microcontrollers.

See the project documentation at

http://avr-ada.sourceforge.net/

This is a binary release for Windows and a source release for other platforms. That means you get a working compiler on Windows and patches, instructions, and a build script for building your own cross compiler for other platforms (Linux). The run time system, the support packages, and the sample programs are included in both distributions. If you want to use AVR-Ada on Windows you first have to install WinAVR-20100110.

The download is available at the green button on page

http://sourceforge.net/projects/avr-ada

If you have difficulties in building or using the compiler or you want to chat about a project, please join and use the mailing list at

http://lists.sourceforge.net/mailman/listinfo/avr-ada-devel.

Status

The AVR-Ada project makes available the gcc based Ada compiler GNAT for the AVR 8-bit microcontrollers (it does not work for AVR32).

More specifically the project provides:

- a GNAT compiler based on the existing AVR and Ada support in GCC
- a minimalistic Ada runtime system
- an extensiv and useful AVR specific support library
- support packages for accessing LCDs, Dallas' 1-wire sensors, or the Sensirion humidity and temperature sensors.

The current release of AVR-Ada is V1.1. It is based on gcc-4.3.3.

The Ada run time system (RTS) is still not even a *run* time system. It is more a compile time system :-). Most files in the RTS are only needed at compile time. As a consequence we don't have support for exceptions nor for tasking (multithreading).

There is extensive AVR specific support. Type and interface definitions, timing routines, eeprom access, UART, and most importantly the necessary port and interrupt definitions for most AVR parts.

Some sample programs in the apps/ directory show how to use the compiler and the library.

Some applications that had been built using AVR-Ada:

- a data logger for a weather station
- a closed loop heating control system
- an astronomical "GoTo" mount for a telescope on an AVR90USB128
- a small robot based on the Asuro platform

- a limited IP stack with ARP, ICMP and UDP (no TCP yet)
- sample programs for the very popular Arduino platform

You are invited to have fun with AVR-Ada. For professional and safety critical applications ask Adacore. They offer their well known support also for a AVR cross compiler.

[see also "AVR-Ada 1.0" in AUJ 30-1 (Mar 2009), p.7 —mp]

GCC 4.4.2 for Mac OS X

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Tue, 08 Dec 2009 18:30:56 +0100
Subject: Announce: gcc 4.4.2 for Mac OS X
Newsgroups: comp.lang.ada

[…]

I just released GCC 4.4.2 for Mac OS X. You find it here:

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

[…]

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Wed, 09 Dec 2009 10:27:59 +0100
Subject: Re: Announce: gcc 4.4.2 for Mac OS X
Newsgroups: comp.lang.ada

> […] Which version of MacOS X?

Compiled on Leopard. But from what I know the compiler runs on SnowLeopard as well - but tests are not finished yet.

From: Jerry Bauck <lanceboyle@qwest.net>
Date: Mon, 14 Dec 2009 16:36:06 -0800 PST
Subject: Re: Announce: gcc 4.4.2 for Mac OS X
Newsgroups: comp.lang.ada

[…]

Thanks, Martin. Can you let us know when someone gets a positive (or negative) result on Snow Leopard? As you probably know, the MacAda group is having trouble getting a Snow Leopard compiler running.

From: Bill Findlay <yaldnif.w@blueyonder.co.uk>
Date: Tue, 15 Dec 2009 00:48:35 +0000
Subject: Re: Announce: gcc 4.4.2 for Mac OS X
Newsgroups: comp.lang.ada

[…]

Like the other compiler I've tried on Snow Leopard, it does not support 64-bit mode, throwing up spurious data alignment errors; and it generates seriously incorrect code for inlined subprograms, even in 32-bit mode, so that optimized binaries are unusable.

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Tue, 15 Dec 2009 09:21:23 +0100
Subject: Re: Announce: gcc 4.4.2 for Mac OS X
Newsgroups: comp.lang.ada

[…]

Currently the compile fails. Probably because it is a 32 -> 64 bit cross compile. I wonder it the initial compile need to be done by someone with access to GNAT Pro/GAP x86_64.darwin10.

> As you probably know, the MacAda group is having trouble getting a Snow Leopard compiler running.

No I did not. But I guessed it. Anyway, my build script is on-line and anybody is invited to improve on it:

http://trac.macports.org/browser/trunk/dports/lang/gnat-gcc/Portfile

GNAT GPL 2009 for Mac OS X (Snow Leopard)

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Tue, 22 Dec 2009 04:09:09 -0800 PST
Subject: GNAT GPL 2009 for Mac OS X (Snow Leopard)
Newsgroups: comp.lang.ada

I've just released a 32-bit build of GNAT GPL 2009 for Snow Leopard; find it at

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

(navigate to GNAT_GPL Mac OS X/2009-snow-leopard-i386).

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Tue, 22 Dec 2009 09:10:35 -0800 PST
Subject: Re: GNAT GPL 2009 for Mac OS X (Snow Leopard)
Newsgroups: comp.lang.ada

> […] Many thanks -- I'll try it ASAP. Any chance of a 64-bit build?

Will have a go, but it's a cross-build so may take some getting my head round!

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Fri, 25 Dec 2009 09:52:43 -0800 PST
Subject: Re: GNAT GPL 2009 for Mac OS X (Snow Leopard)
Newsgroups: comp.lang.ada

Not actually a cross build, which would be specified by configuring with --target=3Dx86_64-apple-darwin10.2.0; I don't know what the technical name is, but configuring with --build=3Dx86_64-apple-darwin10.2.0 does the trick.

Unfortunately the built compiler won't handle exceptions; a severe failing. Perhaps Apple have changed the OS features that GNAT uses to support exception handling? (I tried configuring with --enable-sjlj-exceptions, no joy).

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Wed, 23 Dec 2009 15:38:48 -0800 PST
Subject: Re: GNAT GPL 2009 for Mac OS X (Snow Leopard)
Newsgroups: comp.lang.ada

> […] I thought OS X was officially supported by AdaCore. From the viewpoint of a casual bystander, it looks like someone is asleep at the wheel.

[…]

I have taken the source code provided by AdaCore in GNAT GPL 2009 about 9 months before Snow Leopard was released to the world and rebuilt it with one change (caused by Apple's removal of sigreturn()).

I don't know any of AdaCore's customers who are supported on Mac OS X so I can't say what AdaCore's advice to them has been; possibly those customers who mustmustmust upgrade have been provided with wavefronts.

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Thu, 24 Dec 2009 07:01:31 -0800 PST
Subject: Re: GNAT GPL 2009 for Mac OS X (Snow Leopard)
Newsgroups: comp.lang.ada

[…]

The cross-build (I think I may be wrong about that; perhaps there's an 'architecture' switch?) eventually produced a compiler. Unfortunately it wouldn't process the simplest exception properly (SIGABRT).

I then had the idea, since the problem that stops building is that init.c calls sigreturn() which isn't present in Snow Leopard, why not fake up our own sigreturn() that does exactly what the patch did, then link against that?

This turned out to work as hoped, result being that we can use Apple's GNAT-GPL-2009 on Snow Leopard; see http://sourceforge.net/projects/gnuada/files/(navigate to GNAT_GPL Mac OS X/2009-snow-leopard, get sigreturn.tar.bz2).

Sadly, the failure to handle exceptions with the 64-bit compiler is unchanged (the 32-bit compiler is OK):

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Raiser is
begin
  begin
    raise Constraint_Error;
  exception
    when Constraint_Error =>
      Put_Line ("CE raised.");
  end;
end Raiser;
```

[…]

Visual Ada Developer 7.3

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Sat, 19 Dec 2009 03:07:09 -0800 PST
Subject: Ann: Visual Ada Developer VAD 7.3
Newsgroups: comp.lang.ada

Visual Ada Developer (VAD) 7.3 is now available at

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

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

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

VAD 7.3 Common description.

1. VAD (Visual Ada Developer) is a Tcl/Tk oriented Ada-95(TCL) GUI builder portable to difference platforms, such as Windows NT/Vista/7,Unix (Linux), and Mac.

You may use it as IDE for any Ada-95(C,C++,TCL) project.

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

VAD 7.3 was tested in Windows 32bit and 64bit and Linux x86-64 Kubuntu 9.10

2. Used software

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

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

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

Warning! VAD 7.3 has two realization, for Tcl/Tk 8.5.x and Tcl/Tk 8.6.x , you need to install and test Tcl/Tk first.

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

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

[…]

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

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

[see also "Visual Ada Developer 7.2" in AUJ 30-3 (Sep 2009), p.143 —mp]

GWenerator 0.99

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sun, 28 Feb 2010 09:14:48 -0800 PST
Subject: Ann: GWenerator 0.99
Newsgroups: comp.lang.ada

[…]

There is a new release of the GWenerator.

What's new:

- Ada background or on-demand build
- A few improvements on supported controls

Download: http://sourceforge.net/projects/gnavi/

What is the GWenerator ?

With GWenerator you can design Graphical User Interfaces with various 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, as soon as a new design has been saved. On request, it produces also a test application with all dialogs. Optionally, it launches a new Ada build, automatically or on request.

The command-line equivalent tool, rc2gw, does the code generation on request.

Unlike some other GUI libraries, GWindows is Windows-only (at least currently) - that the "minus" side. So it is intended to target Windows-centric environments.

On the "plus" side, a GWindows application can hold in a single executable. A priori, there is no need to provide any run-time framework, toolkit, dll, installation wizard or to worry about version conflicts, paths, admin rights. This makes the deployment of an application trivial or of minimal complexity.

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

[see also "GWenerator 0.97 and 0.975" in AUJ 30-3 (Sep 2009), p.143 —mp]

QtAda 2.4 and relation with QtAda by Godunko et al.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 6 Dec 2009 11:15:35 +0100
Subject: Announce : QtAda version 2.4
Newsgroups: comp.lang.ada

Leonid Dulman asked me to announce this:

QtAda is an Ada-95(05) interface to Qt4 graphics library Qt version 4.6.0 open source and qt4c.dll(libqt4c.so) built with Microsoft Visual Studio 2008 (2010 beta) in Windows MINGW GCC Windows compiler and GCC in Linux.

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

Qtada it's composed by 9245 procedures and functions, distributed in 256 packages. It supports GUI, SQL, Multimedia, Web, Net and many others thinks.

QtAda for Windows and Linux (Unix) is available from

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

From: Dirk Heinrichs <dirk.heinrichs@online.de>
Date: Sun, 06 Dec 2009 19:59:47 +0100
Subject: Re: Announce : QtAda version 2.4
Newsgroups: comp.lang.ada

[…]

Although I appreciate this very much, I think it's not a good idea to have two different efforts to provide the same interface. Wouldn't it be better to join forces?

It could be completed in less time and would have a larger user base which also finds possible bugs much faster so that in the end the outcome would be an even better interface.

Maybe joining both projects would even result in a larger developer group who work on it (and maybe also on a KDE4 interface).

[…]

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Mon, 07 Dec 2009 11:52:44 +0100
Subject: Re: Announce : QtAda version 2.4
Newsgroups: comp.lang.ada

[…]

OTOH, competition, if possible, may provide for different approaches to writing Qt programs in Ada. One or the other approach might more portable, or be better in some situation. If I'm not mistaken, Qt4Ada (by Leonid Dulman) can be used with Ada 95 compilers; it does not depend on so many Ada 2005 anonymous access types as QtAda does. (Can the latter be compiled at all with compilers other than a recent GNAT?)

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

http://www.qtada.com/

http://qt4ada.sourceforge.net/ (by Yves Bailly, I don't know whether this is an active project)

Licenses might differ, too, I think.

From: Dirk Heinrichs <dirk.heinrichs@online.de>
Date: Mon, 07 Dec 2009 19:01:35 +0100
Subject: Re: Announce : QtAda version 2.4

Newsgroups: comp.lang.ada

> […] OTOH, competition, if possible, may provide for different approaches to writing Qt programs in Ada.

Usually I would agree to this argument. However, we are talking about projects to provide an Ada interface for a GUI toolkit written in another language, not about a project to actually _develop_ a GUI toolkit.

> One or the other approach might more portable, or be better in some situation. If I'm not mistaken, Qt4Ada (by Leonid Dulman) can be used with Ada 95 compilers; it does not depend on so many Ada 2005 anonymous access types as QtAda does.

Any reason why all of this can't be achieved by a single project?

> (Can the latter be compiled at all with compilers other than a recent GNAT?)

That's IMHO another reason to have only one project. At least two persons have used much of their time to achieve more or less the same goal, which is really great, but in the end both projects lack important functionality which is present in the other.

[…]

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sat, 12 Dec 2009 15:43:23 +0100
Subject: Re: Announce : QtAda version 2.4
Newsgroups: comp.lang.ada

[…]

I believe there's a license discrepancy, but QtAda's licensing appears to be rather unclear.

From: Vadim Godunko <vgodunko@gmail.com>
Date: Thu, 17 Dec 2009 00:43:46 -0800 PST
Subject: Re: Announce : QtAda version 2.4
Newsgroups: comp.lang.ada

[…]

QtAda's licensing is pretty clear: QtAda GPL Edition available under GPL license, QtAda Professional edition covered by GNAT Modified GPL.

Someone can ask why we don't use LGPL as Nokia do? The answer is very simple - Nokia licenses Qt Open Source Edition under GPL/LGPL with minor exception, to protect youself from free use Qt in commercial closed source projects (You must provide all your source code on Nokia's request).

[see also "QtAda 3.0 and 2.2" and "QtAda 2.1.1 for Qt 4.5.2" in AUJ 30-3 (Sep 2009), p.145 —mp]

OS2Ada 1.0

From: Leonid Dulman <leonid.dulman@gmail.com>
Date: Sat, 6 Feb 2010 23:14:27 -0800 PST
Subject: Announce : OS2Ada version 1.0
Newsgroups: comp.lang.ada

OS2Ada is an Ada-95(05) API and PM interface to IBM eComStation (Os/2) operating system. OS2Ada is available at http://users1.jabry.com/adastudio/index.html

I am open for any forms of cooperation. Thanks for any help.

OpenToken 4.0a

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Sat, 27 Feb 2010 10:29:10 -0500
Subject: OpenToken 4.0a released
Newsgroups: comp.lang.ada

I've released a new version of OpenToken; 4.0a

It will be in Debian Squeeze, and in Debian testing before that.

See http://www.stephe-leake.org/ada/opentoken.html to download source and see full list of changes.

Major changes:

Lookahead and backtracking is supported in recursive descent parsers. This can generate horribly inefficient parsers if you are not careful.

Fixed major bug in LALR parser generator related to which production gets the accept action. This bug made many small grammars unworkable; now they all work.

Syntax errors reported by LR and recursive descent parsers include the list of expected tokens.

The examples have been improved to more clearly demonstrate the differences and similarities between LR parsing and recursive descent parsing with OpenToken.

There are new examples of recursive descent parsing, showing that naive grammars can work, if inefficiently.

The OpenToken.Token.List_Mixin, .Sequence_Mixin, .Selection_Mixin now specify actions via procedure pointers at run-time, rather than via overloaded procedures. This significantly simplifies specifying recursive descent grammars.

Language_Lexers.HTML_Lexer supports the <pre> tag; the contents are treated as a comment.

[see also "OpenToken 3.1a" in AUJ 30-4 (Dec 2009), p.208 —mp]

Alog 0.3

From: Adrian-Ken Rueegsegger <ken@codelabs.ch>
Date: Sat, 19 Dec 2009 22:03:50 +0100
Subject: Announce: Alog 0.3 released
Newsgroups: comp.lang.ada

We are proud to announce the release of Alog version 0.3.

Alog is a stackable logging framework for Ada. It aims to be straight forward to use and easily extendable. It provides support for various logger types, log facilities, loglevel policies and message transformations.

Further information about Alog can be found on the project website [1].

[…]

[1] - http://www.nongnu.org/alog/

Strings Edit for Ada v2.4

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 26 Dec 2009 12:50:47 +0100
Subject: ANN: Strings edit v2.4 released
Newsgroups: comp.lang.ada

Provides string editing:

1. Integer numbers (generic, package Integer_Edit);
2. Integer sub- and superscript numbers;
3. Floating-point numbers (generic, package Float_Edit);
4. Roman numbers (the type Roman);
5. Strings;
6. Ada-style quoted strings;
7. UTF-8 encoded strings;
8. Unicode maps and sets;
9. Wildcard pattern matching.

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

Changes to the previous version:

1. Bug fix in the function Is_Prefix, which uses character maps.

[see also "Strings Edit for Ada v2.3" in AUJ 30-4 (Dec 2009), p.208 —mp]

Safe Pointers

From: Cristoph Grein <christoph.grein@eurocopter.com>
Date: Wed, 30 Dec 2009 09:49:11 -0800 PST
Subject: Safe Pointers
Newsgroups: comp.lang.ada

Safe Pointers provide a reference counting facility to deal with access types without compromising safety, i.e. safe pointers clean up behind themselves properly; no dangling references can ever be produced.

Ada 95 and Ada 2005 differ enough so that two versions exist. A new Ada 2005 version is ready:

http://www.christ-usch-grein.homepage.t-online.de/Ada/Safe_Pointers.html

Zip-Ada v.38

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sun, 28 Feb 2010 09:09:16 -0800 PST

Subject: Ann: Zip-Ada v.38
Newsgroups: comp.lang.ada

Hello!

There is a new Zip-Ada release.

What's new:

- The "-fast_dec" and "-rand_stable" options were added for ReZip.
- Zip.Create is significantly faster on a large number of files (~1000 or multiples).

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

[…]

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

Excel Writer v.05

From: Gautier de Montmollin <gdemont@users.sourceforge.net>
Date: Sun, 28 Feb 2010 09:17:04 -0800 PST
Subject: Ann: Excel Writer v.05
Newsgroups: comp.lang.ada

[…]

There is a new release of Excel Writer.

What's new:

- Some new built-in Excel numeric formats
- Usage of Excel_Out instead of Ada.Text_IO (and conversely) made a bit easier
- Should compile on any Ada 95+ compiler (a non-compliant detail was fixed)

Download:

http://excel-writer.sf.net

Ada-related Products

AdaCore — CodePeer Tool

From: AdaCore Press Center
Date: Tue, 12 Jan 2010
Subject: AdaCore Launches CodePeer Tool
URL: http://www.adacore.com/2010/01/12/codepeer-launch/

Automated code review assistant helps eliminate bugs and vulnerabilities

NEW YORK and PARIS, January 12, 2010 – AdaCore, a leading supplier of Ada development tools and support services, today announced the release of CodePeer, a source code analysis tool that detects run-time and logic errors in Ada programs. Serving as an efficient and accurate code reviewer, CodePeer identifies constructs that are likely to lead to run-time errors such as buffer overflows, and it flags legal but suspect code typical of logic errors. Going well beyond the capabilities of typical static analysis tools, CodePeer also produces a detailed analysis of each subprogram, including pre- and postconditions. Such an analysis makes it easier to find potential bugs and vulnerabilities early: if the implicit specification deduced by CodePeer does not match the component's requirements, a reviewer is alerted immediately to a likely logic error. CodePeer can be used both during system development − to prevent errors from being introduced or as part of a systematic code review process to dramatically increase the efficiency of human review − and retrospectively on existing code, to detect and remove latent bugs.

Developed by AdaCore in partnership with SofCheck, Inc., CodePeer can be used either as a standalone tool or fully integrated into the GNAT Pro Ada development environment. It is highly flexible, with performance that can be tuned based on the memory and speed available on the developer's machine, and can efficiently exploit multi-core CPUs. CodePeer can be run on partially complete programs; it does not require stubs or drivers.

CodePeer analyzes programs for a wide range of flaws including use of uninitialized data, pointer misuse, buffer overflow, numeric overflow, division by zero, dead code, and concurrency faults (race conditions). These sorts of errors can be difficult and expensive to detect and correct with conventional debugging, but CodePeer identifies them statically, without running the program, and determines not only where the failure could occur, but identifies where the bad values originate, be it within the current subprogram or from some distant subprogram that reached the point of failure through a series of calls. CodePeer also looks for code that, although syntactically and semantically correct, is performing a suspect computation, such as an assignment to a variable that is never subsequently referenced, or a conditional test that always evaluates to the same true or false value.

"Even the best programmers using the best programming languages will sometimes make mistakes," said Robert Dewar, President and CEO of AdaCore. "The key is to detect and correct errors early, and, thanks to our partnership with SofCheck, CodePeer is now available for precisely that purpose. We expect this tool to be especially valuable to our customers with safety-critical or high-security requirements, since CodePeer can identify potential hazards and vulnerabilities."

Internally CodePeer uses static control-flow, data-flow, and value propagation techniques to identify possible errors. It mathematically analyzes every line of code without executing the program, considering all combinations of program input across all paths within the program. It automatically generates both human-readable and machine-readable component specifications in the form of preconditions, postconditions, inputs, outputs, and heap allocations, which along with the error messages can be displayed graphically or as in-line comments in the source code listing to help immediately pinpoint the root cause of any defect. In a multi-threaded system CodePeer identifies where race conditions might occur. To increase performance and usability it internally maintains a historical error database, which allows it to highlight just the new coding problems and to track trends across multiple analyses.

"The technology underlying CodePeer was developed over many years of work on highly optimizing compilers," noted Tucker Taft, Founder and CTO of SofCheck, "but now we are taking all the information the compiler was using internally for its own optimization purposes, augmenting it with advanced whole-program analyses, and presenting it in a way that allows the programmer to fix their software before it breaks."

Webinar

A webinar introducing CodePeer will be presented on March 9, 2010 at 11:00 am (EST) / 5:00 pm (GMT). For more information, or to register, please visit

http://www.adacore.com/home/gnatpro/webinars/

Pricing and Availability

CodePeer is immediately available. Please contact AdaCore (sales@adacore.com) for information on pricing and supported configurations.

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 worldwide 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, railway systems and medical devices, and in security-sensitive domains such as financial services.

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

www.adacore.com

AdaCore — GNAT Pro support for PikeOS

From: AdaCore Press Center
Date: Tue, 26 Jan 2010
Subject: New Safety-Critical RTOS Platform for GNAT Pro High-Integrity Edition
URL: http://www.adacore.com/2010/01/26/support-for-pike-os/

SANTA CLARA, NEW YORK and PARIS, January 26, 2010 – Real-Time & Embedded Computing Conferences (RTECC) – AdaCore and SYSGO today further strengthened their partnership by announcing the release of the GNAT Pro High-Integrity Edition for DO-178B toolset targeting SYSGO's PikeOS platform, a safety-critical real-time operating system (RTOS). GNAT Pro has been ported to this platform in direct response to growing customer demand. This new development follows GNAT Pro support for ELinOS™.

PikeOS has been gaining popularity with avionics developers as a next-generation RTOS for ARINC-653; with Ada's and AdaCore's proven track record in the safety-critical market, including avionics and rail transportation, GNAT Pro is a natural match. GNAT Pro High-Integrity Edition for PikeOS comes with the Zero Footprint (ZFP) and Ravenscar run-time libraries. The ZFP run-time library of GNAT Pro High-Integrity Edition for DO-178B has been used on multiple safety-critical projects and allows simple sequential application development that eases certification to DO-178B Level A. The Ravenscar run-time library adds support for deterministic multi-tasking. AdaCore provides both run-times libraries to allow developers to choose the one best suited to their application requirements.

PikeOS provides an embedded platform where multiple virtual machines can run simultaneously in a secure environment. The Safe and Secure Virtualization (SSV) technology allows multiple operating system APIs, called "Personalities", to run concurrently on one machine, for example an ARINC-653 application together with Linux. GNAT Pro High-Integrity Edition for PikeOS provides ZFP and Ravenscar run-times for both PikeOS Native and APEX (ARINC 653) personalities.

The PikeOS microkernel architecture supports a range of domains, from cost-sensitive, resource-constrained devices to large, complex systems. Because of its simplicity and compactness, PikeOS is suitable for the most demanding real-time applications. PikeOS is certifiable to safety standards, including DO-178B, IEC 61508 and EN 50128. It is also MILS compliant.

"AdaCore has created an entire family of products to support development of both safety- and security-critical applications with our GNAT Pro High-Integrity Family," said Robert Dewar, AdaCore President and CEO. "We strive to continuously expand the number of options available to our customers for safety- and security-critical development. Support for PikeOS is now the latest RTOS addition to our High-Integrity Family of products."

"With new certifiable run-times and qualifiable tools, AdaCore has been strengthening its offering dedicated to the development of safety-critical systems," said Michaël Friess, AdaCore EU Sales & Business Development Manager. "SYSGO has rapidly gained renown and trust in the safety-critical market. It was a natural choice for AdaCore to expand our GNAT Pro High-Integrity Edition family to the PikeOS platform, foster the fruitful partnership with SYSGO, and provide our joint customers with a flexible and efficient solution."

"Our product PikeOS has one of the fastest growth rates in the sectors requiring safety and security critical applications," declares Jacques Brygier, VP Marketing at SYSGO. "The successful partnership we have built with AdaCore regarding our embedded Linux product ELinOS makes our collaboration in the safety-critical domain even more logical, as both companies have in common an excellent reputation and experience in this very demanding business."

About SYSGO

SYSGO excels in providing operating system technology, middleware, and software services for the real-time and embedded device market. A differentiating capability of SYSGO is the secure and certifiable PikeOS™ paravirtualization operating system which is built upon a small, fast, and safe microkernel and supports the cohabitation of independent operating system personalities on a single platform, including ELinOS™, SYSGO's embedded Linux development environment. SYSGO supports international customers with services for embedded Linux, real-time capabilities and certification for safety-critical applications. Target markets include Aerospace & Defense, Industrial Automation, Automotive, Transportation and Network Infrastructure. SYSGO customers include Airbus, Honeywell, Thales, Daimler, Raytheon, Rheinmetall, Rockwell-Collins, Siemens and Rohde & Schwarz. Today, the company has six facilities in Europe, including Germany, France and The Czech Republic and offers a global distribution and support network, extending to North America and the Pacific Rim. SYSGO, ELinOS and PikeOS are trademarks or registered trademarks of SYSGO AG in Germany and in several other countries all over the world. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries. All other product and service names mentioned are the trademarks of their respective companies.

AdaCore — GNAT Pro 6.3

From: AdaCore Press Center
Date: Tue, 9 Feb 2010
Subject: AdaCore Releases GNAT Pro 6.3
URL: http://www.adacore.com/2010/02/09/gnat-pro-6-3/

BRISTOL, United Kingdom, NEW YORK and PARIS, February 9, 2010 – Safety-Critical Systems Symposium – AdaCore, a leading supplier of Ada development tools and support services, today announced the latest major release of its GNAT Pro Ada development environment. GNAT Pro 6.3 is now available on several new platforms including 64-bit Windows, Windows 7, Mac OS X Snow Leopard, VxWorks MILS, and PikeOS (ARINC 653). The product incorporates performance improvements and a variety of enhancements, many based on customer feedback, and it offers the first implementation of new Ada 2012 features. GNAT Pro 6.3 works with a number of complementary tools, sold separately, including the CodePeer automated code review and validation tool.

"We continually strive to provide our customers with world-class development tools, technical consulting, and product support that exceed their expectations," said Robert Dewar, President and CEO of AdaCore. "GNAT Pro 6.3 demonstrates our commitment to this goal and continues the company's tradition of providing annual major releases for our products."

"With this new release, GNAT Pro supports a wide range of additional platforms, with a particular emphasis on 64 bit architectures," said Cyrille Comar, Managing Director, AdaCore Europe. "This regular adaptation to new environments is critical for many of our customers who work on safety critical applications whose lifespans cross many hardware generations and operating system transitions. Our challenge is to find, every year, more efficient ways of producing and qualifying to the highest standards, an increasing number of tools running on an increasing number of platforms."

New GNAT Pro 6.3 features include:

- Support for 64-bit code generation on Windows
- Support for Windows 7
- Improvements to a number of tools, including:
  - o Pretty printer (gnatpp)

- o Coding standard verifier (GNATcheck)
- o Stack size analyzer (GNATstack)

- Other tool enhancements:
  - o Unused dispatching subprogram elimination (gnatelim)
  - o More flexible project handling
  - o More efficient gnatmake and gprbuild
  - o C and C++ binding generation
- Compiler and debugger enhancements, including:
  - o Many additional warnings
  - o More flexible enabling/disabling of warnings
  - o Faster Unbounded_String implementation
  - o Removal of redundant run-time checks
  - o Support for Ada 2012 conditional expressions
  - o Code generation (speed and size) improvements
  - o More compact debugging information
  - o Improved interfacing with C++

GNAT Pro 6.3 also includes a new version of the GNAT Programming Studio (GPS) IDE, GPS 4.4.1, and an improved download interface in AdaCore's GNAT Tracker support tool.

About GNAT Pro

The GNAT Pro development environment, available on more platforms than any other Ada toolset, is a full-featured, multi-language development environment complete with libraries, bindings and a range of supplementary tools. It provides a natural solution for organizations that need to create reliable, efficient, and maintainable code. GNAT Pro implements all three versions of the Ada language standard – Ada 83, Ada 95, and Ada 2005 – and the latest version of GNAT Pro implements some of the new features in Ada 2012. GNAT Pro is based on the widely used GCC technology and is backed by rapid and expert support service.

Tidorum — Bound-T WCET and stack analyser

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Thu, 04 Mar 2010 17:45:36 +0200
Subject: [ANN] Bound-T WCET and stack analyser: free downloads
Newsgroups: comp.arch.embedded

No-cost downloads of size-limited versions of the Bound-T static WCET and stack-usage analyser are now offered on the Bound-T website at

http://www.bound-t.com/.

Bound-T applies static program analysis to the machine code of executable binary files to compute upper bounds on the worst-case execution time (WCET) and the worst-case stack usage of selected subprograms. The results are useful for verifying that real-time programs are schedulable and do not fail due to stack overflow.

Bound-T is now available for the target processors Intel-8051, Atmel AVR, ARM7TDMI, ERC32 (SPARC V7), Renesas H8/300, and ADSP-21020.

Bound-T can be used on Intel/Linux, Intel/MS-Windows, and Mac OS/X host PCs. The website offers downloads for the first two hosts. Bound-T for the Mac will be provided on request.

Bound-T analyses the target program by reading and decoding binary instructions from the executable file; building the control-flow graphs and call graph; modelling the computation to find bounds on the number of iterations of (some) loops and bounds on the stack usage; and finally computing an upper bound on the execution time with Integer Linear Programming (the Implicit Path Enumeration technique). At present, WCET analysis in Bound-T works for processors with fixed, deterministic instruction execution times. Pipeline effects are modelled, but caches and other dynamic accelerators are not.

The no-cost downloadable versions are limited to about 1000 instructions per analysis. The total size of the target program is not limited. The no-cost versions are offered under a liberal licence (http://www.bound-t.com/download/tr-li-no-cost.pdf) that allows any kind of use, including commercial use.

On behalf of Tidorum Ltd, the developer of Bound-T, I hope that you will find the tool useful. While the no-cost versions of course do not include a promise of technical support, Tidorum welcomes all feedback on Bound-T and will respond within our resources.

From: Niklas Holsti <niklas.holsti@tidorum.fi>
Date: Thu, 04 Mar 2010 19:31:18 +0200
Subject: Bound-T WCET and stack usage analyser: free downloads
Newsgroups: comp.lang.ada

As some of you may already have seen on other newsgroups, free downloads are now offered of a tool called Bound-T that uses static program analysis of machine code to compute upper bounds on the execution time (WCET) and stack usage of embedded programs. I do not mean to repeat the announcement here (see comp.arch.embedded or comp.realtime), but just to add that Bound-T is written in Ada, and has also been used to analyse Ada code, as well as C, C++, and assembly-language code.

For more information see http://www.bound-t.com/.

Ada and GNU/Linux

gprbuild and GCC 4.4.2

From: Michael Rohan <michael@zanyblue.com>
Date: Thu, 31 Dec 2009 15:02:42 -0800 PST
Subject: Using gprbuild with GCC 4.4.2?
Newsgroups: comp.lang.ada

[…]

I've been using GNAT from GCC 4.4.2 for a while (tend to like to rebuild my compiler). I decided recently to explore the functionality of AWS from Ada Core however it uses "gprbuild" to build which is not included in the standard GCC release. Is it possible to simply grab it from the source release from Ada Core and "plug it into" a GCC 4.4.2 build? Has anyone tried this?

[…]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 31 Dec 2009 15:38:11 -0800 PST
Subject: Re: Using gprbuild with GCC 4.4.2?
Newsgroups: comp.lang.ada

[…]

http://www.debian.org

http://packages.debian.org/source/sid/libaws

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 01 Jan 2010 05:47:15 -0500
Subject: Re: Using gprbuild with GCC 4.4.2?
Newsgroups: comp.lang.ada

[…]

Also note that I'm working on packaging gprbuild for Debian. It's currently working; just needs some polishing to include the docs and examples.

You can check it out from the Ada France monotone server at

www.ada-france.org, branch org.debian.gprbuild.

PolyORB available in Debian

From: Reto Buerki <reet@codelabs.ch>
Date: Wed, 17 Feb 2010 18:54:29 +0100
Subject: Announce: PolyORB available in Debian
Newsgroups: comp.lang.ada

I'm proud to announce that the PolyORB middleware library has been accepted in the Debian testing distribution and will be part of the next Debian stable release "Squeeze" [1].

I would like to thank the co-maintainers Xavier Grave and Ludovic Brenta for their excellent work and support during the packaging process.

PolyORB provides an uniform solution to build distributed applications; relying either on industrial-strength middleware standards such as CORBA, the Distributed System Annex of Ada 95, distribution programming paradigms such as Web Services, Message Oriented Middleware (MOM), or to implement application-specific middleware.

The following application personalities are currently supported by the Debian package:

- CORBA
- DSA
- MOMA

Please feel free to test the package and report possible problems directly to the Debian Bug Tracking System (BTS) [2].

For those who would like to experiment with the PolyORB package without installing Debian testing or unstable can use the unofficial i386, amd64 backport packages for Debian stable from [3]. The backport does no support the DSA application personality because it requires a newer compiler than the one available in Debian stable.

[1] - http://packages.qa.debian.org/p/polyorb.html

[2] - http://www.debian.org/Bugs/

[3] - http://www.codelabs.ch

libxmlezout and liblog4ada in Debian

From: Xavier Grave <xavier.grave@ipno.in2p3.fr>
Date: Sat, 20 Feb 2010 11:57:50 +0100
Subject: Announce: XML EZ OUT and Log4Ada in Debian
Newsgroups: comp.lang.ada

[…]

I'm happy to announce that libxmlezout and liblog4ada reached Debian testing a few days ago.

libxmlezout 1.06 is the package for XML EZ OUT [1], it is a facility to produce xml to string, file and so on…

liblog4ada 1.0 is a package for Log4Ada [2] using libxmlezout. It is compatible with the log4j framework [3].

I'd like to thanks Ludovic Brenta to be my Debian mentor, sponsor and for tutoring me through this work. Thanks also to Marc Criley for his software [1] and help for the packaging of libxmlezout.

I hope it will be of some use.

Please feel free to test the package and report possible problems directly to the Debian Bug Tracking System (BTS) (Reto TM) [4].

[1] http://www.mckae.com/xmlEz.html

[2] http://green.ada-france.org:8081/branch/changes/org.log4Ada

[3] http://logging.apache.org/log4j/

[4] http://bugs.debian.org

Mailing list for Ada in Debian

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Fri, 26 Feb 2010 08:28:48 -0800 PST
Subject: Ada in Debian: new dedicated mailing list, debian-ada@lists.debian.org
Newsgroups: comp.lang.ada

I have requested and been granted the creation of a new mailing list[1] with web archives[2].

I started out in 2003 as (almost) the sole maintainer of Ada in Debian but, in the past couple of years, several people have joined in the effort. It has become necessary to coordinate work of all these people. Several of these people seconded my request (in fact, Xavier was the one who nudged me to make the request in the first place; thanks to him). The list will thus replace private emails as a means of coordination, making this much more public.

This list is intended for:

- discussion of the Debian Policy for Ada[3]
- release coordination among Ada packages in Debian
- packaging advice
- general help for Ada programmers using the Debian packages

This list is not moderated; posting is allowed by anyone.

Posting address: debian-ada@lists.debian.org

Intended audience:

- maintainers of Ada packages in Debian
- people who would like to join in the effort
- programmers using the Ada packages in Debian
- people considering Debian as an Ada development platform

If you wish to subscribe to the mailing list, please visit [1].

[1] http://lists.debian.org/debian-ada/

[2] http://lists.debian.org/debian-ada/recent

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

Ada and Microsoft

GNAT 2009, Windows and system libraries

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 10 Feb 2010 03:13:11 -0800 PST
Subject: GNAT 2009, Windows and system libraries
Newsgroups: comp.lang.ada

I have a problem writing a proper .gpr file for a project that is composed of Ada and C++ code and that uses Windows socket API.

I have no problem linking Ada with the C++ library, the problem is with system library known as Ws2_32.lib.

When compiling a similar project with a C++ compiler, it is enough to add Ws2_32.lib to the compiler invocation command and it just works. I cannot, however, find a proper way of doing it with Ada projects.

This is my try (Ws2_32.gpr):

```
project Ws2_32 is
  for Externally_Built use "true";
  for Source_Dirs use ();
  for Library_Dir use
    "C:\Program Files\
      Microsoft SDKs\Windows\
      v6.0A\Lib";
  for Library_Name use "Ws2_32";
  for Library_Kind use "dynamic";
end Ws2_32;
```

This .gpr file is "withed" by the project file of the final Ada program. This approach works for my own libraries, but is ineffective with the system library and gnatlink reports zillions of unresolved references.

[…]

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Wed, 10 Feb 2010 03:40:25 -0800 PST
Subject: Re: GNAT 2009, Windows and system libraries
Newsgroups: comp.lang.ada

[…]

Forgive me if it ever does not fulfill your requirements (you may want to rely on GPR project files only), here is how I do when I need this kind of linkage : I put some pragma Linker_Options ("-lkernel32"); pragma Linker_Options ("-lwsock32"); in the private part of specification files.

Notice the "-l" as a prefix and the lack of any "-Wl,"

If you opt for this solution, make sure you put this in a specification file which is required by all package which depends on it (a root package is a good place).

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 10 Feb 2010 07:21:20 -0800 PST
Subject: Re: GNAT 2009, Windows and system libraries
Newsgroups: comp.lang.ada

> I have a problem writing a proper .gpr file for a project that is composed of Ada and C++ code and that uses Windows socket API.

After struggling a while, I was able to compile everything by hand - that is, by manually running the GNAT toolchain. The problem with the use of .gpr files is that in the final invocation of gnatlink, libraries are listed in the order that comes from the depth-first traversal of all .gpr files that are connected by "with" relationships, which leads to unresolved references. The order that I need is depth-first (or more generally, topologically sorted).

How can I change the order of all dependent libraries that are used in the final invocation of gnatlink?

From: Maciej Sobczak <maciej@msobczak.com>
Date: Wed, 10 Feb 2010 07:35:31 -0800 PST
Subject: Re: GNAT 2009, Windows and system libraries
Newsgroups: comp.lang.ada

[…]

Final update: I was able to get the clean compile with proper order of "with" clauses in .gpr files. The only curiosity is that libraries are passed to linker in the order that is *reverse* to the order of relevant "with" statements.

I'm not sure if that was intended, but as long as I can control the results, it is just a minor detail.

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Wed, 10 Feb 2010 07:45:52 -0800 PST
Subject: Re: GNAT 2009, Windows and system libraries
Newsgroups: comp.lang.ada

[…]

The problem seems a little deeper than that.

```
with "a";
with "b";
project P is
   ...
end P;
```

translates to:

```
ld -o p p.o -lb -la
```

which is appropriate if libb.so needs to see the symbols in liba.so. However, in that case, b.gpr ought to have a with "a"; clause. If b.gpr lacks the clause, nothing allows the project manager to assume that libb.so needs liba.so. I suspect that the project manager was trying to be too clever for its own good; it was potentially hiding a problem (missing "with a" in b.gpr) while breaking the Law of Least Astonishment.

I'm not sure whether this is a genuine bug or not. Mmmh. Meditate on this, I will.

References to Publications

Embedded.com — “Executing software contracts”

From: Ada Information Clearinghouse news
Date: Fri, 19 Feb 2010
Subject: Executing software contracts
URL: http://www.adaic.com/whatsnew.html

Jack Ganssle at Embedded.com discusses the Sofcheck and AdaCore CodePeer tool in Executing software contracts; in particular he examines the tool's ability to create contracts for subprograms.

[Read the article by Jack Ganssle at http://www.embedded.com/222900387 —mp]

Ada Inside

Websites powered by Ada Web Server

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon, 21 Dec 2009 21:26:28 +0200
Subject: stronglytyped.org - Web site powered by Ada Web Server (AWS)
Newsgroups: comp.lang.ada

[…]

I know that there is already some sites running AWS[1], but they are still somewhat rare, so I decided to share my site, stronglytyped.org.

The site is running on a Linux-based Virtual Private Server (VPS), which actually hosts three web sites:

http://www.stronglytyped.org/

- the main site, currently no content, just links to Ada resources

http://ahven.stronglytyped.org/

- web pages for my unit testing framework

http://tkoskine.iki.fi/

- my personal homepage

Technical details:

This all is done using AWS' virtual hosting capabilities and there is NO other web server (like Apache or lighttpd) used. The executable runs under restricted account and I forward port 80 to the actual server port using iptables. (Because ports below 1024 are available only for root.)

The VPS is Linode 360 setup from linode.com and it is running Debian stable.

In addition, there is subdomain

http://hg.stronglytyped.org/

which hosts some of my public Ada code, but it is served externally by http://bitbucket.org/ (Python/Django-powered site).

Oh, and special thanks to Debian Ada maintainers, who provide a nice set of Ada packages (including AWS) working out of the box.

[1] Like http://ada-ru.org/ or parts of libre.adacore.com

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 23 Dec 2009 15:15:00 -0800 PST
Subject: Re: stronglytyped.org - Web site powered by Ada Web Server (AWS)
Newsgroups: comp.lang.ada

[…]

FWIW, blazedialer.com is also an AWS site, but I can't give out a login for exploring it.

From: Pascal Obry <pascal@obry.net>
Date: Thu, 24 Dec 2009 09:20:46 +0100
Subject: Re: stronglytyped.org - Web site powered by Ada Web Server (AWS)
Newsgroups: comp.lang.ada

[…]

The photo critic oriented Web site http://v2p.fr.eu.org (fully GPL) is also AWS based. It is for french speakers though.

We are using Google code to host it:

http://code.google.com/p/vision2pixels/

It is possible to download sources from Git repository or to explorer the sources there:

http://git.savannah.gnu.org/gitweb/?p=v2p.git

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Mon, 21 Dec 2009 16:07:15 -0800 PST
Subject: Re: stronglytyped.org - Web site powered by Ada Web Server (AWS)
Newsgroups: comp.lang.ada

> […] Oh, and special thanks to Debian Ada maintainers, who provide a nice set of Ada packages (including AWS) working out of the box.

Thanks for that. I do not ask or receive payment for all that work; my only reward is this kind of recognition, so your post is very motivating. I uploaded AWS 2.7 to unstable just a few days ago.

Also, you are now correct to use the plural since several people have started contributing to Ada in Debian. […]

From: Thomas Løcke <tl@ada-dk.org>
Date: Mon, 21 Dec 2009 21:21:35 +0100
Subject: Re: stronglytyped.org - Web site powered by Ada Web Server (AWS)
Newsgroups: comp.lang.ada

[…]

http://ada-dk.org is also hosted on a Linode 360 setup. They do indeed provide a great service at a very reasonable price.

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]: Embedded Software Engineer

Two Software Engineers are required to join a team of software engineers, developing new embedded platforms for global projects, designed to safety-critical and standards including ARINC 653 and POSIX.

[…] Key Responsibilities:

- Design of real-time embedded architectural and services software compliant with international standards for safety critical systems.
- Integration of bespoke software with commercial RTOS's and software services components.
- Hardware/software integration, debug and test
- Support to application software teams in the use of the processing platforms

Qualifications and Experience:

- First-hand experience in low-level platform and embedded software design, including the use of PowerPC or similar microprocessors.
- Hands-on experience in high-integrity software development programs including the use of software design and testing tools.
- Knowledge of Safety-Critical Engineering Standards, SIL, Def-Stan / Mil-Stan, ARINC 653 etc.
- Experience in a high-integrity industry such as Railway, Aerospace or Defence
- Knowledge of high level languages such as Ada and C++

Job offer [United Kingdom]: Software Test 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 a 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 Engineer

[…] You will take responsibility for the development of Safety Critical software within the software engineering department, throughout the full life cycle.

[…] Key Responsibilities :

- Prepare requirements specifications
- Prepare design documentation
- Write software code (C, C++, Ada, C#)
- Develop test specifications and perform tests
- Support of Integration and System Testing

[…] Qualifications and Experience :

- Degree in computer, software engineering, electronics or mathematical subjects
- Software engineering experience from safety-critical field (aerospace, nuclear, railway, defence, medical devices etc..)
- Proficient in coding in C, C++ and Ada
- OO Analysis
- Knowledge of networking protocols such as TCP/IP

Job offer [United Kingdom]: Software Engineer

We are looking to recruit graduate-level software engineers to support defence projects […]. The ideal candidate would be a university graduate in Software Engineering or a related course and have some commercial experience in object oriented and UML software development.

Ideally, you will have some commercial experience in Ada, C, C#, C++, Java or VB.NET

You will be involved in the design, development, test, debug and support of the software.

Job offer [United Kingdom]: Software Technical Lead

Technical Project Lead / Design Lead

[…]

Short Description:

The Project Technical Lead / Technical Design Authority is responsible for the project technical delivery from inception to delivering to the client, overseeing requirements, design, development, test and installation and delivery.

Essential Skills:

- C++, Java or Ada development background
- Linux
- UML
- Object Oriented Design
- Enterprise Architecture
- Strong Design background

Day to Day Responsibilities:

The position is to provide technical leadership / technical design across team members and customers at all stages of the project lifecycle - from initial bid and proposal, project initiation, requirements gathering, system design, implementation, test, acceptance and in-service support.

The applicant should be able to show good evidence of full lifecycle knowledge and that they are capable of technically leading multi-disciplined teams.

Project Technical Lead must be experienced in:

- System Design / System Architecture of high performance operational systems;
- Working with software engineers to ensure design is developed as required;
- Software development and system integration -proficient in Linux, C++, UML and structured functional and architectural modelling methods;
- Object Orientated Design(OOD) and modular design patterns;
- Implementing and maintaining project processes (CMM/CMMi desirable);
- Enterprise Architecture;
- Must have strong communication and interpersonal skills;
- Direct experience of customer requirements capture;
- Development of solutions to strict delivery timescales and budgets;
- Architectural design across all project disciplines, including hardware, software, test, delivery and support.

Desirable Skills:

- OpenGL

- GIS (Geospatial Information Systems)
- Iterative Methodologies
- System Architectures
- Experience of test architecture and process
- Experience of Defence and/or aircraft industry processes and standards

Job offer [Italy]: Software Developer

We are looking for a software developer with good programming knowledge and good proficiency in methodologies for software design/modeling and configuration management.

Technical requirements:

- Good knowledge of C/C++, Ada 83 or Ada 95
- Good knowledge of UML, OOA/OOD

The ideal candidate has at least 3 years of experience.

Educational background: Laurea [M.Sc. —mp] degree, preferably in Aeronautical Engineering or Computer Engineering.

[translated from Italian —mp]

Job offer [United States]: System Software Engineer

[…]

Required:

- A four year engineering degree and a minimum of eight years prior business, regional or air transport avionics product design/system engineering experience (or at least 12 years experience in lieu of a degree)
- Excellent written and verbal communication skills

Preferred:

- Interest in aircraft systems and operations is highly desired but not absolutely necessary
- Knowledge of RC ProLine II, 4, 21 or Air Transport Avionics systems as well as current and developing avionics industry standards is highly desirable
- Knowledge of C++, DOORS, Ada

Project:

This position will be involved with the avionics systems of Bombardier and/or Cessna aircrafts. Tasks will include writing system specifications for a wide variety of avionics subsystems; coordination within design and development engineering for the capture of technical requirements; system logical and physical interconnection/interface design; prototype test and evaluation; aircraft systems integration testing; and other testing as required to demonstrate compliance with certification requirements. The candidate may also be responsible for providing progress status, on assigned projects, including management of specific work packages for scope, and schedule in accordance with established systems engineering processes.

Job offer [United States]: Junior Software Engineer

[…] is seeking a Jr. Software Engineer who will join the software development team in support of the Multidisciplinary Engineering Services contract on-site at NASA Goddard Space Flight Center in Greenbelt, MD. The team is responsible for the developing the Goddard Dynamic Simulator (GDS) software and responsibilities include include software programming and unit and integration for GDS models using VHDL and FPGAs.

Required Skills:

- Must have a BS degree in Electrical Engineering, Computer Science or related technical discipline or possess equivalent experience in lieu of a degree
- Must have a minimum of 2+ years related engineering and/or software development experience involving electronics hardware and low level software development
- Must be familiar with VHDL programming for FPGA's
- Must be familiar with working in a Linux/UNIX OS programming environment
- Must have experience working within a scientific or research environment

Desired Skills:

- MS degree in Electrical Engineering is desirable
- Digital circuit design and Ada programming experience is desirable

Ada in Context

New ACM challenge is language-discriminatory

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Fri, 18 Dec 2009 12:26:46 +0100
Subject: New ACM challenge is language-discriminatory
Newsgroups: comp.lang.ada

ACM has launched a new programming challenge:

http://queue.acm.org/icpc/index.cfm?page=faq

However, the only languages allowed are C++, C# and Java. I could understand reasons for having everyone compete with the same language, but if more than one is allowed, it should be open to all (including our favorite language, of course ;-)).

Is anyone on this list influential enough at ACM to address this issue, or should we start a (mail) mass action?

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Fri, 18 Dec 2009 16:06:56 +0100
Subject: Re: New ACM challenge is language-discriminatory
Newsgroups: comp.lang.ada

Just a thing that I noted when last looking at ACM Queue (a while ago), namely that the language selection seems to be in accord with

(a) ACM sponsors,

(b) perception of dominating languages in magazines for job seekers, CS professors, and labor merchants.

And maybe they don't have Ada staff to look at solutions written in languages other than the currently supported?

("Currently" is encouraging, anyway.)

From: Adam Beneschan <adam@irvine.com>
Date: Fri, 18 Dec 2009 08:26:08 -0800 PST
Subject: Re: New ACM challenge is language-discriminatory
Newsgroups: comp.lang.ada

[…]

"All"? Georg implied that they may not have the staff to look at Ada solutions, but I'm sure that they at least have a few people there that can understand it; there have got to be tons of other obscure programming languages that nobody there would be able to understand. Plus, are any of the judges going to be enthusiastic about looking at a Forth program, or at an APL program that someone wrote in one line just to prove that they could?? :)

So some discrimination seems necessary---the line has to be drawn somewhere. Too bad they drew it on the wrong side of our language.

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 18 Dec 2009 21:49:41 -0500
Subject: Re: New ACM challenge is language-discriminatory
Newsgroups: comp.lang.ada

[…]

The competition doesn't mention anything about needing to read the code.

Once you sign in, you can get more info:

Player implementations are external to the game itself. A player is a separate executable that communicates with the game via standard input and standard output. The player is executed when the game starts up and continues running until the game is finished. At the start of each turn, the game engine sends the player a description of the game state. The player reads this description from standard input, chooses a move and sends it back by writing it to standard output.

…

Player code will be compiled and will run on a virtual machine running on a 3.0 Ghz

Xeon processor installed with version 5.2 CentOS. Java submissions will be compiled and run with version 1.6 of the Sun JDK, and C++ submissions will be compiled with version 4.1.2 of g++.

So there's no technical reason the code can't be in Ada. Or any other language supported by the CentOS distribution!

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Fri, 18 Dec 2009 21:52:39 -0500
Subject: Re: New ACM challenge is language-discriminatory
Newsgroups: comp.lang.ada

[…]

I don't see the actual definition of how to "win" the competition, but the implication is the only thing that counts is winning games against other code submissions - no judging of code.

We should at least ask for a rationale for the language choices.

From: Karl Nyberg <karl@grebyn.com>
Date: Fri, 18 Dec 2009 13:16:06 -0800 PST
Subject: Re: New ACM challenge is language-discriminatory
Newsgroups: comp.lang.ada

[…]

There is a feedback page associated with the ACM Queue initiative. As a longtime ACM member (since 1978), I have used this feature to request justification for the limitation specified in the competition and the mechanisms necessary to remove it.

Similar feedback is certainly welcome, I'm sure... :-)

On the body of generic units

From: xorquewasp@googlemail.com
Date: Sat, 9 Jan 2010 04:23:27 -0800 PST
Subject: GNAT requires body of generic unit to be present at build?
Newsgroups: comp.lang.ada

A discussion arose in #ada on irc.freenode.net about GNAT requiring the body of a generic unit to be present when compiling code that uses the generic.

A quote from the Ada 95 issues:

{AI95-00077-01} {AI95-00114-01} {extensions to Ada 83} Ada 83 allowed implementations to require that the body of a generic unit be available when the instantiation is compiled; that permission is dropped in Ada 95. This isn't really an extension (it doesn't allow Ada users to write anything that they couldn't in Ada 83), but there isn't a more appropriate category, and it does allow users more flexibility when developing programs. How come GNAT still requires this?

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Sun, 10 Jan 2010 06:23:43 -0800 PST
Subject: Re: GNAT requires body of generic unit to be present at build?
Newsgroups: comp.lang.ada

[…]

AI95-00077-01 Issue 6 starts by saying

"The NOTE in 10.1.4(10), which says that separate compilation of generic bodies is required, is correct. This implies that an implementation must be capable of detecting legality errors in a compilation unit that instantiates a generic unit, without seeing the generic body. It does not imply that the compiler must generate code without seeing the generic body."

So compiling with -gnatc (semantic check only) is OK in the absence of a generic body:

-- The generic body is present, code generation OK

```
nidhoggr:tests simon$ gnatmake -I../src
-c -u –f collection_test_support.ads
gcc -c -I../src
  collection_test_support.ads
```

-- Move the generic body

```
nidhoggr:tests simon$ mv ../src/bc-
containers-collections-unbounded.adb
../src/bc-containers-collections-
unbounded.adb
```

-- Code generation fails

```
nidhoggr:tests simon$ gnatmake -I../src
-c -u –f collection_test_support.ads
gcc -c -I../src
  collection_test_support.ads
collection_test_support.ads:24:06: body
of generic unit "Unbounded" not found
gnatmake: "collection_test_support.ads"
compilation error
```

-- Semantic checks OK

```
nidhoggr:tests simon$ gnatmake -I../src
-c -u –f collection_test_support.ads
-gnatc
gcc -c -I../src –gnatc
  collection_test_support.ads
```

On static libraries imported in a shared library

From: Andrei Krivoshei <andrei.krivoshei@gmail.com>
Date: Tue, 19 Jan 2010 10:02:26 -0800 PST
Subject: GNAT GPL 2009: Shared library project cannot import static library project?
Newsgroups: comp.lang.ada

[…]

I have tried to compile a shared library project using: "GPS 4.3.1 (20090114) hosted on pentium-mingw32msv & GNAT GPL 2009 (20090519)".

My shared library project imports a static library project and compiler (gnatmake) raise an error: "Shared library project cannot import static library project". Why???

Using the GNAT GPL 2008 with the same projects don't raise any errors.

[…]

From: Andrei Krivoshei <andrei.krivoshei@gmail.com>
Date: Wed, 20 Jan 2010 05:15:36 -0800 PST
Subject: Re: GNAT GPL 2009: Shared library project cannot import static library project?
Newsgroups: comp.lang.ada

[…]

More exactly, I have used the next sequence of commands:

```
# gnatmake -c -gnatc  -PMyProject.gpr -
d -XLegacy=3DAda2005
# gnatdll -d ../libMyDLL.dll -e
./src/libMyDLL.def -I../obj -n
../obj/mydll.ali
```

[…]

From: Vadim Godunko <vgodunko@gmail.com>
Date: Thu, 21 Jan 2010 11:01:57 -0800 PST
Subject: Re: GNAT GPL 2009: Shared library project cannot import static library project?
Newsgroups: comp.lang.ada

[…]

It is an error condition not detected by GNAT GPL 2008.

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Wed, 20 Jan 2010 12:12:45 -0800 PST
Subject: Re: GNAT GPL 2009: Shared library project cannot import static library project?
Newsgroups: comp.lang.ada

[…]

> My shared library project imports a static library project and compiler (gnatmake) raise an error: "Shared library project cannot import static library project". Why???

Perhaps because a shared library needs to be compiled to be position-independent (-fPIC on some platforms) but a static library doesn't?

[…]

Have you tried gprbuild? would (probably, I don't know Windows) know how to run gnatdll.

From: Per Sandberg <per.sandberg@bredband.net>
Date: Sat, 23 Jan 2010 23:07:59 +0100
Subject: Re: GNAT GPL 2009: Shared library project cannot import static library project?
Newsgroups: comp.lang.ada

Having a dynamic library importing a static library may lead to very "interesting" behavior of programs in a larger context, if there is any "static" data in the static library and therefore this is treated as an error by default with the GNAT tools.

This default behaviour could be suppressed with switches to gprbuild if you know what you are doing and have read the manuals in depth and understand all the implications.

[…]

Dynamic linking in Ada

From: Steven Shack <stevenshack@stevenshack.com>
Date: Thu, 3 Dec 2009 01:36:21 -0800
Subject: Loadable module in Ada.
Newsgroups: comp.lang.ada

I'd like to create a loadable module in an Ada program. I've searched around, but can't seem to find any examples of this. Say I've got some calculation or a driver and I'd like to have multiple different versions of it with the same interface. I'd like to be able to load and replace these modules at runtime.

In C I'd do this with pic code, dlsym and function pointers. I can't seem to find the equivalent in Ada. Can anyone help me out?

From: Xavier Grave <xavier.grave@ipno.in2p3.fr>
Date: Thu, 03 Dec 2009 11:00:44 +0100
Subject: Re: Loadable module in Ada.
Newsgroups: comp.lang.ada

[…]

I have done something that seems to fit with your need.

Here is a link where you will find a tgz file compiling/running under Linux using dlsym. It is based on an abstract class that your plugin should inherit of.

http://dl.free.fr/pWQwqMpVK [The link is no longer valid —mp]

Be very careful about the elaboration code when you do some "with" of other package. One on the "best" policy is to be the more restrictive that you can with the packages you "with" (pragma Pure and so on...)

[…]

From: Mark Lorenzen <mark.lorenzen@gmail.com>
Date: Thu, 3 Dec 2009 05:40:17 -0800 PST
Subject: Re: Loadable module in Ada.
Newsgroups: comp.lang.ada

[…]

I'm not sure what you mean by loadable module, but if you mean dynamic linking at run-time, you should take a look at this very interesting article:

http://www.adacore.com/wp-content/uploads/2005/04/dynamic_plugin_loading_with_ada.pdf

Note that you will of course still need support from the OS, so there is no pure Ada solution, just as there is no pure C solution (the dl* system calls just happen to have a C API).

From: Pascal Obry <pascal@obry.net>
Date: Thu, 03 Dec 2009 21:06:18 +0100
Subject: Re: Loadable module in Ada.
Newsgroups: comp.lang.ada

Steven, look at the Gwiad module:

http://code.google.com/p/gwiad/

```
$ git clone git://repo.or.cz/gwiad.git
```

It is a framework using whatever OS services underneath based on Ada interface.

[…]

Internationalization in Ada

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Sat, 12 Dec 2009 01:23:07 -0800 PST
Subject: Re: Internationalization for Ada
Newsgroups: comp.lang.ada

> […] Is there a preferred way for internationalization in Ada 2005 - like GNU gettext for GNU/Linux? If anyone has experience with this topic, I would be glad for a link which directs me into the right direction, also if there is something special for GtkAda. […]

See the package GtkAda.Intl in GtkAda.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 12 Dec 2009 12:51:35 +0100
Subject: Re: Internationalization for Ada
Newsgroups: comp.lang.ada

[…]

Yes, but there is also another way based on widget style properties, which I prefer.

Each variable text is made a style property of its container widget. When the widget is initialized, its class record is initialized as well. Here the necessary style properties are added:

```
if <the class record was initialized> then
  Class_Install_Style_Property
  (  Class_Record, -- Freshly initialized
                          class record
     Gnew_String
     (  Name    => "fancy text",
        Nick    => "hey",
        Default => "hey",
        Blurb   => "The text to appear
                    in the label"
  )  );
```

In the widget's "style_set" event handler the texts are actually set. E.g.

```
procedure Style_Set (
  Widget  : access
    Gtk_Fancy_Widget_Record'Class)
is
begin
  Set_Text (
        Widget.Label,
        Style_Get Widget, "fancy text");
  ...
```

The texts and other styles are then defined in the GTK resource file. That need to be done only if necessary, because there is a default for any property. Only in order to change the texts you provide and load another resource file. This can be done several times, if you properly handle the event "style_set".

The advantage of this method is that it does not require any tools or any additional files to work. It does not depend on the OS and its settings. (E.g. when I use German locale, I still don't want texts in German). Style properties can be more than only texts. For example you can have images there. The way properties are matched are very elaborated, you can define quite complex rules (by class name, by widget name etc), instead of simple equality in the case of locales.

A more detailed description how to deal with GTK style properties in GtkAda:

http://www.dmitry-kazakov.de/ada/gtkada_contributions.htm#4

On GTK resource files see:

http://library.gnome.org/devel/gtk/unstable/gtk-Resource-Files.html

From: Christophe Chaumet <devteam@chaumetsoftware.com>
Date: Sun, 13 Dec 2009 15:35:00 +0100
Subject: Re: Internationalization for Ada
Newsgroups: comp.lang.ada

[…]

For my own usage I have a package "intl" which contains all the string literals and there is an instance of this package for each language (intl-fr.ads, intl-en.ads,...) and I have defined a variable in GPS called 'language'. In the project file there is something like:

```
package Naming is
  case language is
   when "french" => for
    Specification ("intl") use "intl_fr.ads";
   when "english" => for
    Specification ("intl") use "intl-en.ads";
  end case;
end Naming;
```

When I want to generate an executable in a language I just select the proper value and the build do all the job. This works only with GPS, but its is independent of any other package like Gtk.

[…]

From: Jacob Sparre Andersen <sparre@nbi.dk>

Date: Mon, 14 Dec 2009 16:18:49 +0100
Subject: Re: Internationalization for Ada
Newsgroups: comp.lang.ada

> For my own usage I have a package "intl" which contains all the string literals and there is an instance of this package for each language […]

But this means that translation has to be done _before_ compilation.

One of the benefits of using the GNU Gettext system is that translators can translate and distribute translations independent of the compiled program.

From: Georg Bauhaus <rm-host.bauhaus@maps.futureapps.de>
Date: Sat, 12 Dec 2009 13:17:57 +0100
Subject: Re: Internationalization for Ada
Newsgroups: comp.lang.ada

[…]

GtkAda's binding to libintl/GNU gettext seems the obvious choice in this case. There is, however, another mode of handling text that benefits from the Ada type system. The two can in fact be combined to yield an enhancement.

One way to distinguish a message from a menu text from a button text from a log entry template text or ... is to ignore their difference and make them all string literals, then inspect their context, and possibly follow some preprocessing conventions so that tools outside the language can hopefully set up a database – non-standard, though widespread in its niche – of strings to be loaded.

You wouldn't be doing this to whole numbers in Ada, would you? The obvious choice is to define different types of numbers for your different numbers. Or to think even harder about what you can do to go from one "measurements unit locale" to another, in case the numbers represent quantities of a certain type.

The pieces of text just enumerated belong to different sets of entities, part of different behavior of the program, serving different functions (labels, help text, log entries). Aren't they be worth a type? The text types will then serve two purposes:

1 - distinguish the different kinds of information that now happen to be represented(!) as objects of type string.

2 - use standard Ada programming to produce translation files guided by the text types.

Don't know, though, whether there isn't too big a blind spot here.

The point is, you use plain Ada tools to

(1) search your program for just the instances of text you want,

(2) make the findings available in a format of some translation tool, e.g. Gtk's or Qt's translation tools (The latter does _not_ require your program to use Qt.)

(3) use any mechanism of choice to load the translations

Part 3, with whichever mechanism, requires some form of unique identifier. This text identifier could be the variable's fully qualified name---unavailable at run time in Ada, but available to ASIS base tools.

Or you choose yourself a unique enumeration literal, like the codes of SQL diagnostic messages.

Or some numbering scheme such as the one that HTTP uses for status codes and corresponding message text.

```
package Example is
   type Fillin_Slot_Count is
      range 0 .. Max_Fillin_Slots;
   -- "A % has been found on line %"
   -- has 2 slots
   type Label_Text is access String;
   type Label_ID is
     (Name_Field, Age_Field,
      Amount_Field, ...);
   Known_Message : array (Label_ID)
      of Label_Text;
   type Field_Label
        (Lang : Supported_Language;
         For_ID : Label_ID)
    is
     new Limited_Controlled with
     record
        Text : Known_Messages
            (For_ID);
        Insertions : Fillin_Slot_Count;
     end record;
   overriding procedure Initialize
      (Object : in out Field_Label);
   -- check consistency,
   -- e.g. is no of slot markers =
   -- Object.Insertions?
   procedure Finalize
      (Object : in out Field_Label);
   -- free storage used etc.
   Confirmation_Button := ...
     Field_Label'(Lang => en_UK,
        For_ID => Confirmation_Button,
        Text => Known_Message (...)));
end Example;
```

The important variable in Example is Known_Messages. With it, it is now easy to write a simple Ada program that performs part (2) above. That is, the program writes out a representation of all messages to be translated, in the format needed by your preferred translation tool. For example, GNU Gettext or Qt Linguist.

Likewise, the definitions in package Example above provide all information needed to load a specific message in a specific language at run time.

Interfacing Ada code compiled with different compilers

From: David Henry <tfc.duke@gmail.com>
Date: Wed, 2 Dec 2009 06:54:05 -0800 PST
Subject: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

[…]

Is it possible to interface Ada code compiled with an older compiler (GNAT 3.14) and for which I just have specs (ads), objects and ali files (so I can't recompile it), with Ada code compiled with a newer compiler (GNAT 2009), and for which I have all the sources?

From: Jean-Pierre Rosen <rosen@adalog.fr>
Date: Wed, 02 Dec 2009 17:21:35 +0100
Subject: Re: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

[…]

Simple answer: no. The ali file contains the version of the compiler that was used, and the binder will refuse a unit from a different version.

Now, if you are in a desperate situation, you can try and edit the ali file to fool the binder (it is a plain text file, and quite easy to understand). Who knows? If you are very lucky, it might even work…

From: Pascal Obry <pascal@obry.net>
Date: Wed, 02 Dec 2009 18:40:52 +0100
Subject: Re: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

> Simple answer: no. The ali file contains the version of the compiler that was used, and the binder will refuse a unit from a different version.

Hum... For "simple" code not using the Ada runtime, won't it work to consider this old Ada code as it was some external C code:

0. Remove all .ali files coming from 3.14 compiler. Move away specs too.

1. Create interface specs for say "old.o" object code:

```
procedure Foo (P : in Integer);
pragma Import (Ada, Foo);
```

2. When linking add the old object file

```
$ gnatmake xyz -largs old.o
```

If the old code uses the Ada runtime (tasks, controlled objects, …) this just won't work of course.

From: Simon J. Wright <simon.j.wright@mac.com>
Date: Wed, 2 Dec 2009 14:15:47 -0800 PST
Subject: Re: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

[…]

Your problem (well, one of them!) is going to be with getting elaboration right.

Then there's differences between the runtimes expected by generated code.

For the elaboration problem, do you have enough 3.14 artifacts to allow you to run the 3.14 binder? You could maybe make a library out of the 3.14 code (+ 3.14 runtime)?

Not sure how much reliance I'd want to place on the result.

From: David Henry <tfc.duke@gmail.com>
Date: Wed, 2 Dec 2009 23:48:39 -0800 PST
Subject: Re: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

[…]

I'm afraid my old code is using the GNAT runtime :(

I imagined once something like interfacing old code with C and then the C with new code. But linking may be hard, especially with the GNAT runtime dependency.

From: Pascal Obry <pascal@obry.net>
Date: Thu, 03 Dec 2009 21:03:00 +0100
Subject: Re: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

[…]

Yes, if there is some GNAT runtime dependencies that's almost impossible. Especially in your case where you have code using a very old runtime where some routines may have changed or even removed in new runtime.

From: Florian Weimer <fw@deneb.enyo.de>
Date: Sat, 05 Dec 2009 11:22:22 +0100
Subject: Re: Interfacing Ada with Ada
Newsgroups: comp.lang.ada

[…]

If your platform has a decent linker, you can link the old code and only make very few symbols available. You need to link the GNAT run-time library statically, but that's probably the least of your problems.

On deprecated subprograms in Ada

From: Michael Rohan <michael@zanyblue.com>
Date: Wed, 20 Jan 2010 19:47:28 -0800 PST
Subject: No "pragma Deprecated (NAME)"?
Newsgroups: comp.lang.ada

[…]

I was re-organizing some code and wanted to change some procedure/function names, e.g.,

```
function To_String (T : My_Type)
  return String;
```

becomes

```
function Name (T : My_Type)
  return String;
```

Without consulting the docs, I tried

```
function Name (T : My_Type)
  return String;
function To_String (T : My_Type)
  return String
  renames Name;
pragma Deprecated (To_String);
```

I guess I'm too used to annotating methods in Java and was surprised I couldn't do this in Ada.

The only way to do this is to simply rename and fix the compilation errors which is may be in the Ada style: just get it right.

[…]

From: Jeffrey R. Carter <jrcarter@acm.org>
Date: Wed, 20 Jan 2010 20:57:55 -0700
Subject: Re: No "pragma Deprecated (NAME)"?
Newsgroups: comp.lang.ada

[…]

"Deprecated" is not a language-defined pragma, but unrecognized pragmas should be ignored (ARM 2.8). I see nothing wrong with your function renaming. What error message did you get?

From: Michael Rohan <michael@zanyblue.com>
Date: Wed, 20 Jan 2010 20:00:36 -0800 PST
Subject: Re: No "pragma Deprecated (NAME)"?
Newsgroups: comp.lang.ada

[…]

Yes, it generated a warning as un-recognized but I was hoping for the functionality, i.e., the implementation of this pragma where usage of the old name generates deprecated warning when used.

[…]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Thu, 21 Jan 2010 01:30:33 -0800 PST
Subject: Re: No "pragma Deprecated (NAME)"?
Newsgroups: comp.lang.ada

[…]

GNAT has an implementation-defined "pragma Obsolescent", see

http://gcc.gnu.org/onlinedocs/gnat_rm/Pragma-Obsolescent.html

Support for Transport Layer Security in Ada

From: Riccardo Bernardini <framefritti@gmail.com>
Date: Fri, 18 Dec 2009 04:44:06 -0800 PST
Subject: Transport Layer Security for Ada?
Newsgroups: comp.lang.ada

[…]

a really fast question: does it exist a freely available package for using TLS (Transport Layer Security) for Ada (even GNAT-dependent is OK)? (e.g., a binding to the GNU TLS library)

[…]

From: Ludovic Brenta <ludovic@ludovic-brenta.org>
Date: Fri, 18 Dec 2009 07:26:51 -0800 PST
Subject: Re: Transport Layer Security for Ada?
Newsgroups: comp.lang.ada

[…]

There used to be one as part of AWS but they removed it. You can still get the last version of the sources from AdaCore's Subversion repository:

http://libre2.adacore.com/viewvc/trunk/AWS/ssl/ssl-gnutls.ads?revision=8743&view=markup&sortby=log&pathrev=8743

(yes, it is a single file AFAICT).

Implementation of Ada.Execution_Time in GNAT

From: Ingo Sander <sander.ingo@gmail.com>
Date: Fri, 4 Dec 2009 03:09:35 -0800 PST
Subject: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

I cannot get the package Ada.Execution_Time to work with GNAT, although the GNAT documentation says that the real-time annex is fully supported… I use the GNAT version 4.4 on a Ubuntu 9.10 distribution.

The typical error message I get is

```
gcc -c executiontime.adb
Execution_Time is not supported in this
configuration compilation abandoned
```

How can I configure GNAT to support the Ada.Execution_Time package?

[…]

Below follows an example program that generates the error message.

[…]

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 4 Dec 2009 12:26:26 +0100
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

I cannot tell anything about Ubuntu, but the program you provided contains language errors.

It also has the problem that "delay" is non-busy in Ada, i.e. the program will count 0 CPU time for a very long time, at least under Windows, where the system services, which I presume, Ada.Execution_Time relies on, are broken.

Anyway, here is the code which works to me:

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Real_Time;
use Ada.Real_Time;
with Ada.Execution_Time;
use Ada.Execution_Time;
procedure ExecutionTime is
  task T;
  task body T is
    Start : CPU_Time := Clock;
  begin
    loop
      Put_Line (Duration'Image
          (To_Duration (Clock - Start)));
      for I in 1..40 loop
        Put ("."); -- This does something!
      end loop;
    end loop;
  end T;
begin
  null;
end ExecutionTime;
```

Under Windows this shows rather poor performance, which again is not surprising, because as I said there is no way to implement Ada.Execution_Time under Windows.

Maybe Linux counts CPU time better, I never investigated this issue.

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Fri, 04 Dec 2009 13:10:26 +0100
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

The reasons are explained in the GNAT source files.

[…]

```
--  This unit is not implemented in
-- typical GNAT implementations that lie
-- on top of operating systems, because
-- it  is infeasible to implement in such
--  environments.
--  If a target environment provides
--  appropriate support for this package
--  then the Unimplemented_Unit
-- pragma should be removed from this
-- spec and an appropriate body
-- provided.

with Ada.Task_Identification;
with Ada.Real_Time;

package Ada.Execution_Time is
  pragma Preelaborate;

  pragma Unimplemented_Unit;
```

From: Ingo Sander <sander.ingo@gmail.com>
Date: Mon, 7 Dec 2009 00:08:55 -0800 PST
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

Still I wonder why it is written in the GNAT reference specification that the real time annex is fully implemented [1].

"Real-Time Systems (Annex D) The Real-Time Systems Annex is fully implemented."

According to the ARM 'Execution Time' is part of the real-time annex [2], so it should be implemented.

So, does "fully implemented" mean that it is only in principle fully implemented, but that the underlying OS/hardware (in my case 64-bit Ubuntu-Linux (9.10) on an Intel QuadCore) has to support this features as well?

[…]

[1] http://gcc.gnu.org/onlinedocs/gnat_rm/Specialized-Needs-Annexes.html#Specialized-Needs-Annexes

[2] http://www.adaic.org/standards/05rm/html/RM-D-14.html

From: John B. Matthews <jmatthews@wright.edu>
Date: Mon, 07 Dec 2009 12:13:20 -0500
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

I'm guessing "fully implemented" on supported platforms and "not required in all implementations [1]." GNAT actually meets the implementation and documentation requirements [2]. My OS simply doesn't have the required facilities; it's designed for a GUI user, not real-time. If I were cross-developing, I'd perhaps create a fake body.

On Linux, it might be possible to get something relatively informative out of /proc/<PID>/task/<TID>/stat.

[1]<http://gcc.gnu.org/onlinedocs/gnat_rm/Specialized-Needs-Annexes.html#Specialized-Needs-Annexes>

[2]<http://www.adaic.org/standards/05rm/html/RM-D-14.html>

From: John B. Matthews <jmatthews@wright.edu>
Date: Fri, 04 Dec 2009 13:28:24 -0500
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

> How can I configure gnat to support the Ada.Execution_Time package?

I defer to Dmitry A. Kazakov about Windows, but this variation produces similar results on MacOS 10.5 & Ubuntu 9.10 using GNAT 4.3.4:

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Real_Time;
use Ada.Real_Time;
procedure ExecutionTime is
  task T;
  task body T is
    Start : Time := Clock;
    Interval : Time_Span :=
          Milliseconds(100);
  begin
    loop
     Put_Line(Duration'Image
          (To_Duration(Clock - Start)));
      delay To_Duration(Interval);
    end loop;
  end T;
begin
  null;
end ExecutionTime;

<console>
$ ./executiontime
 0.000008000
 0.100168000
 0.200289000
 0.300409000
 0.400527000
 0.500575000
…
</console>
```

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 4 Dec 2009 20:01:57 +0100
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

Your code counts the wall clock time. On the contrary Ada.Execution_Time should do the task time, i.e. the time the task actually owned the processor or, maybe, the time the system did something on the task's behalf.

This package heavily depends on the OS services at least when the tasks are mapped onto the OS scheduling items (like threads).

As far as I know it is impossible to implement it reasonably under Windows, because the corresponding service (used by the Task Manager too) counts time quanta instead of the time. This causes a massive systematic error if tasks are switched before they consume their quanta. I.e. *always* when you do I/O or communicate to other tasks. The bottom line, under Windows Ada.Execution_Time can be used only for tasks that do lengthy computations interrupted only by the scheduler, so that all counted quanta were consumed and no time were spent in uncounted quanta.

I don't know, if or how, this works under Linux or Max OS.

From: John B. Matthews
<jmatthews@wright.edu>
Date: Fri, 04 Dec 2009 16:50:01 -0500
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

Ah, thank you for clarifying this. Indeed, one sees the secular growth in the output as overhead accumulates. I meant to suggest that other parts of Annex D may be supported on a particular platform, even if Ada.Execution_Time is not.

[…]

I should have mentioned that both systems specify "pragma Unimplemented_Unit" in Ada.Execution_Time. On Mac OS X 10.5, Ada.Real_Time.Time_Span_Unit is 0.000000001, but I'm unaware of a Mac clock having better than microsecond resolution, as suggested in the output above. I'm running Linux in VirtualBox, so I suspect any results reflect the host OS more than anything else.

From: Randy Brukardt
<randy@rrsoftware.com>
Date: Fri, 4 Dec 2009 20:59:25 -0600
Subject: Re: gnat: Execution_Time is not supported in this configuration
Newsgroups: comp.lang.ada

[…]

Obviously this depends on the purpose. For many profiling tasks, the Windows implementation is just fine. The quanta seem to be short enough that most tasks run long enough to be counted. (And I/O has probably already be reduced to the minimum before even applying a profiler, if not, you're probably profiling the I/O first, not the CPU.) But I would have to agree that it isn't all that real-time.

In any case, thanks for the clear explanation of the limitations of the Windows services. I'm sure that I'll run into them sooner or later and I'll hopefully remember your explanation.

Ensuring resource cleanup

From: Florian Weimer
<fw@deneb.enyo.de>
Date: Mon, 08 Feb 2010 15:16:12 +0100
Subject: Ensuring resource cleanup
Newsgroups: comp.lang.ada

It looks as if I might need to do some Ada maintenance programming soon. I can use the GNAT from GCC 4.3.

Are there any new ways to ensure resource cleanup?

I tried to use Ada.Finalization. Limited_Controlled in the past, but there were several issues with it: there was some run-time overhead (because of the tag and because the finalizer is abort-deferred, which seemed to defeat inlining and scalar replacement of aggregates), it was impossible to instantiate generics containing such types below the library level, and having multiple different types inheriting from Limited_Controled in the same package often resulted in multi-dispatch errors (and using 'Class required exposing the tagged nature of the type in the interface, which has other drawbacks).

Explicit cleanup using cleanup subprograms is okay, too, provided that there is some tool to ensure that they are used properly. This is despite the rather cumbersome syntax:

```
declare
  X : Object;
begin
  Init (X);
  begin
    Make_Use_Of (X);
    exception
      when others =>
        Cleanup (X);
        raise;
  end;
  Cleanup (X);
end;
```

But I'd really have a tool that ensures that Init and Cleanup calls are properly paired in this way.

Are there other approaches I don't know about yet?

From: Jean-Pierre Rosen
<rosen@adalog.fr>
Date: Mon, 08 Feb 2010 16:29:38 +0100
Subject: Re: Ensuring resource cleanup
Newsgroups: comp.lang.ada

[…]

Such a tool exists, of course ;-). AdaControl rule Unsafe_Paired_Calls.

From: Robert A Duff
<bobduff@shell01.TheWorld.com>
Date: Mon, 08 Feb 2010 10:31:20 -0500
Subject: Re: Ensuring resource cleanup
Newsgroups: comp.lang.ada

> […] I tried to use Ada.Finalization.Limited_Controlled in the past, but there were several issues with it: there was some run-time overhead AdaCore is working on a more efficient implementation of finalization.

> (because of the tag and because the finalizer is abort-deferred, …

I think if you use the appropriate pragma Restrictions, so the compiler knows there are no aborts, it will avoid the cost of deferring and undeferring aborts (which is quite high on some systems).

> …which seemed to defeat inlining

Inlining of what? The Initialize and Finalize calls? It seems feasible to inline them in most cases, but I'm not sure if GNAT is capable of that. Try it.

> …and scalar replacement of aggregates), it was impossible to instantiate generics containing such types below the library level,…

Ada 2005 allows such nesting (with or without generics).

> …and having multiple different types inheriting from Limited_Controlled in the same package often resulted in multi-dispatch errors (and using 'Class required exposing the tagged nature of the type in the interface, with has other drawbacks).

Yes, but in my experience these are minor issues.

Another way to avoid multi-dispatch is to put the procedure in a nested package.

Both workarounds are annoying, I admit.

> […] But I'd really have a tool that ensures that Init and Cleanup calls are properly paired in this way.

You could wrap with this procedure:

```
procedure With_Cleanup (
   Action : not null access
                  procedure (...));
```

so you only have to write the above pattern once (per type that needs cleanup), and you can call it with any Action procedure you like.

Note that this is slightly different from using Limited_Controlled, because it does not clean up in case of abort. If you don't use abort, then this is not an issue.

[…]

From: Florian Weimer
<fw@deneb.enyo.de>
Date: Mon, 08 Feb 2010 17:01:55 +0100
Subject: Re: Ensuring resource cleanup
Newsgroups: comp.lang.ada

[…]

What is the appropriate pragma? This doesn't seem to have an effect:

[…]

Most of the benefit of inlining them, because there are multiple subprogram calls involved, including indirect ones. I doubt GCC treats them as intrinsics, so they interfere with register allocation etc. GCC doesn't seem to be able to devirtualize the implicit call to Finalize, either.

I don't understand why GNAT needs to maintain a separate finalization list, either. C++ use regular exception handling for this task.

> […]

> procedure With_Cleanup (Action : not null access procedure (...));

> […]

Yes, I need to try that. Back when the original code was written, anonymous access-to-subprogram types were still rather buggy.

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Mon, 08 Feb 2010 11:18:42 -0500
Subject: Re: Ensuring resource cleanup
Newsgroups: comp.lang.ada

[…]

That one says "I promise not to say 'with Ada.Asynchronous_Task_Control'", which is not related to aborts. It is considered obsolescent, because there's now a more general No_Dependence restriction.

To get rid of aborts, I think you need both No_Abort_Statements and Max_Asynchronous_Select_Nesting => 0. I believe this will cause the overhead of abort deferral to go away, but to be sure, you should try it.

> […] I don't understand why GNAT needs to maintain a separate finalization list, either. C++ use regular exception handling for this task.

The new implementation of finalization I mentioned avoids those lists. Except that lists are needed in the case of heap-allocated objects, because they need to be finalized when the scope of the access type is left (unless finalized early by Unchecked_Deallocation). C++ does not require that, so is easier to implement, at the cost of possibly missing some finalizations.

But anyway, efficient finalization is most important for stack-allocated objects.

> […] Back when the original code was written, anonymous access-to-subprogram types were still rather buggy.

And inefficient, because they used trampolines. GNAT got rid of trampolines except in some corner cases. Note that trampolines will cause your program to crash if DEP is enabled on Windows (or the equivalent feature on Linux). There's a restriction for that, too -- search the GNAT docs for "trampoline".

From: Robert A Duff <bobduff@shell01.TheWorld.com>
Date: Mon, 08 Feb 2010 15:47:51 -0500
Subject: Re: Ensuring resource cleanup
Newsgroups: comp.lang.ada

> Max_Asynchronous_Select_Nesting => 0 does indeed the trick.

Don't you need No_Abort_Statements as well?

[…]

You're welcome. What goes on at run time for a select-then-abort is very similar to what goes on for aborting a task. Deferring aborts applies to both.

On the proposal for pre- and post-conditions in Ada

From: Georg Bauhaus <rm.dash-bauhaus@futureapps.de>
Date: Thu, 04 Feb 2010 12:26:11 +0100
Subject: Specifying the order of ops on an ADT with aspects
Newsgroups: comp.lang.ada

Defining a private type, I'd like to know whether it is possible to specify, with the help of the new aspects, possible orders of calling the operations.

Showing my ignorance, to which extent might it be possible to analyse the order of calls at compile time?

(AI05-0145-{1,2}, AI05-0183)

```
generic
  type P is private;
package Order is
  pragma Pure (Order);
  -- ascertain that Pre functions do
  -- not have side effects
  type States is (S0, S1, S2, S3);
  type T is tagged private;
  function State_Of (Object : in T)
     return States;
  procedure Op1 (Object : in out T;
                        Param : in P)
     with Pre => State_Of (Object) = S0;
  procedure Op2 (Object : in out T;
                        Param : in P)
     with Pre => State_Of (Object) = S1
                 and Knoptuous (Object);
  procedure Op3 (Object : in out T;
                        Param : in P)
      with Pre => State_Of (Object)
                in S1 .. S2;
  function Knoptuous (Object : in T)
     return Boolean;
private
  type T is tagged
    record
      State_Of : States := S0;
    end record;
end Order;
```

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Thu, 4 Feb 2010 10:07:17 -0800 PST
Subject: Re: Specifying the order of ops on an ADT with aspects
Newsgroups: comp.lang.ada

[…]

Your package example looks good to me, except the assumption you've made about "analyse the order of calls at compile time"

The new pre-post-condition […], is not intended to be checked at compile time nor even at run time. I gonna miss this last one (the first one would be a heavy pain to implement), but I suppose some compiler vendors will probably have an option for that and will go the Eiffel way with this contract clauses: enable or disable clauses-check, just like you can enable or disable generation of debugging information with any compiler.

AI05-0145-2 says

http://www.ada-auth.org/cgi-bin/cvsweb.cgi/ai05s/ai05-0145-2.txt?rev=1.4

> This is based on the previous alternative AI05-0145-1. The Pre/Post aspects are specified using the aspect_specification syntax defined in AI05-0183-1. There is no message associated with the failure of a precondition or postcondition check: it was deemed that these annotations are intended for verification, and that for debugging purposes the Assert pragma is sufficient.

The last sentence is the most important for your topic.

All providing I've really understood your question

"intended for verification, and that for debugging purposes the Assert pragma is sufficient"

I was exactly feeling the opposite, that Assert pragmas are not sufficient and are hard to maintain and copy in implementations accordingly to contracts in specifications.

From: Yannick Duchêne <yannick_duchene@yahoo.fr>
Date: Fri, 5 Feb 2010 08:55:31 -0800 PST
Subject: Re: Specifying the order of ops on an ADT with aspects
Newsgroups: comp.lang.ada

> I agree that no message for a failing precondition or post-condition check is bad. A newer Ada standard does not necessitate a better language.

Don't be sad, pretty sure most of vendors will provide it ;) After all, the Ada standard does not specify anything either about debugging information and the like, and indeed, that's not its area. This may be the reason why of the actual ARG vote.

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Fri, 5 Feb 2010 19:34:17 +0100
Subject: Re: Specifying the order of ops on an ADT with aspects
Newsgroups: comp.lang.ada

[…]

Whatever, but I see no need in yet another syntax for run-time assertions. Statically checked contracts in the form of pre- and post-conditions would be a great language improvement […]

And my painful experience tells me that no check is optional. There is either one or none. I bet that any suppressed check will eventually fail in the production code, unless you do things like code coverage etc, but these would eliminate the very need to check something that you already proved to hold.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 5 Feb 2010 16:15:12 -0600

Subject: Re: Specifying the order of ops on an ADT with aspects
Newsgroups: comp.lang.ada

[…]

I think you guys miss the point of that statement. A Precondition that fails raises Assert_Error (unless suppressed, of course). That gets handled in the normal way, whatever your implementation does for unhandled exceptions. Nothing new here.

But the original proposal included an optional message string, similar to the one the Assert pragma has. We decided to drop that because programs that fail Preconditions are just wrong, and there is no need to go into detail *why* they're wrong.

I'd expect Janus/Ada to report something like:

** Unhandled Assert_Error - precondition check failed

In any case, Ada has nothing to say about how unhandled exceptions are reported.

We're working hard on a proposal to give the compiler enough information to be able to do static analysis of preconditions/postconditions, etc. We won't mandate that this time, but we surely want the possibility to exist -- otherwise there is little value to specifying these over plain old Assert pragmas. (Perhaps next time we'll be able to mandate some static checking.)

Data structures for syntax trees

From: Tero Koskinen <tero.koskinen@iki.fi>
Date: Mon, 1 Feb 2010 21:34:41 +0200
Subject: Data structures for syntax trees
Newsgroups: comp.lang.ada

[…]

I am looking for advice on what kind of data structures to use for abstract syntax trees. At the moment, I want to present JSON language as a tree: http://www.json.org/

The requirement is that the data structure works with GNAT GPL and Janus/Ada. So far, the Janus/Ada part has been giving me trouble. (With GNAT everything works.)

I have tried to avoid access types and create a tree structure using tagged types. However, Janus/Ada has some bugs with either class-wide types or parameterized types containing controlled types and passing/copying them around causes crashes when the produced executable is run. RR Software is aware of the issue, but hasn't been able to give an estimate when the issue will be fixed. So, I am looking for alternatives.

My current (crashing[1]) attempt is available at

http://bitbucket.org/tkoskine/jdaughter/src/950f001bf2aa/src/json-data.ads

http://bitbucket.org/tkoskine/jdaughter/src/950f001bf2aa/src/json-parser.adb

(Basically I have bunch of tagged types derived from JSON_Root_Type and a few container types, which take JSON_Root_Type'Class objects.)

RR Software told me that using access types should work, but I would avoid that as long as possible.

Is there still some nice way to avoid access types (in the public API at least) while also not using class-wide or controlled types?

And what kind of data structures others have used to present syntax trees?

[1] crash usually happens in line 134 or 146 of json-data.adb

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Mon, 1 Feb 2010 21:03:44 +0100
Subject: Re: Data structures for syntax trees
Newsgroups: comp.lang.ada

[…]

I am using access types to non-controlled class-wide objects allocated in an arena pool. A syntax tree is usually removed as a whole.

```
Tree_Pool : Stack_Storage.Pool (
      2048, 128);
type Node is abstract tagged limited
      null record;
function Image (Item : Node)
      return String is abstract;
type Node_Ptr is access Node'Class;
for Node_Ptr'Storage_Pool
     use Tree_Pool;

type Term is abstract new Node
  with record
     Location : Parsers.Multiline_Source.
                                Location;
end record;

type Expression (Count : Positive) is
  new Node with record
    Operation : Operations;
    Location  : Parsers.Multiline_Source.
                               Location;
  Operands  : Argument_List (1..Count);
end record;
```

And so on. The example can be found here:

http://www.dmitry-kazakov.de/ada/components.htm#12.9

I cannot tell if that works with Janus, I don't have the compiler.

Alternatively I would use a directed graph. That again would be access types to the arena pool of the graph nodes, parent-child relations maintained implicitly by the pool.

Use of 'use at' inside a record

From: PForan <pforan@gmail.com>
Date: Wed, 9 Sep 2009 13:07:51 -0700 PDT
Subject: using the 'use at' feature inside a record?
Newsgroups: comp.lang.ada

[…]

I have a question about the "use at" feature and how it can be used inside a record to simulate C's unions. I often have an array of bytes, as well as a string which is "use at"'ed on top of the byte buffer, very useful as I can play with the data in any way I choose. i.e.

```
byte_buf: array(1..100) of unsigned_8;
str_buf: string(1..100);
for str_buf use at byte_buf'address;
```

Is it possible to have something like this within a record?

[…]

From: Stephen Leake <stephen_leake@stephe-leake.org>
Date: Thu, 10 Sep 2009 19:42:47 -0400
Subject: Re: using the 'use at' feature inside a record?
Newsgroups: comp.lang.ada

'use at' is obsolete (LRM J.7); you should now use:

```
for str_buf'address use
   byte_buf'address;
```

If you are implementing an Ada variant record type that must match a C union type that has no field for the discriminant, use pragma Unchecked_Union (LRM B.3.3).

From: Per Sandberg <per.sandberg@bredband.net>
Date: Thu, 10 Sep 2009 05:17:33 +0200
Subject: Re: using the 'use at' feature inside a record?
Newsgroups: comp.lang.ada

Well there are always unchecked unions: "ARM B.3.3 Pragma Unchecked_Union"

```
type foo (dummy : Integer := 0) is
record
  case dummy is
   when 1 => as_integer  : integer;
   when 2 => as_float : Short_Float;
   when 3 => as_String : String (1 .. 4);
   when others =>
     as_Stream_Element_Array :
     Ada.Streams.Stream_Element_Array
                                (1 .. 4);
  end case;
end record;
pragma unchecked_Union(foo);
```

The above construct is an exact equivalent of C's unions.

Conference Calendar

Dirk Craeynest

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

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

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

2010

April 06-09 — 21st **Australian Software Engineering Conference** (ASWEC'2010), Auckland, New Zealand. Topics include: Empirical Research in Software Engineering; Formal Methods; Legacy Systems and Software Maintenance; Measurement, Metrics, Experimentation; Object and Component-Based Software Engineering; Open Source Software Development; Quality Assurance; Real-Time and Embedded Software; Software Design and Patterns; Software Engineering Education; Software Re-use and Product Development; Software Risk Management; Software Security, Safety and Reliability; Software Verification and Validation; Software Vulnerabilities; Standards and Legal Issues; Testing, Analysis and Verification; etc.

April 06-10 — 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.

April 13-15 — 2nd **NASA Formal Methods Symposium** (NFM'2010), Washington, D.C., USA. Topics include: Formal verification, including theorem proving, model checking, and static analysis; Model-based development; Techniques and algorithms for scaling formal methods, such as parallel and distributed techniques; Empirical evaluations of formal methods techniques for safety-critical systems; etc. Deadline for registration: April 9, 2010.

☺ 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.

April 13-16 — ACM-BCS **Visions of Computer Science conference** (Visions'2010), Edinburgh, UK. Topics include: Programming Methods and Languages; Software Engineering, and System Design Tools; Distributed and Pervasive Systems; Robotics; Medical Applications; etc.

April 15-16 — 2nd **International Workshop on Software Engineering for Resilient Systems** (SERENE'2010), London, UK. Topics include: methods and tools that ensure resilience to faults, errors and malicious attacks; Requirements, software engineering & re-engineering for resilience; Verification and validation of resilient systems; Error, fault and exception handling in the software life-cycle; Frameworks, patterns and software architectures for resilience; etc. Deadline for registration: April 8, 2010 (spring school), April 10, 2010 (workshop).

☺ 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.

> ☺ 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; 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.

April 26-29 22nd Annual **Systems and Software Technology Conference** (SSTC'2010), Salt Lake City, Utah, USA.

April 27 **Introduction to SPARK 9 webinar**, Internet. Topics include: the new features of the AdaCore/Altran Praxis joint offering - SPARK Pro 9.

☺ April 27 EDCC2010 - **Workshop on Critical Automotive applications: Robustness and Safety** (CARS'2010), Valencia, Spain. Topics include: design, implementation and operation of critical automotive applications and systems, with particular emphasis on dependability issues, software engineering for robustness, security and safety issues, real-time embedded systems technologies, architectural solutions and development processes for dependable automotive embedded systems.

☺May 01-08 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.

> ☺ May 01 3rd **International Workshop on Multicore Software Engineering** (IWMSE'2010). Topics include: Frameworks for multicore software; Modeling techniques for multicore software; Software components and composition; Programming models and their impact on multicore software engineering; Testing and debugging parallel applications; Software reengineering for parallelism; Development environments and tools for multicore software; Experience reports from research projects or industrial projects; etc.
>
> May 02 6th **International Workshop on Software Engineering for Secure Systems** (SESS'2010). Topics include: Architecture and design of trustworthy systems, Separation of the security concern in complex systems, Secure programming, Static analysis for security, Trustworthiness verification and clearance, Defining and supporting the process of building secure software, etc.

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.

> May 31 3rd **International Workshop on Software Engineering for Computational Science and Engineering** (SECSE'2010). Topics include: Lessons learned from the development of CSE applications; The use of empirical studies to better understand the environment, tools, languages, and processes used in CSE application development and how they might be improved; etc.
>
> May 31 7th **International Workshop on Practical Aspects of High-level Parallel Programming** (PAPP'2010). Topics include: high-level parallel language design, implementation and optimisation; modular, object-oriented, functional, logic, constraint programming for parallel, distributed and grid computing systems; industrial uses of a high-level parallel language; etc.

May 31 – June 02 10th Annual **International Conference on New Technologies of Distributed Systems** (NOTERE'2010), Tozeur, Tunisia. Topics include: Domain Specific languages for distributed systems; Reliability and scalability of distributed systems; Modeling, Formal and Semi-formal methods, and tools for distributed systems; Software and middleware for embedded distributed systems and their applications; etc.

☺ June 01-04 **DAta Systems In Aerospace** (DASIA'2010), Budapest, Hungary.

June 05 **Ada-Belgium Spring 2010 Event**, Leuven, Belgium. Includes: 2010 Ada-Belgium General Assembly and Workshop on Creating Debian Packages of Ada Software. Deadline for registration: May 14, 2010.

☺ June 06-09	10th **International Conference on State-of-the-art in Scientific and Parallel Computing** (PARA'2010), Reykjavík, Iceland. Topics include: High Performance Computing (HPC) programming tools, HPC software engineering, Parallel computing in physics, Scientific computing tools, etc. Deadline for submissions: April 1, 2010 (abstracts).	
June 07-09	DisCoTec2010 - 10th IFIP **International Conference on Distributed Applications and Interoperable Systems** (DAIS'2010), Amsterdam, The Netherlands. Theme: "Applications and services for a complex world". Topics include: models, methodology and concepts supporting distributed applications; middleware and software engineering techniques supporting distributed applications; etc.	
June 14-15	2nd USENIX **Workshop on Hot Topics in Parallelism** (HotPar'2010), Berkeley, California, USA. Topics include: the broad impact of multicore computing in all fields, including application design, languages and compilers, systems, and architecture.	
♦ June 14-18	15th **International Conference on Reliable Software Technologies - Ada-Europe'2010**, Valencia, Spain. Sponsored by Ada-Europe, in cooperation with ACM SIGAda. Deadline for early registration: May 24, 2010.	
June 16-18	**Code Generation 2010**, Cambridge, UK. Topics include: Model-driven software development, Tool and technology development and adoption, Code Generation and Model Transformation tools and approaches, Defining and implementing modelling languages, Language evolution and modularization, Case studies, etc.	
☺ June 21-23	**Automotive - Safety & Security 2010**, Stuttgart, Germany. Organized by Gesellschaft für Informatik mit den Fachgruppen Ada, etc, and Ada-Deutschland. Topics include (in German): Zuverlässigkeit und Sicherheit für fahrbetriebs-kritische Software und IT-Systeme; Evaluation und Zertifizierung von Sicherheitseigenschaften automobiler Firmware/Software; Multi-Core-Architekturen; Zuverlässige Echtzeit-Betriebssysteme; Fortschritte bei Normen und Standardisierungen; etc.	
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.	
☺ June 21-25	24th **European Conference on Object Oriented Programming** (ECOOP'2010), Maribor, Slovenia. Topics include: research results or experience in all areas relevant to object technology, including work that takes inspiration from, or builds connections to, areas not commonly considered object-oriented; such as: Analysis, design methods and design patterns; Concurrent, real-time or parallel systems; Distributed systems; Language design and implementation; Programming environments and tools; Type systems, formal methods; Compatibility, software evolution; Components, modularity; etc.	
	☺ June 21	1st **International Workshop on Real-time Object-Oriented TechnologieS** (ROOTS'2010). Topics include: New real-time programming paradigms and language features; Reports on R&D progress in the field; Emerging tools and trends; Emerging standards (or the need for new standards); Practical experiences, particularly from industry; Safety-critical software certification; etc. Deadline for paper submissions: April 19, 2010.
	☺ June 22	2nd **International Workshop on Distributed Objects for the 21st Century** (DO21'2010). Topics include: constructive ideas, new programming paradigms, novel programming language abstractions, domain specific languages, frameworks, tools or architectures for distributed object computing; State-of-the-art distributed object systems; Multi-paradigm approaches; Alternative (non-OO) approaches (and their pros/cons); etc. Deadline for paper submissions: April 19, 2010.
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.	
June 23-25	CompArch2010 - 1st **International Symposium on Architecting Critical Systems** (ISARCS'2010), Prague, Czech Republic. Topics include: Rigorous development, Fault tolerance based on the	

	architecture, Safety-critical systems & architecture, Secure systems & architecture, Relevant domains with critical systems, Industrial needs, etc.
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 on Objects, Models, Components, Patterns** (TOOLS Europe'2010), Málaga, Spain. Topics include: Object technology, including programming techniques, languages, tools; Distributed and concurrent object systems; Real-time object-oriented programming and design; Experience reports, including efforts at standardisation; Applications to safety- and security-related software; Trusted and reliable components; Domain specific languages and language design; Language implementation techniques, compilers, run-time systems; Practical applications of program verification and analysis; etc.
July 05-12	37th **International Colloquium on Automata, Languages and Programming** (ICALP'2010), Bordeaux, France.
☺ July 07-09	9th **International Symposium on Parallel and Distributed Computing** (ISPDC'2010), Istanbul, Turkey. Topics include: Parallel Computing; Distributed Systems Methodology and Networking; Parallel Programming Paradigms and APIs; Tools and Environments for Parallel Program Analysis; Task Scheduling and Load Balancing; Performance Management in Parallel and Distributed Systems; Distributed Software Components; Real-time Distributed and Parallel Systems; Security in Parallel and Distributed Systems; Fault Tolerance in Parallel and Distributed Systems; Parallel Scientific Computing and Large Scale Simulations; Parallel and Distributed Applications; etc.
July 12-14	2010 **International Conference on Software Engineering Theory and Practice** (SETP'2010), Orlando, Florida, USA. Topics include: Software development, maintenance, and other areas of software engineering and related topics.
July 15-19	22nd **International Conference on Computer Aided Verification** (CAV'2010), Edinburgh, UK. Topics include: Algorithms and tools for verifying models and implementations, Program analysis and software verification, Applications and case studies, Verification in industrial practice, etc.
	☺ July 20-21 **Workshop on Exploiting Concurrency Efficiently and Correctly** ((EC)^2). Topics include: deficiencies in current languages and tools; multi-core software design, correctness issues, and correctness approaches; programming languages and paradigms that facilitate concurrency exploitation; novel approaches for teaching concurrency; significant case studies; etc.
July 22-24	5th **International Conference on Software and Data Technologies** (ICSOFT'2010), Athens, Greece. Topics include: Software Engineering, Programming Languages, Distributed and Parallel Systems, etc.
July 25-28	29th Annual ACM SIGACT-SIGOPS **Symposium on Principles of Distributed Computing** (PODC'2010), Zurich, Switzerland. Topics include: multiprocessor and multi-core architectures and algorithms; synchronization protocols, concurrent programming; fault-tolerance, reliability, availability; middleware platforms; distributed data management; security in distributed computing; specification, semantics, verification, and testing of distributed systems; etc. Deadline for submissions: April 27, 2010 (brief announcements).
☺ August 23-25	16th IEEE **International Conference on Embedded and Real-Time Computing Systems and Applications** (RTCSA'2010), Macau SAR, P.R.China. Topics include: Software design for heterogeneous multi-core embedded platform, Multi-thread programming for multi-core embedded platform, Embedded system design practices, Real-time scheduling, Timing analysis, Programming languages and run-time systems, Middleware systems, Design and analysis tools, Case studies and applications, etc. Deadline for submissions: April 9, 2010. Deadline for early registration: June 25, 2010.
☺ Aug 31 – Sep 03	16th **International European Conference on Parallel and Distributed Computing** (Euro-Par'2010), Ischia, Italy. Topics include: all aspects of parallel and distributed computing, such as Support tools and environments, Scheduling, High performance compilers, Distributed systems and algorithms, Parallel and distributed programming, Multicore and manycore programming, Theory and algorithms for parallel computation, etc.

☺ Sep 11-15 — 19th **International Conference on Parallel Architectures and Compilation Techniques** (PACT'2010), Vienna, Austria. Topics include: ground-breaking research related to parallel systems ranging across instruction-level parallelism, thread-level parallelism, multiprocessor parallelism and large scale systems, such as Parallel computational models; Compilers and tools for parallel computer systems; Support for concurrency correctness in hardware and software; Parallel programming languages, algorithms and applications; Middleware and run-time system support for parallel computing; Reliability and fault tolerance for parallel systems; Modeling and simulation of parallel systems and applications; Parallel applications and experimental systems studies; Case studies of parallel systems and applications; etc. Deadline for submissions: April 3, 2010 (tutorials, workshops).

☺ Sep 13-16 — 39th **International Conference on Parallel Processing** (ICPP'2010), San Diego, California, USA. Topics include: compilers and languages, etc.

September 20-24 — 25th IEEE/ACM **International Conference on Automated Software Engineering** (ASE'2010), Antwerp, Belgium. Topics include: Component-based systems; Maintenance and evolution; Model-based software development; Model-driven engineering and model transformation; Modeling language semantics; Open systems development; Product line architectures; Program understanding; Program transformation; Re-engineering; Specification languages; Software architecture and design; Testing, verification, and validation; etc. Deadline for submissions: May 17, 2010 (tool demonstration papers).

> ☺ Sep 20 — 3rd **International Workshop on Academic Software Development Tools** (WASDeTT'2010). Topics include: How to integrate and combine independently developed tools? What are the positive lessons learned and pitfalls in building tools? What are effective techniques to improve the quality of academic tools? What particular languages and paradigms are suited to build tools? Deadline for submissions: July 19, 2010.

☺ Sep 20-21 — 15th **International Workshop on Formal Methods for Industrial Critical Systems** (FMICS'2010), Antwerp, Belgium. 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. Deadline for submissions: April 10, 2010 (abstracts), April 18, 2010 (papers).

September 20-22 — 15th **European Symposium on Research in Computer Security** (ESORICS'2010), Vouliagmeni, Athens, Greece. Topics include: Accountability, Information Flow Control, Formal Security Methods, Language-based Security, Security Verification, etc. Deadline for submissions: April 1, 2010.

☺ Sep 27-29 — CBSoft'2010 - 14th **Brazilian Symposium on Programming Languages** (SBLP'2010), Salvador, Bahia, Brazil. Topics include: Programming language design and implementation, Design and implementation of programming language environments, Object-oriented programming languages, Program transformations, Program analysis and verification, Compilation techniques, etc. Deadline for submissions: May 17, 2010 (abstracts), May 24, 2010 (papers).

Sep 29 – Oct 01 — 9th **International Conference on Software Methodologies, Tools and Techniques** (SoMeT'2010), Yokohama, Japan. Topics include: Software methodologies, and tools for robust, reliable, non fragile software design; Software developments techniques and legacy systems; Automatic software generation versus reuse, and legacy systems; Intelligent software systems design, and software evolution techniques; Agile Software and Lean Methods; Software optimization and formal methods for software design; Software maintenance; Software security tools and techniques, and related Software Engineering models; Formal techniques for software representation, software testing and validation; Software reliability, and software diagnosis systems; Model Driven Development (DVD), code centric to model centric software engineering; etc.

October 10-13 — 9th **International Conference on Generative Programming and Component Engineering** (GPCE'2010), Eindhoven, The Netherlands. Topics include: Generative techniques for Product-line architectures, Distributed, real-time and embedded systems, Model-driven development and architecture, Safety critical systems; Component-based software engineering (Reuse, distributed platforms and middleware, distributed systems, evolution, patterns, development methods, formal methods, etc.); Integration of generative and component-based approaches; Industrial applications; etc. Deadline for submissions: May 17, 2010 (abstracts), May 24, 2010 (papers).

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

♦ October 24-28 — ACM **SIGAda Annual International Conference on Ada and Related Technologies** (**SIGAda'2010)**, Fairfax, Virginia, USA (a suburb of Washington, DC). Sponsored by ACM SIGAda, in cooperation with SIGBED, SIGCAS, SIGCSE, SIGPLAN, Ada-Europe, and the Ada Resource Association. Deadline for submissions: June 25, 2010 (technical articles, extended abstracts, experience reports, panel sessions, industrial presentations, workshops, tutorials).

☺ Nov 01-03 — 29th IEEE **International Symposium on Reliable Distributed Systems** (SRDS'2010), Delhi, India. Topics include: security, safety-critical systems and critical infrastructures, fault-tolerance in embedded systems, analytical or experimental evaluations of dependable distributed systems, formal methods and foundations for dependable distributed computing, etc. Deadline for submissions: April 10, 2010 (full papers).

November 08-12 — 13th **Brazilian Symposium on Formal Methods** (SBMF'2010), Natal, Rio Grande do Norte, Brazil. Topics include: Formal aspects of popular languages and methodologies; Logics and semantics of programming and specification languages; Type systems in computer science; Formal methods integration; Code generation; Formal design methods; Abstraction, modularization and refinement techniques; Techniques for correctness by construction; Formal methods and models for real-time, hybrid and critical systems; Models of concurrency, security and mobility; Theorem proving; Static analysis; Software certification; Teaching of, for and with formal methods; Experience reports on the use of formal methods; Industrial case studies; Tools supporting the formal development of computational systems; Development methodologies with formal foundations; etc. Deadline for submissions: June 10, 2010 (papers).

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

2011

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

15th International Conference on

RELIABLE SOFTWARE TECHNOLOGIES ADA-EUROPE 2010

VALENCIA, SPAIN, 14-18 JUNE

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

In cooperation with
ACM SIGAda

UNIVERSIDAD
POLITECNICA
DE VALENCIA

GENERAL INFORMATION

The 15th International Conference on Reliable Software Technologies – Ada-Europe 2010 will take place in Valencia, Spain, on 14-18 June 2010. The conference has established itself as an international forum for providers, practitioners and researchers into reliable software technologies. Following tradition, the conference will span a full week, with a three-day technical program from Tuesday to Thursday accompanied by vendor exhibitions, and a string of parallel tutorials on Monday and Friday.

ABOUT THE VENUE

Valencia, situated on the Mediterranean coast of eastern Spain, is the capital city of the autonomous region Comunidad Valenciana. Not many cities are capable of so harmoniously combining a fine array of sights from the distant past with innovative constructions now being erected. Valencia, founded in 138 BC, is one of these fortunate few. From the remains of the Roman forum located in today's Plaza de la Virgen to the emblematic City of Arts and Sciences, this town has transformed its physiognomy over the years while preserving its monuments from the past.

Sightseeing around the city begins in the old quarter, where the conference venue is located. Still standing as proof of the old defending wall are the graceful Torres de Serranos, the spacious Torres de Quart and some remains of the apron wall in the basement of the Valencia Institute of Modern Arts. On the old riverbed of the river Turia lie the nursery gardens, along with the Fine Arts Museum and the modern part of the city. Life in the city spreads down to the seafront with the harbor and the beaches of Las Arenas and La Malvarrosa.

SOCIAL PROGRAM

The social program will schedule two events: a welcome reception on Tuesday in the Botanical Garden, and a banquet dinner on Wednesday in a typical *Masía* —an old country house— at a very short distance from the city.

Ada-Europe 2010 Valencia

OVERVIEW OF THE WEEK

	Morning	Late Morning	Early Afternoon	Afternoon
Monday **14 June** Tutorials	Developing High-Integrity Systems with GNATforLEON/ORK+ *J. de la Puente, J. Zamorano*		Hypervisor Technology for Building Safety-Critical Systems: XtratuM *I. Ripoll, A. Crespo*	
	Software Design Concepts and Pitfalls *W. Bail*		How to Optimize Reliable Software *I. Broster*	
	Using Object-Oriented Technologies in Secure Systems *J. P. Rosen*		Developing Web-aware Applications in Ada with AWS *J. P. Rosen*	
Tuesday **15 June** Sessions & Exhibition	**Keynote Talk** What to Make of Multicore Processors for Reliable Real-Time Systems? *Theodore Baker*	**Multicores and Ada**	**Software Dependability** **Vendor Session**	**Critical Systems** **Vendor Session**
Wednesday **16 June** Sessions & Exhibition	**Keynote Talk** Control Co-design: Algorithms and their Implementation *Pedro Albertos*	**Real-Time Systems**	**Industrial Presentations**	**Industrial Presentations**
Thursday **17 June** Sessions & Exhibition	**Keynote Talk** Ada: Made for the 3.0 World *James Sutton*	**Language Technology**	**Industrial Presentations**	**Distribution and Persistency**
Friday **18 June** Tutorials	SPARK: the Libre Language and Toolset for High-Assurance Software *R. Chapman*			
	C#, .NET and Ada: Keeping the Faith in a Language-Agnostic Environment *B. Brosgol, J. Lambourg*			

FURTHER INFORMATION

The conference web site gives full and up to date details of the program and the venue, including travel advice, maps and hotels close by. A limited number of rooms have been blocked for the conference in several hotels nearby the conference venue, but please be prompt in booking your accommodation since the demand is high in Valencia for the conference dates. Online registration is open, with reduced fees until May 24.

For Exhibiting and Sponsoring details please contact the Exhibition Chair, Ahlan Marriott, by email at *Ada@white-elephant.ch*. A sliding scale of sponsorship provides a range of benefits. All levels include display of your logo on the conference web site and in the program. The lowest level of support is very affordable.

The organizers are grateful to the exhibitors and sponsors of the conference (preliminary list)

AdaCore The GNAT Pro Company · atego · ALTRAN PRAXIS · Ellidiss Software TNI Europe Limited

GOBIERNO DE ESPAÑA · MINISTERIO DE CIENCIA E INNOVACIÓN · GENERALITAT VALENCIANA CONSELLERIA D'EDUCACIÓ · DISCA DEPARTAMENT D'INFORMÀTICA DE SISTEMES I COMPUTADORS · etsinf Escola Tècnica Superior d'Enginyeria Informàtica · ai2 INSTITUTO DE AUTOMÁTICA E INFORMÁTICA INDUSTRIAL

Call for Technical Contributions – SIGAda 2010

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

Fairfax, Virginia (Washington DC Area), USA
October 24-28, 2010

Submission Deadline: June 25, 2010
Sponsored by ACM SIGAda
http://www.acm.org/sigada/conf/sigada2010

SUMMARY: Reliability, safety, and security are among the most critical requirements of contemporary software. The application of software engineering methods, tools, and languages all interrelate to affect how and whether these requirements are met.

Such software is in operation in many application domains. Much has been accomplished in recent years, but much remains to be done. Our tools, methods, and languages must be continually refined; our management process must remain focused on the importance of reliability, safety, and security; our educational institutions must fully integrate these concerns into their curricula.

The conference will gather industrial and government experts, educators, software engineers, and researchers interested in developing, analyzing, and certifying reliable, safe, long-lived, secure software. We are soliciting technical papers and experience reports with a focus on, or comparison with, Ada.

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

POSSIBLE TOPICS INCLUDE BUT ARE NOT LIMITED TO:

- Challenges for developing reliable, safe, long-lived, secure software
- Transitioning to Ada 2005
- Ada and SPARK in the classroom and student laboratory
- Language selection for highly reliable systems
- Mixed-language development
- Use of high reliability subsets or profiles such as MISRA C, Ravenscar, SPARK
- High-reliability standards and their conformance to DO-178B and preparing for DO-178C
- Software process and quality metrics
- System of Systems
- Real-time networking/quality of service guarantees
- Real-Time Parallel Processing
- Analysis, testing, and validation
- Use of ASIS for new Ada tool development
- High-reliability development experience reports
- Static and dynamic analysis of code
- Integrating COTS software components
- System Architecture & Design
- Information Assurance
- Ada products certified against Common Criteria / Common Evaluation Methodology
- Distributed systems
- Fault tolerance and recovery
- Performance analysis
- Implementing Service Oriented Architecture
- Embedded Hard Real-Time Systems

KINDS OF TECHNICAL CONTRIBUTIONS:

TECHNICAL ARTICLES present significant results in research, practice, or education. Articles are typically 10-20 pages in length. These papers will be double-blind refereed and published in the Conference Proceedings and in ACM Ada Letters. The Proceedings will be entered into the widely-consulted ACM Digital Library accessible online to university campuses, ACM's 80,000 members, and the software community.

EXTENDED ABSTRACTS discuss current work for which early submission of a full paper may be premature. If your abstract is accepted, you will be expected to produce a full paper, which will appear in the proceedings. Extended abstracts will be double-blind refereed. In 5 pages or less, clearly state the work's contribution, its relationship with previous work by you and others (with bibliographic references), results to date, and future directions.

EXPERIENCE REPORTS present timely results on the application of Ada and related technologies. Submit a 1-2 page description of the project and the key points of interest of project experiences. Descriptions will be published in the final program or proceedings, but a paper will not be required.

PANEL SESSIONS gather a group of experts on a particular topic who present their views and then exchange views with each other and the audience. Panel proposals should be 1-2 pages in length, identifying the topic, coordinator, and potential panelists.

INDUSTRIAL PRESENTATIONS Authors of industrial presentations are invited to submit a short overview (at least 1 page in size) of the proposed presentation to the Industrial Committee Chair by August 1st 2010. The authors of selected presentations shall prepare a final short abstract and submit it to the Committee Chair by October 1st^{h}, 2010, aiming at a 20-minute talk. The authors of accepted presentations will be invited to submit corresponding articles for publication in the ACM Ada Letters.

WORKSHOPS are focused work sessions, which provide a forum for knowledgeable professionals to explore issues, exchange views, and perhaps produce a report on a particular subject. A list of planned workshops and requirements for participation will be published in the Advance Program. Workshop proposals, up to 5 pages in length, will be selected by the Program Committee based on their applicability to the conference and potential for attracting participants.

TUTORIALS offer the flexibility to address a broad spectrum of topics relevant to Ada, and those enabling technologies which make the engineering of Ada applications more effective. Submissions will be evaluated based on relevance, suitability for presentation in tutorial format, and presenter's expertise. Tutorial proposals should include the expected level of experience of participants, an abstract or outline, the qualifications of the instructor(s), and the length of the tutorial (half-day or full-day). Tutorial presenters receive complimentary registration to the other tutorials and the conference.

HOW TO SUBMIT: Send contributions by **June 25, 2010**, in Word, PDF, or text format as follows:

Technical Articles, Extended Abstracts, Experience Reports, and Panel Session Proposals to: Program Chair, Lt. Col. Jeff Boleng (Jeff.Boleng@usafa.edu)

Tutorial Proposals to: Tutorials Chair, Dr. Robert Pettit (RPettit@gmu.edu)

Industrial Presentations Proposals to: Industrial Committee Chair, Prof. Liz Adams (adamses@cs.jmu.edu)

FURTHER INFORMATION:

CONFERENCE GRANTS FOR EDUCATORS: The ACM SIGAda Conference Grants program is designed to help educators introduce, strengthen, and expand the use of Ada and related technologies in school, college, and university curricula. The Conference welcomes a grant application from anyone whose goals meet this description. The benefits include full conference registration with proceedings and registration costs for 2 days of conference tutorials/workshops. Partial travel funding is also available from AdaCore to faculty and students from GNAT Academic Program member institutions, which can be combined with conference grants. For more details visit the conference web site or contact Prof. Michael B. Feldman (mfeldman@gwu.edu)

OUTSTANDING STUDENT PAPER AWARD: An award will be given to the student author(s) of the paper selected by the program committee as the outstanding student contribution to the conference.

SPONSORS AND EXHIBITORS: Please contact Greg Gicca (Gicca@AdaCore.com) and Kristen Ferretti (kef@ocsystems.com) for information about becoming a sponsor and/or exhibitor at SIGAda 2010.

IMPORTANT INFORMATION FOR NON-US SUBMITTERS: International registrants should be particularly aware and careful about visa requirements, and should plan travel well in advance. Visit the conference website for detailed information pertaining to visas.

ANY QUESTIONS?:

Please submit your questions to Conference Chair Alok Srivastava (alok.srivastava@auatac.com) or Local Arrangements Co-Chairs Avtar Dhaliwal (avtar_dhaliwal@gencosystems.com) and Florence Gubanc (fgg@ocsystems.com).

Opinion: The Word 'Coding' Considered Harmful

Brian Tooby *
High Integrity Systems Ltd, Sawbridgeworth, Herts.

Let us bury the word 'coding' as an activity relevant to Ada. Coding is the process of transforming the comprehensible into the incomprehensible, and is relevant only where machines are programmed in less abstract (or less meaningful) terms than Ada permits.

We should recognize its equivalent today as (computer aided) design: what comes next is performed by the Ada development tools, and is Computer Implemented Manufacture. In fact the process of software systems development is now one of CAD/CIM, contrasted to the more widely understood hardware equivalent of CAD/CAM. Recognition of this fact in textbook descriptions of the life-cycle would have a beneficial clarifying influence on the development of the software engineering industry. This might even go so far as to change the nature of courses offered by schools, polytechnics and universities, resulting eventually in an increased number of badly needed qualified recruits to the industry.

Programmers are supposed to be people who 'code'. Anyone from child to systems analyst can and does program a computer as part of work or play: that's a mechanistic description of the task. Ada teaches abstraction, so let's talk about software and systems design.

For the purposes of project management, team working and modular engineering, it is still worth distinguishing the activities of 'architectural' or 'external' design from that of 'detailed' or 'internal' design. The informal nature of this distinction is reflected in the fact that to carry out (completely) the first, one has to do (or imagine, or have available the results of) the second.

The distinction is worthwhile because the human creative process is helped by having an early but hazy view of the whole system, and clarifying and correcting this picture as detail is filled in; however, the actual behaviour of the system, and its ultimate modularization, will depend on the mass of detail. Recognizing this fact shows that while formal specification techniques are of great importance, their role in the cloud-crystallization process itself is limited; one cannot specify in advance a full technical contract, or specification, for a (complex) component of a larger system, and give it to another person or team to 'implement'. Semi-formal techniques, aided by paper analysis or prototyping, are used instead.

However, the picture changes as we become able to construct a design from existing modules, particularly when these are accompanied by a full formal (static and dynamic) specification. In the near future this will be approximated, often surprisingly closely, by the Ada specification and the Ada body respectively. The challenge will be to improve on this, at least to the extent where it becomes possible to bring Ada module designs entirely into the software CAD domain.

So let us say goodbye to 'programmers' and 'coding' in the world of software and systems engineering. The internal design of a module plays a much more subtle and influential role in the development of a complete system than those words suggest. Some improved clarity in describing our own profession might pay enormous dividends.

**** Affiliation and contacts as in the original publication***

E. Schonberg, "Origins and history of GNAT"
Originally printed in *Ada-Europe News*, Issue 20, March 1995

Origins and history of GNAT

Edmond Schonberg *

New York University, Courant Institute, 251 Mercer Street, New York, NY 10012; email: schonberg@cs.nyu.edu.

Introduction

The GNAT project at New York University is in the process of completing a compiler for Ada95. Because GNAT (the GNU-NYU Ada Translator) uses the GCC retargettable code generator, it is relatively easy to port, and it already runs successfully on most modem machines, from RISC workstations to i86-based personal computers. The project is scheduled to be completed in June 1995. GNAT has been developed in collaboration with, and following the guidelines of the Free Software Foundation, and is distributed freely, with sources, electronically over the Internet and on various physical media. An independent software organization, Ada Core Technologies (ACT) has been created to insure the ongoing maintenance of GNAT beyond June 1995. ACT plans to formally validate GNAT on various platforms in the second quarter of 1995. Validated GNAT compilers will continue to be freely available to the software community.

Early Ada activities at New York University

The GNAT team at New York University has been involved with the definition and the implementation of Ada for close to 15 years, from the very first release of the preliminary reference manual for Ada83. The NYUADA project, as it was called then, was at first interested in optimization techniques for high-level languages. It was evident at once that the preliminary RM provided only a very partial definition of the semantics of the language, and that the presence of concurrency affected the potential meaning of all operations in a critical way. In an effort to provide an operational semantics of the language that could serve as a framework for optimization studies, Robert Dewar wrote an interpreter for Ada that focused on the most novel aspects of the language, the interaction between flow of control, tasking, and exceptions. In order to make this operational definition as perspicuous as possible, the interpreter was written in SETL, a very-high level language developed at New York University, and whose basic constructs are those of the theory of finite sets.

The structure of the interpreter was that of a denotational definition, in the sense that its basic data structures were concrete representations of the continuation and the store, and the semantics of each primitive instruction was described in terms of explicit modifications of these. Tasking was described operationally by assigning a separate store and continuation to each task, and the granularity of the primitives in the interpreter provided a coherent description of the interaction between sequential flow of control, task communication, and the raising of exceptions.

Ada/Ed

This initial interpeter was less than 2000 lines long, and took as input a handwritten parse tree for Ada. In order to provide a definition of the static semantics as well, it was decided to complete the interpreter with a front-end that would produce a semantically analyzed abstract syntax tree. Gerry Fisher (now at IBM Corp.) used SETL to build an LALR parser for Ada, and to develop a series of novel algorithms for error recovery. Edmond Schonberg wrote the semantic analyzer, also in SETL. The resulting system was a full-fledged translator for the language, and was used by the ACVC team, headed by John Goodenough, in the development of the validation suite. The system was by any standards remarkably slow (10 lines/sec compilation rate, 1 line/sec execution rate on a VAX/780) but in 1982-1983 the only readily available translator for Ada, and was used as a teaching tool by a number of universities and industrial sites. Dubbed Ada/Ed, to emphasize its educational role, the system was the first validated translator for Ada. Its performance was the source of many jokes ("Ada/Ed is used for the real-time simulation of paper-and-pencil calculations") but it was for a while a usable educational resource, and it also fulfilled its original purpose of providing an informal operational definition of the language, complementary to the Reference Manual and to the Ada Compiler validation suite.

Ada/Ed was also a complete prototype of a translator, and after 1984 the activities of the project turned towards a large-scale experiment in Software prototyping. The SETL text of Ada/Ed was used as the design document for a conventional byte-code interpreter, written in C, and whose performance was comparable to that of a conventional BASIC interpreter. This was done in two steps: first the denotational interpreter was transformed into a conventional interpreter operating on a linear code stream, and a bona fide code generator was added to produce the stream. This new component was written in SETL as well. Finally, the SETL system was translated into C.

The result of this effort was Ada/Ed-C, which was also validated under the ACVC. Dave Shields, (now at IBM Corp.) led the effort of Setl-to-C translation, writing the implementation of a variety of set primitives and setting the guidelines so that the translation from Setl to C could be performed almost mechanically, requiring not detailed information about the Ada/Ed algorithms themselves. The design of the code generator and run-time library was the work of Philippe Kruchten (now at Rational) and Jean-Pierre Rosen (now at Adalog in Paris). The C version was

** Affiliation and contacts as in the original publication* completed by Bernard Banner and Gail Schenker, who are now senior members of the GNAT project.

Ada/Ed-C was enhanced by Michael Feldman and his students, at George Washington University. They added a user-friendly shell, akin to the programming environment of Turbo-Pascal, that made the system much more accessible to beginning students. The result, called GWU-Ada/Ed, has been used widely in introductory Ada courses worldwide. GWU-Ada/Ed owes its success to the combination of ease of use, relative completeness, and free availability.

When the revision of the language started, the venerable Ada/Ed system found additional use as a prototyping tool. A major concern in the design of Ada95 has been to minimize the amount of surprises that implementers may face in making the transition between Ada83 compilers and Ada95 compilers. It is worth noting that in spite of what the designers of Ada83 considered a conservative language design, Ada did in fact push the limits of compiler technology in the previous decade, and robust compilers appeared on the market later than expected. To avoid a similar fate for Ada95 compilers, the Ada9X project office decided to fund several prototyping efforts, to evaluate the complication inherent in the constructs being proposed for the new language.

The NYU team focused on object-oriented constructs (tagged types, type extensions, dynamic dispatching) and on the new generic facilities, which are closely related. These features (as they stood in 1991-92) were implemented in SETL within the old Ada/Ed system, and proved to be relatively straightforward to build, confirming the statements of the Mapping and Revision team that "the difficult parts of Ada9X are the Ada83 features".

Towards Ada/9x

In mid-1992, the NYU team received a contract from the Ada9X project office and the US Air Force, to build a free compiler for Ada9X, using the GCC technology for the code generation phase. The purpose of the system was to place the hands of Ada users, at the earliest possible date, a tool with which they could explore the new language. The emphasis was to be on early availability rather than completeness, and it was understood that the resulting system might not be validated and that early implementation of object-oriented features was more important than full semantic legality checks. In addition, the choice of GCC meant that we would adopt the copyright policies of the Free Software Foundation, and thus make the system available at no cost to the largest number of potential users.

In addition to the distribution advantages, and to our sympathy towards the goals of the Free Software foundations, there were two important reasons for this choice. The first was technical: the quality of the code generated by the GCC compiler system, and its remarkable retargetability. The second was that the chief maintainer of GCC, Richard Kenner, was a senior staff member of another project at New York University, and was eager to extend GCC to other languages, other platforms, and other application areas.

Our initial technical discussions with Richard Stallman, the founder and guiding spirit of the Free Software Foundation, led us to another important technical choice: the compilation model for GNAT should look as much as possible like the compilation model for other languages. This meant that interfacing with other languages would be much simpler. It also meant that the conventional Ada model of a monolithic program around which all compilations revolve would have to be substantially altered. Under the goading of Stallman, we found a way of enforcing the "one compilation, one object file" model, and at the same time of preserving the semantics of the Ada compilation rules, which turn out not to need a centralized library. The result is a system that is much more comfortable to non-Ada users, and that works well in a mixed-language programming environment.

The Development of GNAT

Another early technical (and political) decision was to use Ada itself to write GNAT. Even though the rest of GCC is written in C (close to 500K lines for the various front-ends, the common back-end, utilities, and machine description files) it was not acceptable to use C itself for the Ada component. We started by writing a compiler for a small subset of Ada83, and used a commercial compiler to compile it. In June 1993, the subset was sufficiently complete to compile itself, and the compiler was bootstrapped in time for the Ada-Europe conference. The use of Ada9X features in the compiler itself has grown substantially, as the implementation has become more complete, and GNAT currently can only be compiled with itself. The compiler uses child libraries very heavily, and tagged types more sparingly. Very little code makes uses of dynamic dispatching.

The use of Ada for the front-end establishes a clear division of concerns between the two halves of the system, and allows us to program in our favorite language. However, it requires the construction of a new interface between a language specific front-end and a language-independent code generator. It turns out to be impractical to generate directly the abstract syntax tree used by other front-ends of GCC. This is due mostly to the semantic gap between Ada and C, the language for which the GCC internal form was designed. We chose instead to build the front-end around an AST that is well-suited to Ada, and to design a purely functional interface that traverses the Ada AST and generates on the fly the GCC tree that drives code generation. This tree transduction phase was dubbed Gigi (for Gnat-to-Gnu) and was originally designed by Franco Gasperoni, now at the Ecole Superieure de Telecommunications in Paris. Subsequent design and implementation of Gigi was the work of Brett Porter, Richard Kenner, Cyrille Comar, and other members of the GNAT team.

Software development in a global community

The Free Software Foundation is a non-profit organization founded by Richard Stallman, with the purpose of furthering the development of free software. The main products of the FSF are the extendable editor EMACS, the GCC suite of retargettable compilers, the run-time debugger GDB, and an operating system under development, originally called GNU (which is a self-referential acronym that stands for "GNU is Not Unix"). The model of free software distribution fostered by the FSF is diametrically opposed to the existence of software patents. The FSF distributes its products under rules that are intended to encourage their spread and prevent proprietary claims from being attached to any of them. These rules are embedded in a copyright notice attached to every product, the Gnu Public Licence (GPL, also known as "copyleft"). The GPL gives unlimited rights of copying to any user of the software, and enjoins any user that modifies the software and redistributes it, to distribute the modified sources as well.

The Free Software Foundation, by its very nature, has relied heavily on volunteer work. The maintenance of GCC and its various front-ends, as well as ports of the system to new machines and new operating systems, have depended on the efforts and the good will of many. The development of GNAT has benefited from the sane Internet-wide enthusiasm, and GNAT would have only a fraction of its impact on the Ada community if it weren't for the constant efforts of patient users. The distribution model of GCC is obviously a tremendous asset: enterprising users can examine the sources, spot errors, suggest improvements, volunteer better implementations, etc. Experienced GCC users can port the system to new platforms, leveraging on the ease with which GCC can be configured as a cross-compiler. Thanks to their efforts, versions of GNAT for Linux, FreeBSD, NetBSD, NextStep, Amiga-DOS, and on Solaris, as well as a cross-compiler for the 1750A, have already appeared. An extensive list of contributors appears in the following paper [1]. We must thank all of them, as well as those others whose names we do not have room to mention, and whose remarks have resulted in constant improvements in the quality of GNAT.

Towards a long-lived GNAT

The GNAT system has given the Ada community an opportunity to examine Ada95 even before the language received its ISO standardization, and long before GNAT itself was complete. The existence of a partial compiler has been of benefit to the user community, and the interest of this community has in turn facilitated the development and increasing polish of compiler itself. This appealing model of exploratory development must now give way to industrial practice. Serious use of any compiler by an industrial organization requires some guaranties about the ongoing support available for it. In the case of Ada, formal validation is required before mission-critical projects can use a given compiler, and repeated validations are required as the ACVC test suite evolves. The GNAT project at New York University is in no position to either validate GNAT or provide the long-term maintenance that users will need. Instead, we have decided to create a private organization whose purpose will be to maintain the compiler and insure that it remains freely available under the GPL model.

This organization, Ada Core Technologies, will offer maintenance contracts to industrial users of GNAT that require them. Enhancements and bug fixes that follow from this maintenance activity will be incorporated into the system, which we will continue to distribute freely over the Internet. We hope that the synergy between developers and users of GNAT will continue, and that GNAT will help spread the use of the best-designed modern programming language.

[1] Editor's note: "*The GNAT project: A GNU-Ada 9X Compiler*", Ada-Europe News, Issue 20, March 1995

We don't know nothing

John Barnes *

*John Barnes Informatics, 11 Albert Road, Caversham, Reading, RG4 7AN;
tel: 01734 474125; email: jgpb@jbinfo.demon.co.uk.*

I ruminate from time to time about how little we really know about software. We have gathered a lot of information rather in the sense that botanists gathered data on flora in the nineteenth century. But it is hard to see much absolute truth in it all.

The fact that we argue about the merits of one language over another must mean that there is no widespread understanding of any underlying principles against which languages can be measured. We are dominated by our own little experiences, by fashion and not by truth. But we do know a little and I will attempt to put forward some thoughts regarding our faint knowledge.

A big problem is that myopic bean-counters have no view of anything beyond the next budget. Thus we are forced to move forward by evolution rather than revolution. Evolution appears comforting, seems to preserve investment in old programs (which probably need rewriting anyway e.g. to get the date right), and distributes costs over space and time so that they are hidden from the beanie. Of course the beanie doesn't know they are hidden, the poor fool believes they don't exist.

But the costs of evolution are very high. They preserve outmoded working practices, force backward compatibility with old technology and as a consequence cause society to struggle with unnecessary complexity.

Evolution is like a stone. It tends to roll downhill, but unlike a stone it gathers lots of moss. As languages evolve they gather all sorts of stuff (FORTRAN is a prime example) and take years to throw off redundant features; but the central structure inevitably becomes fossilised.

But of course this is all my personal opinion and as an old friend once said with infinite wisdom "Opinions are like arseholes, everybody has one".

There is an analogy with physics and mathematics. Cast your mind back 100 years. It was a world of order and structure. Newton's laws had been solidly accepted for hundreds of years. Maxwell had cracked electromagnetic theory. And mathematics was seen as done with Whitehead's axiomatic approach,

Of course, there were a few very minor anomalies. The orbit of Mercury seemed not right and there was the black body radiation business. But generally, it was a clockwork world: "give us the data and we can compute the future".

And now; we "know" it is a world of chaos and doubt. Einstein showed that things weren't even straight. Dirac showed that God did play dice. And Goedel wrecked the foundations of mathematics by showing that things couldn't be proved.

What a mess. This has been a chaotic century for mathematics and physics; but now we have the Standard model but of course that raises more questions than it answers. What are quarks made of? Why is an electron so big? One thing we do seem to learn is that the more we know the more we know we don't know. And physicists still grope for that elusive Theory of Everything. But does God laugh as he lets us peel each layer off the onion? Does he create new layers on the fly as required to keep us humble?

Perhaps there is an analogy with software languages. ALGOL 60 was the revelation that there could be order. Perhaps ALGOL 68 was the classical world of software. It was pretty well understood and that seemed to be it.

But now where are we. Is OOP the "New Physics" of software? It certainly causes doubt and confusion as we grope for that unified theory of programming.

But how much of software language is based on God's real truth? As opposed to man's specific invention. Is there an ultimate knowable generic model of languages? And if so, is Ada 95 close to one instantiation of that model?

So what do we know? If you were asked to design a language from scratch, what would you feel confident about? Pretty little really. I contend that we really know only two things:

- assignment is not equality, and
- bracketed control structures are best.

Since assignment is not equality (and I think I feel comfortable that we know that), it seems to me that we should use a different symbol. Thus ALGOL 60 used := as distinct to =. After hundreds of years of mathematics it is clear that = is accepted for equality and that

```
X = X + 1
```

is always just false and cannot possibly mean anything else; so we need something other than = for assignment.

And the other thing is the choice between structures such as

```
if condition then one statement;
```

as in ALGOL 60, Pascal and C. And

```
if condition then several statements fi;
```

as in ALGOL 68, Modula and Ada.

The problem with the former approach is that we have to introduce some compound statement form if we want

** **Affiliation and contacts as in the original publication***

several statements to be governed by the condition thus

```
if condition then
begin several statements end;
```

The problem with the compound approach is illustrated by the following favourite example.

```
if The_Signal = Clear then  -- Ada
   Open_Gates;
   Start_Train:
end if;

if (The_Signal == 0)          /* C */
{  Open_Gates ();
    Start_Train ();
}
```

Consider what happens if we accidentally add a semicolon at the end of the first line in both cases. In Ada it fails to compile and the error is mechanically detected before it can do any harm.

In C it still compiles, a null statement is controlled by the condition and so the gates open and the train goes whatever the state of the signal. No error at compile time and a nasty runtime error is the result.

Note also how C uses the unnatural == for = having ruined = by using it for assignment. And worse, consider what happens if one of the = is omitted. It then becomes an assignment whose result is used for the condition. So even if the signal were at Danger it is now set Clear as a nasty side effect!

So expression languages have pitfalls as well so perhaps that is another piece of knowledge:

- conditions and assignments should not be mixed.

So ALGOL 68 fails this test as well.

I don't think we know much else, we are not really even sure about a type Integer since we cannot agree about that upper limit. Of course we do seem to have experience of lots of other things and indeed it would be foolish to assert that our experience didn't count for something. But it is all warm glimpses rather than firm knowledge. Certainly OOP (as opposed to OOD) is exploratory hacking at the jungles of the unknown.

Accepting my two (or three) pitiful bits of real knowledge, how do languages stand up to them?

Well FORTRAN, PL/l, COBOL and C/C++ fail the := test. And ALGOL 60, Pascal and C/C++ fail the bracket test. And if we add the expression test then ALGOL 68 and C/C++ fail that. Note, dear reader, that C and C++ of widely used or classic languages are the only languages that completely fail to incorporate the only real knowledge we have. Millions of programmers out there are forced by their management and teachers to use dangerous and outmoded techniques. We need a revolution brothers. Stuff this evolution.

And now to OOP. We feel that abstraction is good stuff. But it mustn't leak. A leaky abstraction with hidden holes is a dangerous beast. But isn't inheritance just one big leak?

You get properties you don't see in the declaration. Thus given

```
type Gender is (Male, Female);
type Person(Sex: Gender) is abstract tagged
  record
    Birth: Date;
  end record;
```

then when you declare

```
type Old_Person is new Person with
  record
    Pension: Money;
  end record;
```

the Old_Person has visible Sex even though it is not mentioned in the declaration. Things are not what they seem.

And writing

```
type Weight is (Light, Medium, Heavy);
type Boxer(W: Weight) is new
    Person(Sex => Male) with ...
```

results in a type Boxer which does not have visible Sex even though it is mentioned in the declaration.

Clearly however, if OOP is to deliver reuse then inheritance is likely in some form. But the design trick is to minimise the surprises associated with inheritance. Ada does this rather better than C++ by its cleaner rules for dispatching and its sharp distinction between a type and a set of types (T and T'Class). And Ada permits multiple inheritance in a controlled manner with no ad-hoc rules.

The confusion between types and classes is a really sad thing about C++. A whole generation of programmers has been badly served by this confusion; an illustration of the sad drift of society into an illiterate and non-numerate mob.

But Ada 95 is an opportunity to rectify this. Here we have a language that passes the three tests and is crisply clear about types, classes and inheritance.

So where are we going. The realisation that software really matters is growing but there is little awareness among those of authority that the language in which it is written really matters. Of course, much else matters as well, but best practice should be used for all aspects of the development of software systems.

Freedom is all the rage. What do we want? Freedom to make errors or freedom from errors? The pendulum of society is swung towards anarchy and the freedom of the individual. Towards the criminal rather than the victim. Towards rights rather than duties. But I believe we have a duty to write our software the best way we can. We should not hanker after some fantasy rights to write flaky programs that have the higher risk of damaging our fellow citizens and the environment.

Let us hope the pendulum will swing back before we sink in a sea of C.

The Ravenscar Tasking Profile for High Integrity Real-Time Programs

Brian Dobbing *
Aonix, 5040 Shoreham Place, San Diego, CA 92122 USA; email: brian@uk.aonix.com.

Abstract

The Ravenscar Profile defines a simple subset of the tasking features of Ada in order to support efficient, high integrity applications that need to be analysed for their timing properties. This paper describes the Profile and gives the motivations for the features it does (and does not) include. An implementation of the Profile is then described in terms of development practice and requirements, run-time characteristics, certification, size, testing, and scheduling analysis. Support tools are discussed as are the means by which the timing characteristics of the run-time can be obtained. The important issue of enforcing the restrictions imposed by the Ravenscar Profile is also addressed.

Introduction

High-integrity systems traditionally do not make use of high-level language features such as Ada tasking. This is despite the fact that such systems are inherently concurrent. Concurrency is viewed as a "systems" issue. It is visible during design and in the construction of the cyclic executive that implements the separate code fragments, but it is not addressed within the software production phases. Notwithstanding this approach, the existence of an extensive range of concurrency features within Ada does allow concurrency to be expressed at the language level with the resulting benefits of having a standard approach that can be analysed and checked by the compiler, and supported by other tools.

The requirement to analyse both the functional and temporal behaviour of high integrity systems imposes a number of restrictions on the concurrency model that can be employed. These restrictions then impact on the language features that are needed to support the model. Typical features of the concurrency model are as follows.

- A fixed number of activities (we shall use the Ada term *task* to denote an independent concurrent activity).
- Invocations. The invocation event can either be temporal (for a time-triggered task) or a signal from either another task or the environment. A high-integrity application may restrict itself to only time-triggered tasks.
- Tasks only interact via the use of shared data. Updates to any shared data must be atomic.

These constraints furnish a model that can be implemented using fixed priority scheduling (either preemptive or non-preemptive) and analysed in a number of ways:

- The functional behaviour of each task can be verified using the techniques appropriate for sequential code (e.g., [1]). Shared data is viewed as just environmental input when analysing a task. Timing analysis can ensure that such data is appropriately initialised and temporally valid.
- Following the assignment of temporal attributes to each task (period, deadline, priority, etc.), the system-wide timing behaviour can be verified using the standard techniques in fixed priority analysis (e.g. [2]).

Tasking Features

The Ada95 language revision has both increased the complexity of the tasking features and provided the means by which subsets (or profiles) of these features can be defined. To all of the Ada83 features (dynamic task creation, rendezvous, abort) has been added protected objects, ATC (asynchronous transfer of control), task attributes, finalization, requeue, dynamic priorities and various low-level synchronization mechanisms. Subsets are facilitated by pragma Restrictions that allows various aspects of the language to be limited in scope or removed from the programmer completely.

Whilst the full language produces an extensive collection of programming aids (e.g. see [3]) from which higher-level abstractions can be constructed, there are a number of motivations for defining restricted models:

- increase efficiency by removing features with high overheads
- reduce non-determinacy for safety-critical applications
- simplify run-time kernel for high-integrity applications
- remove features that lack a formal underpinning
- remove features that inhibit effective timing analysis

Of course, the necessary restrictions are not confined to the tasking model, but this paper only considers concurrency. To implement a restricted concurrency model in Ada

** **Affiliation and contacts as in the original publication***

requires only a small selection of the available tasking features. At the Eighth International Real-Time Ada Workshop (1997) the following profile (called the Ravenscar Profile) was defined for high-integrity, efficient, real-time systems [4].

The Ravenscar Profile

The Ravenscar Profile is defined by the following:

- *Task type and object declarations at the library level* – that is, no hierarchy of tasks, and hence no exit protocols needed from blocks and subprograms.
- *No unchecked deallocation of protected and task objects* – removes the need for dynamic objects.
- *No dynamic allocation of task or protected objects* – removes the need for dynamic objects.
- *Tasks are assumed to be non-terminating* – this is primarily because task termination is generally considered to be an error for a real-time program which is long-running and defines all of its tasks at start-up.
- *Library level Protected objects with no entries* – these provide atomic updates to shared data and can be implemented simply.
- *Library level Protected objects with a single entry* – used for invocation signalling; but removes the overheads of a complicated exit protocol.
- *Barrier consisting of a single Boolean variable* – no side effects are possible and exit protocol becomes simple.
- *Only a single task may queue on an entry* – hence no queue required; this is a static property that can easily be verified, or it can lead to a bounded error at runtime.
- *No requeue* – leads to complicated protocols, significant overheads and is difficult to analyse (both functionally and temporally).
- *No Abort or ATC* – these features leads to the greatest overhead in the run-time system due to the need to protect data structures against asynchronous task actions.
- *No use of the select statement* – non-deterministic behaviour is difficult to analyse, moreover the existence of protected objects has diminished the importance of the select statement to the tasking model.
- *No use of task entries* – not necessary to program systems that can be analysed; it follows that there is no need for the accept statement.
- *"Delay until" statement but no "delay" statement* – the absolute form of delay is the correct one to use for constructing periodic tasks.
- *"Real-Time" package* – to gain access to the real-time clock.
- *No Calendar package* – "Real-Time" package is sufficient.
- *Atomic and Volatile pragmas* – needed to enforce the correct use of shared data.
- *Count attribute (but not within entry barriers)* – can be useful for some algorithms and has low overhead.
- *Ada.Task_Identification* – can be useful for some algorithms and has low overhead, available in reduced form (no Abort_Task or task attribute functions Callable or Terminated)
- *Task discriminants* – can be useful for some algorithms and has low overhead.
- *No user-defined task attributes* – introduces a dynamic feature into the run-time that has complexity and overhead.
- *No use of dynamic priorities* – ensures that the priority assigned at task creation is unchanged during the task's execution, except when the task is executing a protected operation.
- *Protected procedures as interrupt handlers* – required if interrupts are to be handled.

The inclusion of protected entries allows event based scheduling to be used. For many high integrity systems only time-triggered actions are employed, hence such entries and their associated interrupt handlers are not required.

The profile defines dispatching to be *FIFO within priority* with protected objects having *Ceiling Locking*. However it also allows a non-preemptive policy to be defined. Co-operative scheduling (that is, non-preemption between well-defined system calls such as "delay until" or the call of a protected object) can reduce the cost of testing as preemption can only occur at well-defined points in the code. It can also reduce the size of the run-time.

With either dispatching policy, the Ravenscar Profile can be supported by a relatively small run-time. It is reasonable to assume that a purpose-built run-time (supporting only the profile) would be efficient and "certifiable" (i.e. built with the evidence necessary for its use in a certified system). An equivalent run-time for a constrained Ada 83 tasking model has already been used in a certified application [5].

With the profile, each task should be structured as an infinite loop within which is a single invocation event. This is either a call to "delay until" (for a time-triggered task) or a call to a protected entry (for an event-triggered task).

The use of the Ravenscar profile allows timing analysis to be extended from just the prediction of the worst-case behaviour of an activity to an accurate estimate of the worst-case behaviour of the entire system. The computational model embodied by the Ravenscar profile is very simple and straightforward. It does not include, for example, the rendezvous or the abort, and hence does not allow control flow between tasks (other than by the release of a task for execution in the event triggered model). But it does enable interfaces between activities (tasks) to be checked by the compiler.

Preemptive execution, in general, leads to increased schedulability and hence is more efficient in the use of system's resources (e.g. CPU time). As preemption can occur at any time, it is not feasible to test all possible preemption points. Rather, it is necessary for the run-time system (RTS) to guarantee that the functional behaviour of a task will not be affected by interrupts or preemption. For a high integrity application evidence to support this guarantee would need to be provided by the compiler vendor (or RTS supplier). For the Ravenscar profile the RTS will be simple and small.

Not only does the use of Ada increase the effectiveness of verification of the concurrency aspects of the application, it also facilitates a more flexible approach to the system's timing requirements. The commonly used cyclic executive approach imposes strict constraints on the range and granularity of periodic activities. The Ravenscar profile will support any range and a fine level of granularity. So, for example, tasks with periods of 50ms and 64ms can be supported together. Moreover, changes to the timing attributes of activities only require a re-evaluation of the timing analysis. Cyclic executives are hard to maintain and changes can lead to complete reconstruction.

In a control system information may be translated through several stages. Input of sensor data may be scaled, filtered, used in control law calculations, scaled for output and finally output to a transducer. Safety critical standards e.g. DO-178B 6.4.4.2(a) requires that *"the analysis should confirm the data coupling and control coupling between the code components"*. This has been achieved in the past using cyclic executives which pass data between the code components in strict sequence. With the introduction of more sophisticated sensors, and the requirements to build more responsive systems, the data input and output rates and the rates of the computational processes may not be the same. A natural mapping for such systems is to use tasks with event triggers which enable data to be acquired, processed and output to transducers, at rates which are optimal for each processing step. Events provide a direct link between data and the code used in its processing. The Ravenscar profile facilitates the construction of concurrent programs where the code/data coupling is controlled, defined by the language and checked by the compiler (in contrast to facilities offered by run-time kernels defined independently of the language). The analysis to confirm coupling would be performed by code reviews to show that data is only accessed through synchronised or protected constructs.

Finally, note that the inclusion of a small number of event triggered activities does not fundamentally change the structure of the concurrent program or the timing analysis, but it does impose significant problems for the cyclic executive. Polling for 'events' is a common approach in high integrity systems; but if the 'event' is rare and the deadline for dealing with the event is short then the time triggered approach is very resource intensive. The event-triggered approach will work with much less resources.

Code Templates

The profile does not require the application to use any particular coding style for the execution of the tasks, protected objects, and interrupt handlers. However if the application is required to undergo schedulability analysis, certain task templates and coding styles are useful in defining the activities that are to be analyzed. These are described below:

Time-Triggered Task. The task body for a time-triggered task typically has, as its last statement, an outermost infinite loop containing one or more *delay until* statements [RM section 9.6]. (The basic form of a cyclic task has just a delay until statement either at the start or at the end of the statements within the loop.) The model supports only one time type for use as the argument – Ada.Real_Time.Time [RM section D.8] – which maps directly to the underlying system clock for the maximum precision. Note that task termination is a bounded error condition in the Ravenscar profile; hence the loop is infinite. Example:

```
task body Cyclic is
   Next_Period: Ada.Real Time.Time := First_Release;
   Period: Ada.Real_Time.Time_Span :=
      Ada.Real_Time.Milliseconds(50):
   - - other declarations
begin
   - - Initialization code
   loop
         delay until Next_Period;
         - - Periodic response code
         Next_Period := Next_Period + Period;
   end loop;
end Cyclic;
```

Event-Triggered Task. The task body for an event-triggered task typically has, as its last statement, an outermost infinite loop containing as the first statement either a call to an Ada protected entry [RM section 9.5] or a call to wait for the state of an Ada "Suspension Object" [RM section D.10] to become true.

The suspension object is the optimized form for a simple suspend/resume operation. The protected entry is used when extra operations are required:

- Data can be transferred from signaller to waiter atomically (i.e., without risk of race condition) by use of parameters of the protected operations and extra protected data.
- Additional code can be executed atomically as part of signalling by use of the bodies of the protected operations.

Example:

```
protected Event is
   entry        Wait    (D: out Data);
   procedure  Signal (D: in Data);
private
```

```
    Current    : Data; - - Event data declaration
    Signalled : Boolean := False;
end Event;

protected body Event is
    entry Wait (D : out Data) when Signalled is
    begin
        D   := Current;
        Signalled := False;
    end Wait:
    procedure Signal (D : in Data) is
    begin
        Current : = D;
        Signalled := True:
    end Signal;
end Event;

task body Sporadic is
    My_Data : Data;
    - - other declarations
begin
    - - Initialization code
    loop
        Event.Wait (D => My_Data);
        - - Response code processing My_Data
    end loop;
end Sporadic;
```

Interrupt Handlers. The code of an interrupt handler will often be used to trigger a response in an event-triggered task. This is because the code in the handler itself executes at the hardware interrupt and so typically the major part of the processing of the response to the interrupt is moved into the task, which executes at a software priority level with interrupts fully enabled. The interrupt handler typically will store any interrupt data in its protected object and then releases the waiting event-triggered task by changing the state of the protected data Boolean used as the entry barrier in the same protected object, as shown in the example protected object Event.

Shared Resource Protected Object. A protected object used to ensure mutually exclusive access to a shared resource, such as global data, typically contains only protected subprograms as operations, i.e., no protected entries. Protected entries are used for task synchronization purposes. A protected procedure is used when the internal state of the protected data must be altered, and a protected function is used for information retrieval from the protected data when the data remains unchanged.

Example:

```
protected Shared_Data is
    function Get return Data;
    procedure Put (D : in Data);
private
    Current : Data; - - Protected shared data declaration
end Shared_Data;

protected body Shared_Data is
    function Get return Data is
    begin
        return Current;
    end Get;

    procedure Put (D in Data) is
    begin
        Current := D;
    end Put;
end Shared_Data;
```

Implementing the Ravenscar Profile

Ada compiler vendor Aonix has undertaken the development of an Ada95 compilation system which implements the Ravenscar profile, known as *Raven* [6], hosted on Windows NT and Spare Solaris, and targeting the PowerPC, MC680xO and Intel range of processors. This section describes some of the key elements of this implementation.

Development Practices

The principle goal of the implementation was to develop a runtime system for Ada95 restricted as per the Ravenscar profile, which was suitable for inclusion in:

- A safety-critical application requiring formal certification
- A high-integrity system requiring functional determinism and reliability
- A concurrent real-time system with timing deadlines requiring temporal determinism, e.g. schedulability analysis
- A real-time system with execution time constraints requiring high performance
- A real-time system with memory constraints requiring small and deterministic memory usage

Consequently, a rigorous set of development practices was enforced based on the traditional software development model, including:

- Documentation of the software requirements
- Definition and documentation of the design to meet these requirements, including traceability
- Formal design reviews
- Formal code walk-throughs of the runtime implementation
- Definition and documentation of the runtime tests to verify correct implementation of the design
- Documentation of the formal verification test results
- Capture of all significant items within a configuration management system

Requirements

The software requirements include the following elements:

- The runtime design shall support both the preemptive and non-preemptive implementations of the Ravenscar profile.
- The runtime design shall optimise a purely sequential (non-tasking) program by not including any runtime overhead for tasking.
- The design shall structure the runtime such that a library of additional runtime Ada packages which have not undergone formal certification can be supplied as a stand-alone "extras", for applications which require the extra functionality but not the rigors of certification.
- The runtime algorithms shall be coded such that the worst case execution time is deterministic and as short as possible.
- The runtime algorithms shall be coded such that the average case execution time is as short as possible.
- The runtime algorithms shall be coded so as to minimise the use of global data, and so as not to acquire memory dynamically. (The total global memory requirement of the runtime system shall be small and deterministic.)
- The runtime algorithms shall be coded so as to conform to the certification coding standards.
- The runtime algorithms shall be coded as to conform to Ravenscar profile plus sequential code restrictions.
- A coverage analysis tool shall be provided for certification purposes.
- A schedulability analyser shall be provided which supports standard algorithms used in fixed-priority timing analysis.
- Enforcement of the Ravenscar profile, plus other restrictions on sequential constructs, shall be performed at compile-time wherever possible. (This eliminates runtime code to perform the checks, and the risk of runtime exceptions being raised in the event of check failure.)
- The compilation system tools shall be verified using the Ada Compiler Validation Capability (ACVC) test suite, by coupling the tools to an alternate runtime system for the same target processor family, and which supports full Ada95. Runtime algorithms, which are common to the Raven runtime and the alternate runtime, shall also be verified in this way.
- The runtime kernel shall be verified using the verification tests written to validate the correct implementation of the requirements.

Design Considerations

Enforcing the Restrictions: The Ravenscar profile restrictions apply only to the concurrency model. It was therefore necessary first to define the additional restrictions that apply to sequential code. These are not defined in this paper, but in essence they follow the same goals of ensuring deterministic execution, simplifying the runtime support, and eliminating constructs with high overhead.

The requirement of enforcing as many restrictions as possible at compile time was met using the Ada95 pragma Restrictions [RM section 13.12]. A few of the needed restrictions were already defined using standard restriction identifiers in RM sections D.7 and H.4. However, many of the restrictions required new (implementation-defined) identifiers. These identifiers have been submitted to the ISO Annex H Rapporteur Group that has endorsed the Ravenscar Profile in its recommendations on the use of Ada in High-Integrity Systems. Within the Raven implementation, the set of needed restriction pragmas is supplied in source form to facilitate compilation into the Ada program library as configuration pragmas.

A compiler that enforces a subset to satisfy safety requirements needs to be carefully constructed. The compilation algorithms should not be changed to implement a particular subset, thereby preserving the value of its maturity and testing, including ACVC validation. This is an important means of raising the trust in the correctness of the toolset being used. Instead, the changes to generate the subset compiler are confined to reporting on violations of the subset in response to the presence of pragma Restrictions.

Two of the Ravenscar profile restrictions are enforced at runtime in the Raven product:

- Violation of No_Task_Termination is classed as a bounded error, which is defined to cause permanent suspension of the task. A mechanism to invoke a user-written handler for this situation is provided, which gives a hook for the application to apply remedial action.
- Violation of Max_Entry_Queue_Depth=1 is a runtime check since the Raven implementation has chosen not to restrict each protected entry to having only one *statically determinable* calling task, in keeping with the corresponding model which Ada95 uses for Suspension Objects [RM section D.10 (10)]. Consequently violation of this restriction results in Program_Error exception being raised.

Runtime system code which processes bounded error conditions or raises exceptions when a restriction is violated, known as deactivated code (not dead code), is not excluded from certification considerations. DO-178B states [7] that the "*software planning process should describe how the deactivated code will be defined, verified and handled to achieve system safety objectives*." [DO-178B section 4.2(h)]. Coverage testing of this deactivated code is also required by DO-178B: "*... additional test cases and test procedures (should be) developed to satisfy the required coverage objectives.*" [DO-178B section 6.4.4.3(h)]. Thus the level of trust of this error handling code is the same as that of the remainder of the runtime system.

Compilation Unit Closures: The requirements for there to be no runtime overhead due to tasking in a purely sequential (non-tasking) program, and that a non-certifiable library of packages be available stand-alone, providing Ada features beyond the basic kernel functionality, are met using coding conventions regarding closures of library units, as regulated by the use of Ada context ('with') clauses.

Three runtime unit closures are defined: for the sequential program kernel, for the tasking program and for the 'extensions' packages (which are not certifiable). The coding standards are such that the sequential kernel units are not allowed to 'with' tasking kernel units, and neither the sequential nor the tasking kernel units are allowed to 'with' 'extensions' units. Thus the separation of concerns (sequential versus tasking and certifiable versus not-certifiable) is enforced by the compiler using Ada semantics.

Other Runtime Constraints: The principle requirements governing the style of coding to be used for the runtime system are highly compatible and complementary, leading to algorithms which are small, easy to understand, and functionally and deterministic, coupled with use of simple static data structures.

Certifiability: The requirements for certifiability impinge on the source code by means of specifying fixed format header comments for compilation units and all subprograms. The information in these headers includes:

- Overview of purpose or functionality
- Requirement(s) which are met
- Detailed definition of global data / parameter usage
- Detailed definition of algorithm

This description is checked against the actual code during walk-through audits, and is used to verify that the implementation conforms to the design, and that the design fully meets the requirements.

Performance: Several techniques are used to improve the performance of the runtime. Simple and very short runtime subprograms can be defined as having calling convention *Intrinsic* [RM section 6.3.1] which means that their code is built into the compiler and is used directly in place of the call.

Typically this is used for immutable code sequences such as arithmetic and relational operators for types such as Time and Time_Span in package Ada.Real_Time [RM section D.8] and for highly time critical simple operations such as getting the identity of the currently-executing task.

Other short subprograms can be defined as being *inlined* [RM section 6.3.2] which gives similar performance gain by avoiding the procedure call and return overhead, but without having to actually build the generated assembler code into the compiler code generator.

In addition, since the runtime code itself must abide by the restricted Ada subset, this automatically excludes use of non-deterministic and dynamic constructs, plus those with high execution overhead or code size. Thus the code is written using simple Ada constructs which translate to equivalently simple assembler code, making it fast to execute, easy to verify, readable and maintainable.

Early indications of the performance of the runtime system are very encouraging and are listed in Table 1.

PIWG T test (Microseconds)	Rendezvous (VxWorks)	Protected Object (VxWorks)	Protected Object (Ravenscar)
T000001	67.54	8.24	1.8
T000002	96.03	8.48	1.2
T000003	84.31	8.26	1.5
T000004	113.77	8.30	1.2
T000005	107.81	8.30	1.1
T000006	121.87	8.17	1.1
T000007	97.49	8.20	1.8
T000008	245.44	16.32	1.2

Table 1 - Runtime Performance on Ultra 604133MHz

Worst Case Execution Time: In order to perform accurate schedulability analysis, it is necessary to input the runtime execution overhead (see [8]). For hard real-time systems in which the failure to meet a hard timing deadline is catastrophic to the entire system, worst case execution times are generally used in the computations. The user can generally either analyse the Ada code [9] or measure the worst case time for application code using tests that exercise the various code paths, but for the runtime system operations, the user has no direct way of knowing which scenario will produce the worst case time, unless the runtime source code is available and also documentation to describe the criteria which determine the execution path at each decision point.

Thus, for every runtime operation with variable execution time, or whose operation can include a voluntary context switch, the vendor must provide metrics which typically define the worst case execution time either as an absolute number of clock cycles or as a formula based on application-specific data (e.g. number of tasks). For the runtime tasking kernel implementing the Ravenscar profile, this set of metrics will include:

- Entry and exit times for protected operations, including entry calls and barrier evaluation
- Entry and exit times for processing of the delay until statement
- Timer interrupt and user interrupt overheads
- Rescheduling times, such as the time to select a new task to run and the time to perform a context switch

Clearly, this imposes strict constraints on the algorithms used to implement these operations such that their worst case execution time is not overly excessive. For example, use of a linear search proportional to the maximum number of tasks in the program would be unacceptable for a program with a large number of tasks. So, the runtime contains optimisations to minimise critical worst case timings.

Runtime Size: The runtime was designed and coded to minimise the size of both the code and the data. For example, an important optimisation in the Ada pre-linker tool (the "binder") is elimination of uncalled subprograms from the executable image. But this optimisation is only fully effective if the code is structured in a very modular way. For example, the runtime treatment of user-defined interrupt handlers as protected procedures should not be included in the image if interrupts are not used by the program. A more extreme example of this is the requirement that no code or data which is specific to the tasking kernel should be included in the image if the program does not use tasking.

In addition to this, the coding of the runtime data and algorithms was carefully crafted to optimise on speed and space, taking advantage of the various optimisations supported by the compiler.

Regarding data usage, the runtime does not make any use of dynamically acquired memory, which is also a restriction on the sequential code of the application, thereby eliminating the need to support a heap with its associated non-determinism during allocation. The global data used is as small as possible, exploiting packing of data except where poor-quality code would be generated to access it. The data is packaged so that it is eliminated if the feature that it supports is not used (e.g. the interrupt handling table is eliminated when there are no interrupts in the program). The major component of the runtime data is the stack and Task Control Block (TCB) which is required for each task's execution. Each application program is required to declare the memory areas to be used for the stacks and TCBs in the Board Support Package. This provides a simple interface to tune the stack sizes to the worst case values, whilst also giving full application-level determinism on the amount of storage which is reserved for this purpose.

Early indications of the size of the runtime system are very encouraging and are listed in Table 2.

(K-bytes)	Code	Data	Stack
Null program	3.7	0.45	0.81
Hello World	4.2	0.52	0.86
Minimal Tasking	12.1	1.10	1.80

Table 2 -Application Program Sizes (Power PC)

Additional Supporting Tools

The additional tools that have been included in the implementation to support certification and schedulability analysis include:

- Condition code and Coverage Analysis tool
- Schedulability Analyser and Scheduler Simulation tool

Coverage Analysis (AdaCover): Under the DO-178B guidelines [7], it is necessary to perform coverage analysis to show that all the object code (both the application. program part and the Ada runtime system) has been executed, including all possible outcomes of conditions, by the verification tests. The entire runtime system is subjected to coverage analysis as of its auditing process. For the user application code, the tool AdaCover is provided to assist in formal certification.

AdaCover is in two logical parts:

- A target-resident monitor which records the execution of every instruction in the program, including the results of every decision point.
- A host-resident tool which annotates the compiler-generated assembly code listings with the results of stage 1, thereby providing the user with a report of coverage at either the object code or source code level, for the set of executed verification tests.

Schedulability Analysis (PerfoRMAx): The PerfoRMAx tool embodies classic schedulability analyser and scheduler simulation functionality. Given a definition of the actions performed by the tasks in the application in terms of their priority, execution time, period and interaction with shared resources, plus certain runtime system overhead times, the tool performs analysis of the schedulability of the task set based on a user-selectable scheduling theory, for example Rate Monotonic Analysis (RMA)[2].

The tool is also able to provide a graphical view of the processor load based on a static simulation of the scheduling of the tasks by the runtime system, thereby giving clear indication of potential regions of unschedulability. If such regions exist, the tool outputs messages highlighting the cause of the unschedulability together with suggestions for corrective action.

Testing

The testing activity is split into two components:

- Use of the ACVC test suite to verify the validation status of the compiler, binder and code generator support routines
- Development of a specific test suite to certification level for the Raven runtime.

ACVC Testing: Since a substantial number of ACVC tests violate Ravenscar profile restrictions, particularly relating to the tasking tests, it is not possible under current rules to validate such a subset. However by use of the same compilation system tools linked to a full Ada95 runtime system for the same target processor family, it is possible to run the full ACVC suite, thereby validating the correctness of the compiler, binder and common code generator support routines (e.g. block move). The validated compiler contains all the processing to treat pragma Restrictions, but since the tests do not include these pragmas in the source code, no enforcement of the subset is performed and hence all the tests can execute.

Certification Tests: A test suite has been created to verify the correctness of the kernel runtime sub-programs, thereby complementing the ACVC testing (which was not able to test these), whilst also ensuring the level of reliability specified by the requirements.

Each test contains a header in the source code that includes:

- Identification of the requirement to be tested
- Identification of the runtime module under test
- Test description
- Test case definition, including inputs and expected results

The results of executing the tests against each baseline development of the runtime system are documented.

Packaging

When the Raven product is purchased, an option is available to purchase separately all the material required for formal certification. This option includes:

- Full runtime source code
- Full development documentation which is relevant to certification
- Full test pack, including sources, scripts and documentation, so that the tests can be re-executed on the runtime code during formal certification of an application.

Conclusion

This paper has described the Ravenscar profile, a subset of Ada95 tasking intended to model concurrency in safety-critical, high-integrity, and general real-time systems. The use of a powerful, structured and highly checked language such as Ada is vitally important in all market sectors demanding high reliability and efficiency.

The paper has also described a commercial-off-the-shelf implementation of the profile for the PowerPC, MC680x0 and Intel processor families which has proved the feasibility of developing production-quality tool support and a certification-quality runtime system for the Ravenscar profile.

On-going work within the International Standards Organisation Working Group 9 exists to incorporate the profile concepts within the recommendations on the use of Ada in high integrity systems.

References

[1] High Integrity Ada, The SPARK Examiner Approach, J. Barnes, Addison Wesley Longman Ltd (1997)

[2] A Practioner's Handbook for Real-Time Analysis: A Guide to Rate Monotonic Analysis for Real-Time Systems, M.H.Klein et al, Kluwer Academic Publishers (1993)

[3] Concurrency in Ada, A.Burns and A.J.Wellings, Cambridge University Press (1995)

[4] Proceedings of the 8th International Real-Time Ada Workshop: Tasking Profiles, ACM Ada Letters (September 1997)

[5] T-SMART – Task-Safe Minimal Ada Real-time Toolset, B. Dobbing and M. Richard-Foy, in Proceedings of the 8th International Real-Time Ada Workshop, pages 45-50, ACM Ada Letters (September 1997)

[6] ObjectAda/Raven Compilation System for PowerPC, Aonix (1998)

[7] Software Considerations in Airborne Systems and Equipment Certification, RTCA/DO-178B/ED-12B, RTCA Inc (December 1992)

[8] Engineering and Analysis of Fixed Priority Schedulers, D.Katcher et a1, In IEEE Trans. Software Engineering 19 (1993)

[9] Combining Static Worst-Case Timing Analysis and Program Proof, R.Chapman, A Burns, AJ. Wellings. In Real-Time Systems 11(2):145-171 (September 1996)

[RM] Ada95 Reference Manual, ANSI/ISO/IEC-8652: 1995, Intermetrics Inc. (January 1995)

The SPARK way to Correctness is Via Abstraction

John Barnes *

11 Albert Road, Caversham, Reading RG4 7AN, United Kingdom; tel: +44 118 947 4125

Abstract

This paper gives a short introduction to the SPARK language and illustrates how the use of abstraction leads towards correctness.

Keywords: Abstraction, Spark, Ada

Introduction

Abstraction is a key concept in the design of many systems whether they be made of intangible software or real hard stuff such as an automobile. A good system will be such that the various components interact through well-defined interfaces in an appropriate manner. This should eliminate unwanted interactions which might occur if the interfaces are not properly defined. The brake pedal of your car should not change the volume of the radio and so on. This desirable state can be achieved by ensuring that interactions only occur via defined interfaces and moreover that the functionality of the components are completely and correctly specified by the interface definitions (the whole truth and nothing but the truth).

Ada provides interfaces through specifications – typically package specifications containing subprogram specifications. However, these subprogram specifications do not provide a full definition of the subprograms. All they provide is enough information to enable the compiler to construct calls of the subprograms but say little if anything about what the subprograms might actually do. Although the Ada approach enables information hiding to be achieved and good component specifications to be written, and indeed encourages these through its style, nevertheless it does not ensure correctness and completeness.

SPARK enables Ada specifications to be strengthened by providing more information about interfaces and the behaviour of components. This extra information can be provided at various levels. At the simplest level it ensures that a component can only interact with certain objects but need say nothing about what it does to them; at the highest level it provides a complete definition of what it does to the objects. At the simplest level it thus prevents unexpected side effects whereas at the highest level it can lead to complete proofs of correctness.

SPARK should be looked upon as a language in its own right. In practical terms, it is a subset of Ada with additional information provided through annotations which take the form of Ada comments. Programs are therefore compiled with a normal Ada compiler and in addition are examined with independent SPARK tools which also analyse the annotations.

It is often felt that formal tools are hard to use and require a great deal of effort. One of the advantages of SPARK is its flexibility. It can be used for formal proof but a great deal of benefit can be obtained by its use at the simplest level which requires little effort. This paper outlines some important features of SPARK using a number of examples.

Abstraction

The first part of this paper introduces the basic ideas of abstraction and refinement.

A simple example

We start by considering a very simple example which shows how the SPARK annotations increase the level of information concerning abstraction. Consider the information given by the following Ada procedure specification

```
procedure Add(X: in Integer);
```

Frankly, it tells us very little. It just says that there is a procedure called Add and that it takes a single parameter of type Integer whose formal name is X. But it says nothing about what the procedure does. It might do anything at all. It certainly doesn't have to add anything nor does it have to use the value of X. It could for example subtract two unrelated global variables and print the result to some file. But now consider what happens when we add the lowest level of SPARK annotation. The specification might become

```
procedure Add(X: in Integer);
--# global in out Total;
```

This states that the only global variable that the procedure can access is that called Total. Moreover it has mode information similar to that of parameters; indeed a global variable can be looked upon as a parameter in which the actual is always the same. The SPARK rules also say more about the modes. Whereas in Ada the modes provide permission to read or update as appropriate, in SPARK such reading or updating is mandatory (SPARK generally abhors unused entities). So the specification tells us that the initial value of Total must be used (**in**) and that a new value will be produced (**out**) and also that the parameter X (**in**) must be used.

So now we know rather a lot. We know that a call of Add will produce a new value of Total and that it will use the initial value of Total and the value of X. We also know that Add cannot affect anything else. It certainly cannot print anything nor have any other malevolent side effect.

The next level of annotation gives the detailed dependency relations so that the specification becomes

** Affiliation and contacts as in the original publication*

```
procedure Add(X: in Integer);
--# global in out Total;
--# derives Total from Total, X;
```

In this particularly simple example, this adds no further information. We already knew that we had to use X and the initial value of Total and produce a new value of Total and this is precisely what this derives annotation says.

Finally we can add the third level of annotation which concerns proof and obtain

```
procedure Add(X: in Integer);
--# global in out Total;
--# derives Total from Total, X;
--# post Total = Total~ + X;
```

The postcondition explicitly says that the final value of Total is the result of adding its initial value (distinguished by ~) to the value of X. So now the specification is complete.

It is important to emphasize that these annotations are part of the procedure specification. (In the case of distinct specification and body, the annotations are not repeated in the body; if there is no distinct specification then they occur in the body before the reserved word **is**.) The annotations separate the interaction between the caller and the specification from that between the specification and the implementation. Hence the Examiner (the main SPARK tool) carries out two sets of checks; it checks that the annotations are consistent with the procedure body and it also checks that the annotations are consistent with each call of the procedure.

Thus when we come to implement Add, if we access a global other than Total or use Total or X in a way inconsistent with the mode information then the SPARK Examiner will produce appropriate error messages.

Generally, the higher levels of annotation enable the Examiner to carry out a more searching analysis.

State

The idea of state is vitally important. Programs do things by changing the state of objects in a general sense. In Ada, state is typically held in the form of variables in packages. A simple example is provided by a random number generator in which the state of the sequence is held in a variable hidden in a package body. Consider

```
package Random_Numbers
--# own Seed;
--# initializes Seed;
is
  procedure Random(X: out Float);
  --# global in out Seed;
  --# derives X, Seed from Seed;
end Random_Numbers;

package body Random_Numbers is
  Seed: Integer;
  Seed_Max: constant Integer := ... ;

  procedure Random(X: out Float) is
  begin
    Seed := ... ;
    X := Float(Seed) / Float(Seed_Max);
  end Random;

begin                   -- initialization part
  Seed := 12345;
end Random_Numbers;
```

This example shows the package body containing the declaration of a variable Seed and the body of the subprogram Random. Each call of Random updates the value of Seed using some pseudo-random algorithm and then updates X by dividing by the constant Seed_Max. Each successive value of Seed depends upon the previous value and is preserved between calls of Random. The variable Seed is initialized in the initialization part of the package body.

This example also illustrates a number of other annotations. The variable Seed has to be mentioned in both an own annotation and an initialization annotation of the package specification. The own annotation makes it visible to other annotations and the initializes annotation indicates that it must be initialized by the elaboration of the package. The procedure Random contains a global annotation for Seed as well as a derives annotation.

The initializes annotation can also be satisfied by initializing Seed in its declaration. An alternative approach might be to declare some procedure Start in the package Random_Numbers (to be called from outside) whose purpose is to assign a first value to Seed. In this case an initializes annotation would not be required but the Examiner will complain if flow analysis reveals that Random is being called before Start.

It is important to observe that from the Ada point of view the variable Seed is not declared until the body and is thus not known to the compiler at the point of the specification of the subprogram Random. However, Seed is a global variable of Random from the point of view of SPARK and thus must be mentioned in the annotation for Random so that flow through Random may be tracked; the own annotation ensures that Seed is known to the Examiner at the specification of Random.

The derives annotation shows explicitly that each call of Random produces a number X derived from Seed and also modifies Seed. As mentioned earlier this annotation is optional.

The variable Seed is protected from manipulation by users of the procedure Random by being declared within the body of the package although it is visible in the annotations in the specification. It could be argued that making the existence of Seed known to the user is a violation of abstraction. However, we certainly ought to know that the procedure Random does something to some state external to itself otherwise we could deduce that each call of Random would inevitably produce the same value each time it is called. On the other hand we don't need to know exactly what Seed is and indeed in this example the external view reveals no details.

Abstract state machines

The random number package is a very simple example of an abstract state machine. In general an abstract state machine is an entity, which has well defined states plus a set of operations, which cause state transitions; properties of the state can be observed by calling appropriate functions.

An abstract state machine is typically represented in Ada by a package, with variables which record its state declared in its body. Procedures that act on the machine and functions that observe its state are specified in the visible part of the package specification. All other details are hidden in the package body.

The following shows the full details of a single stack treated as an abstract state machine with the state initialized automatically on elaboration.

```
package The_Stack
--# own S, Pointer;
--# initializes Pointer;
is
  procedure Push(X: in Integer);
  --# global in out S, Pointer;
  --# derives S from S, Pointer, X &
  --#         Pointer from Pointer;

  procedure Pop(X: out Integer);
  --# global in S; in out Pointer;
  --# derives Pointer from Pointer &
  --#         X from S, Pointer;
end The_Stack;

package body The_Stack is
  Stack_Size: constant := 100;
  type Pointer_Range is range 0 .. Stack_Size;
  subtype Index_Range is
               Pointer_Range range 1 .. Stack_Size;
  type Vector is array (Index_Range) of Integer;
  S: Vector;
  Pointer: Pointer_Range;

  procedure Push(X: in Integer) is
  begin
    Pointer := Pointer + 1;
    S(Pointer) := X;
  end Push;
  procedure Pop(X: out Integer) is
  begin
    X := S(Pointer);
    Pointer := Pointer - 1;
  end Pop;
begin
  Pointer := 0;
end The_Stack;
```

The stack state variables S and Pointer are declared in the body of the package and Pointer is initialized. These internal variables are not directly accessible to users of the stack object. However, their existence and the existence of the initialization of Pointer are made visible to the Examiner for the purpose of analysis by the **own** and **initializes** annotations in the package specification just as the variable Seed of the package Random was made visible.

However, the above technique is not satisfactory since we have made visible considerable detail of the internal representation of the state of the machine, namely the existence of the individual variables S and Pointer. If at some later stage we need to change the implementation then there is a high risk that the specification will need to be changed because of the SPARK rules even though it would not need to be changed by the Ada rules. This would in turn give rise to tiresome dependencies since it would require all the calls to be reexamined and recompiled.

(A minor problem with the package as written is that when we come to use it we will get messages saying that S is being used before it is given a value. Of course we know that the dynamic behaviour is such that the initialization of S is unnecessary but the Examiner is not aware of this. Perhaps the best solution is simply to initialize S as well.)

Refinement

The problems of unnecessary dependencies can be overcome by using abstract own variables to provide what is known as refinement. An abstract own variable does not correspond to a concrete Ada variable at all but instead represents a set of variables used in the implementation.

As a consequence, an abstract own variable occurs in two annotations, the own variable clause in the package specification and then also in a refinement definition in the body giving the set onto which it is mapped.

The stack example could then be rewritten as

```
package The_Stack
--# own State;              -- abstract variable
--# initializes State;
is
  procedure Push(X: in Integer);
  --# global in out State;
  --# derives State from State, X;

  procedure Pop(X: out Integer);
  --# global in out State;
  --# derives State, X from State;
end The_Stack;

package body The_Stack
--# own State is S, Pointer;     -- refinement definition
is
  Stack_Size: constant := 100;
  type Pointer_Range is range 0 .. Stack_Size;
  subtype Index_Range is
               Pointer_Range range 1 .. Stack_Size;
  type Vector is array (Index_Range) of Integer;
  S: Vector;
  Pointer: Pointer_Range;
```

```
    procedure Push(X: in Integer)
    --# global in out S, Pointer;
    --# derives S from S, Pointer, X &
    --#         Pointer from Pointer;
    is
    begin
      Pointer := Pointer + 1;
      S(Pointer) := X;
    end Push;

    procedure Pop(X: out Integer)
    --# global in S; in out Pointer;
    --# derives Pointer from Pointer &
    --#         X from S, Pointer;
    is
    begin
      X := S(Pointer);
      Pointer := Pointer - 1;
    end Pop;

  begin                   -- initialization
    Pointer := 0;
    S := Vector'(Index_Range => 0);
  end The_Stack;
```

This enables the more abstract specification to be linked with the concrete body. The refinement acts as the link and says that the abstract own variable State is implemented by the two concrete variables S and Pointer.

Note moreover that the subprogram bodies have to have a refined version of their global and derives annotations (if provided) written in terms of the concrete variables.

One consequence of the refinement is that both Pointer and S have to be initialized because we have promised that the abstract variable State will be initialized. Of course, as mentioned earlier, we know that the dynamic behaviour is such that the initialization of S is unnecessary and we could omit it in practice and ignore the consequential message from the Examiner.

The various constituents of the refinement must either be variables declared immediately within the package body (such as S and Pointer) or they could be own variables of private child packages or of embedded packages declared immediately within the body. The process of refinement can be repeated since an own variable in the constituent list might itself be an abstract own variable of the child or embedded package.

It is worth summarizing some key points regarding the visibility of state variables of abstract state machines.

- The **own** annotation of an abstract state machine makes the existence of its state visible wherever the machine is visible.
- Annotations of subprograms external to a machine which (indirectly) read or update its state (by executing subprograms of the machine) must indicate that they import or export the machine state.
- Only the existence of the machine state (and its reading or updating) is significant in this context. The details can still be hidden by refinement.

The second point is important and states that annotations have to be explicitly transitive. Thus a procedure that calls Push and Pop also has to be annotated to indicate that it changes the state of the stack.

```
procedure Use_Stack
--# global in out The_Stack.State;
--# derives The_Stack.State from The_Stack.State;
is
begin
  The_Stack.Push( ... );
  ...
  The_Stack.Pop( ... );
  ...
end Use_Stack;
```

Finally note that one abstract state machine could be implemented using another abstract state machine embedded within it. Thus if a machine B is to be embedded in a machine A, this can be done by embedding the package representing B in the body of the package representing A. The state of B can then be represented as an item in the refinement. Alternatively the package representing B could be a private child of the package representing A.

Refinement of course relates to top-down design and provides a natural way of implementing such a design. It is especially important that refinement can be cascaded; this avoids a combinatorial explosion of visible data items which might otherwise occur especially in large programs. The key point is that it makes the existence of state known without giving away the details - the irrelevant detail is kept hidden.

The location of state

It is very important to ensure that state is located sensibly. In order to illustrate this first consider the following simple example

```
procedure Exchange(X, Y: in out Float)
--# derives X from Y &
--#         Y from X;
is
  T: Float;
begin
  T := X;  X := Y;  Y := T;
end Exchange;
```

The parameters X and Y have mode **in out**. This requires them to be both read and updated. The (optional) derives annotation in addition states that the final value of X depends upon the initial value of Y and vice versa. Note that the final value of X does not depend upon the initial value of X.

The scope of program objects should always be as restricted as possible. The rules of SPARK discourage the use of a global variable simply as a 'temporary store'. For example we might try to redefine the procedure Exchange so that the temporary T is global by writing

```
procedure Exchange(X, Y: in out Float)
--# global out T;
--# derives X from Y &
--#          Y from X;
is
begin
  T := X;  X := Y;  Y := T;
end Exchange;
```

But this is illegal because it violates one of several rules of completeness. The one that is violated here is that every variable mentioned in a global definition must be used somewhere in the dependency relation. We have to add T to the derives annotation thus

```
--# derives X from Y &
--#          Y, T from X;
```

and this forces us to admit that we actually change T. Moreover, flow analysis of a call of Exchange will reveal the use of T. Thus a succession of calls such as

```
Exchange(A, B);
Exchange(P, Q);
```

results in the following message from the Examiner

```
               Exchange(A, B);
               ^1
!!! (  1)     Flow Error    : Assignment to T is
              ineffective.
```

This is because the value of T produced by the first call of Exchange is overwritten by the second call without being used. Remember that analysis of the calls is done using only the abstract view presented by the specification and so the internal use of the value of T in the body is not relevant. Note further that this message will be produced even if the optional derives annotation is omitted.

Unnecessary state should thus be avoided. Indeed, the use of unnecessary state as in this example requires annotations for T on the subprogram calling Exchange and so on transitively. The annotations therefore cascade and so the use of unnecessary state is very painful and thereby discouraged.

But some state is necessary and we have seen how refinement may be used to ensure that although the existence of state in an abstract state machine must be made visible, nevertheless the fine details are properly hidden. (We can have our abstraction cake and still eat it!)

There is an interesting analogy between abstraction through refinement and the composition of records out of components. Consider a private type defining a position where the full type reveals the details in terms of *x*- and *y*-coordinates

```
type Position is private;

...

type Position is
  record
    X_Coord, Y_Coord: Float;
  end record;
```

Such a record type is sensible because the two coordinates are logically related; we can then consider a value of the type Position as a single entity which can be manipulated as a whole without knowing the details of its inner construction.

Refinement allows an abstract own variable to provide an external view of a more detailed set of variables within the package. Using the analogy to records, we should only use refinement to group together naturally related items. Thus the refinement of the variable State of the package The_Stack into the variables Pointer and S is appropriate.

Proof

For some applications formal proof is a valuable technique for showing correctness. SPARK has comprehensive facilities for proof including the ability to develop proofs with refinement when there are two views of a state. In order to illustrate this it is necessary to explain some of the basic techniques involved.

The proof process

The general idea is that we state certain hypotheses which we assert are always satisfied when a subprogram is called (the *preconditions*) and we also state the conditions which we want to be satisfied as a result of the call (the *postconditions*). These conditions are given as further annotations in the subprogram specification. We then have to show that the postconditions always follow from the preconditions.

The Examiner processes the text and generates one or more theorems (conjectures really since they might not turn out to be true) which then have to be proved in order to show that the postconditions do indeed always follow from the preconditions. These theorems which are called *verification conditions* are often trivially obvious. If they are not then there are two tools which can be used. These are the Simplifier which carries out routine simplification and the Proof Checker which is an interactive assistant that enables the user to explore the problem and hopefully construct a valid proof.

In order for the proof tools to function correctly, they need to be aware of the various rules which can be used. For the predefined types these are built into the system but other rules can be provided as we shall see in a moment.

As a first example consider once more the procedure Exchange. There is no precondition since it is designed to work no matter what the values of the parameters happen to be. But there is of course a postcondition and so the procedure becomes

```
procedure Exchange(X, Y: in out Float)
--# derives X from Y &
--#          Y from X;
--# post X = Y~ and Y = X~ ;
```

```
is
  T: Float;
begin
  T := X;  X := Y;  Y := T;
end Exchange;
```

Note again the use of the tilde character with in out parameters; the decorated form indicates the initial imported value of the parameter whereas the undecorated form indicates the final exported value.

The verification condition generated by the Examiner for the procedure Exchange is

```
H1:   true .
      ->
C1:   y = y .
C2:   x = x .
```

The notation used is that there are a number of hypotheses (H1, H2, ...) followed by a number of conclusions (C1, C2, ...) which have to be verified using the hypotheses. Note that the conditions are written in a language known as FDL (Functional Definition Language) which has a strong mathematical flavour.

In this example there is no precondition and so effectively no hypotheses (this is represented as the single hypothesis H1 which is true). The two conclusions to be proved are that y = y and x = x which are reasonably self-evident and so it is pretty clear that the procedure Exchange is correct.

If we were stubborn and wanted to be completely confident then we could submit the above verification condition to the Simplifier which would reduce it to simply

```
*** true .     /* all conclusions proved */
```

Verification conditions often appear mysterious and not obviously related to the code; they are produced by a "hoisting process" whereby the postcondition is transformed backwards through the statements in order to arrive at the so-called weakest precondition; this is the condition that must hold at the start in order for the postcondition to hold. We then have to show that the weakest precondition follows from the given precondition. In the verification condition, the hypotheses correspond to the given precondition and the conclusions to be proved correspond to the weakest precondition. However, the details of the hoisting transformations need not concern us in this paper.

Loops

Significant computations usually have loops and these cause complexity in proving correctness. The problems arise because the code of a loop is usually traversed a number of times with different conditions.

The approach taken is to cut a loop so that the various parts can be treated separately. The cut is made by inserting an assert statement which gives conditions that are to be true at that point. The conditions can be thought of as postconditions for the sequence of code arriving at the cutpoint and as preconditions for the sequence going on from the cutpoint.

A simple example is provided by the following integer division algorithm which might be used on a processor without a hardware divide instruction.

```
procedure Divide(M, N: in Integer; Q, R: out Integer)
--# derives Q, R from M, N;
--# pre (M >= 0) and (N > 0);
--# post (M = Q * N + R) and (R < N) and (R >= 0);
is
begin
  Q := 0;
  R := M;
  loop
    --# assert (M = Q * N + R) and (R >= 0);
    exit when R < N;
    Q := Q + 1;
    R := R - N;
  end loop;
end Divide;
```

Each transversal of the loop adds one to the trial quotient and subtracts the divisor N from the corresponding trial remainder until the remainder first becomes less than the divisor. Clearly it only works if both M and N are not negative and also the divisor must not be 0; hence the precondition.

The postcondition has two parts. First the output parameters must have the appropriate mathematical relation implied by the division process and secondly the remainder must be less than the divisor and not negative, so we have

```
--# post (M = Q * N + R) and (R < N) and (R >= 0);
```

The choice of assertion is fairly obvious. As noted above, the final postcondition has two parts, the division relation and the upper and lower bounds on the remainder. All the loop does is keep the division relation true and reduce the remainder until it satisfies the upper bound (as well as keeping the lower bound satisfied). The assertion is simply that the division relation is true and that the remainder satisfies the lower bound; the exit statement is taken when the upper bound is satisfied as well. The initial statements before the loop are designed to ensure that the assertion is true when the loop is first entered.

There are therefore three sections of code to be verified. They are from the start to the beginning of the loop, around the loop, and from the loop to the end. The assert statement acts as the postcondition for the first section and as the precondition for the last section. It also acts as both precondition and postcondition for the loop itself; since it is unchanged by the loop it is often referred to as a loop invariant.

When the Examiner is applied to this subprogram, it produces verification conditions corresponding to the three sections. From the start to the assertion the verification condition is

```
H1:   m >= 0 .
H2:   n > 0 .
        ->
C1:   m = 0 * n + m .
C2:   m >= 0 .
```

Conclusion C2 is trivially obvious since it is just the hypothesis H1. Conclusion C1 is pretty obvious as well.

The verification condition for going around the loop from assertion to assertion is

```
H1:   m = q * n + r .
H2:   r >= 0 .
H3:   not (r < n) .
        ->
C1:   m = (q + 1) * n + (r - n) .
C2:   r - n >= 0 .
```

and that from the assertion to the final end is

```
H1:   m = q * n + r .
H2:   r >= 0 .
H3:   r < n .
        ->
C1:   m = q * n + r .
C2:   r < n .
C3:   r >= 0 .
```

In all cases the Simplifier reduces all the conclusions to true. It is also quite straightforward to show that they are true by hand – although perhaps a little tedious in the case of the loop itself which requires some manipulation. However, such trivial manipulation is prone to error if done by hand and the great advantage of the Simplifier is that it does not make careless mistakes.

Having shown that the verification conditions for the three separate sections of code are true it then follows that the procedure is correct. (To be honest we have only proved that it is partially correct; this means that it is correct provided that it terminates.)

In practice one does not bother to look at the unsimplified conditions and so the process is quite straightforward.

Proof functions

Annotations such as postconditions can be very expressive. Not only can we use the variables of the program but various other notations are also available. We have already noted the use of the tilde character to distinguish initial and final values of in out parameters. The following examples illustrate other possibilities.

```
type Atype is array (Index) of T;

procedure Swap_Elements(I, J: in Index;
                             A: in out Atype);
--# derives A from A, I, J;
--# post A = A~[I => A~(J); J => A~(I)];
```

The postcondition means that the final value of A is the initial value with elements I and J interchanged. Note carefully that it is the initial value of A that is referred to on the right hand side and so there are three uses of the tilde character.

```
function Max(X, Y: Integer) return Integer;
--# return M => (X >= Y -> M = X) and
--#          (Y >= X -> M = Y);
```

This illustrates that functions have return annotations rather than postconditions. The annotation should be read as return M such that if X >= Y then M is X and if Y >= X then M is Y.

```
function Value_Present(A: Atype; X: T) return Boolean;
--# return for some M in Index => (A(M) = X);
```

This function returns true if at least one component of the array has the value X. Remember that Index is the index type of the array type Atype.

```
function Find(A: Atype; X: T) return Index;
--# pre Value_Present(A, X);
--# return Z => (A(Z)) = X) and
--#      (for all M in Index range Index'First .. Z-1 =>
--#          (A(M) /= X));
```

This function returns the index of the first component of the array with the value X. Note the precondition which uses the previous function to ensure that such a value does exist. All Ada functions can be used in annotations in this way with any global variables being added as explicit additional parameters (remember the earlier remark that global variables can be looked upon as parameters that are always the same).

Sometimes, however, the functional nature of the annotation language is not rich enough in which case we can add our own so-called proof functions which do not exist as Ada functions at all.

As an elementary example consider the following implementation of the factorial function

```
--# function Fact(N: Natural) return Natural;

function Factorial(N: Natural) return Natural
--# pre N >= 0;
--# return Fact(N);
is
  Result: Natural := 1;
begin
  for Term in Integer range 1 .. N loop
    Result := Result * Term;
    --# assert Term > 0 and Result = Fact(Term);
  end loop;
  return Result;
end Factorial;
```

The approach we take is to introduce a proof function Fact which we can use in the annotations even though it is not defined in the Ada program text. An interesting observation is that although recursion is not permitted in SPARK because dynamic storage is forbidden, nevertheless proof rules can use recursion in their definition because proof is done offline independently of program execution.

The Examiner is now able to produce verification conditions; it does this without needing to know what the proof function Fact actually means because the process of producing verification conditions simply involves formal substitution.

There are four paths including one from start to finish which bypasses the loop in the case of N being zero. We will look at the verification conditions for just two of them. That from the assertion to the finish is

```
H1:  term > 0 .
H2:  result = fact(term) .
H3:  term = n .
    ->
C1:  result = fact(n) .
```

This is clearly correct by simply substituting from H3 into H2 irrespective of what Fact actually means. That from assertion to assertion is more interesting

```
H1:  term > 0 .
H2:  result = fact(term) .
H3:  not (term = n) .
    ->
C1:  term + 1 > 0 .
C2:  result * (term + 1) = fact(term + 1) .
```

In order to prove this we need a mathematical theorem for the Fact function namely

$$\text{fact}(n) = n \times \text{fact}(n\text{-}1) \qquad n > 0$$

The other two paths need the other obvious mathematical theorem

$$\text{fact}(0) = 1$$

In order to prove the verification conditions using the Proof Checker, it is necessary to give the Checker the rules corresponding to the above theorems. These can be expressed in the following form

```
rule_family fact:
  fact(X) requires [X : i] .

fact(1): fact(N) may_be_replaced_by
                              N * fact(N-1) if [N > 0] .
fact(2): fact(0) may_be_replaced_by 1 .
```

Given such rules the proofs can be entirely mechanized.

The reader might feel that this is all a bit of a cheat. However, the approach is typical of many safety-related mechanisms. Two routes to the solution are provided using entirely different technologies; one uses the Ada program and the other uses the annotations and proof rules. Since they agree we have a high degree of confidence in their correctness.

Proof and refinement

We are now in a position to return to the theme of abstraction and consider how we might add annotations for proof to the stack example.

We saw how we could have two views of the state of the package The_Stack – an external abstract view provided by the abstract variable State and an internal concrete view provided by the two variables S and Pointer. In order to develop proofs we need to map abstract conditions for the external view onto concrete conditions for the internal view. The package might become

```
package The_Stack
--# own State: Stack_Type;        -- abstract variable
--# initializes State;
is
  --# type Stack_Type is abstract;          -- proof type

  --# function Not_Full(S: Stack_Type) return Boolean;
  --# function Not_Empty(S: Stack_Type)
                                      return Boolean;
  --# function Append(S: Stack_Type; X: Integer)
                                   return Stack_Type;

  procedure Push(X: in Integer);
  --# global in out State;
  --# pre Not_Full(State);
  --# post State = Append(State~, X);

  ...  -- similarly Pop

end The_Stack;

package body The_Stack
--# own State is S, Pointer;       -- refinement definition
is
  ... -- etc as before

  procedure Push(X: in Integer)
  --# global in out S, Pointer;
  --# pre Pointer < Stack_Size;
  --# post Pointer = Pointer~ + 1 and
  --#        S = S~[Pointer => X];
  is
  begin
    Pointer := Pointer + 1;
    S(Pointer) := X;
  end Push;

  ...  -- similarly Pop plus initialization

end The_Stack;
```

The above omits the derives annotation partly for simplicity but also to emphasize that derives annotations are not necessary in order to develop proofs although we have shown them in earlier examples for completeness.

The abstract own variable State now includes a type announcement for the proof type Stack_Type. In developing the verification conditions, the Examiner converts this proof type into an FDL record type having two components corresponding to the variables S and Pointer. (Note again the strong analogy between refinement and record composition.)

There are also proof functions Not_Full and Append (with parameters of the proof type) which are used to give the pre- and postconditions for Push. The proof function Not_Empty is required for Pop.

Three verification conditions are generated for Push – one shows that the refined precondition follows from the abstract precondition, one shows that the abstract postcondition follows from the refined postcondition and the other (the usual one) shows that the refined postcondition follows from the refined precondition. The first is

```
H1:  not_full(state) .
H2:  s = fld_s(state) .
H3:  pointer = fld_pointer(state) .
      ->
C1:  pointer < stack_size .
```

The notation should be self-evident, H2 means that the refined variable S corresponds to the field s of the abstract State.

To complete the proofs we need proof rules for the proof functions in terms of the concrete variables such as

```
not_full(S) may_be_replaced_by
                             fld_pointer(S) < stack_size .
```

Given such rules the verification conditions can all be proved.

The stack package might be used by external procedures which themselves have proof annotations in terms of the proof functions. Of course they can only see the external view of the stack and so rules need to be developed in terms of that view. But the rules can themselves be proved using the concrete view.

Design and implementation

One of the goals of this paper is to show that SPARK uses abstraction as a key ingredient in showing correctness. The important thing about abstraction is controlling the level of visibility. We are familiar in Ada with the idea of having more than one view of a type, for example the full view and the partial view of a private type. SPARK allows private types of course but as we have seen extends this idea of views to the representation of state through refinement. We have also seen how proofs may be developed around the two representations.

But it must not be thought that proof is the major goal of SPARK. The real goal is developing correct programs more cheaply and also of course convincing the customer that they are correct within a given budget. Sometimes formal proof is the appropriate tool to being convinced that the program is correct – but for most purposes it would be overkill.

But perhaps the real strength of SPARK is that it encourages good design by revealing the flow of information. For example, suppose we have a package Stuff which contains a procedure Do_It which in turn calls the procedures Push and Pop and thereby manipulates The_Stack. The Ada structure might be

```
package Stuff is
  procedure Do_It;
end Stuff;

with The_Stack;
package body Stuff is
  procedure Do_It is
  begin
    ...
    The_Stack.Push( ... );
    ...
    The_Stack.Pop( ... );
    ...
  end Do_It;
end Stuff;

with Stuff;
procedure Main is
begin
  Stuff.Do_It;
end Main;
```

By just looking at the procedure Main we have absolutely no idea what it does. Even if we look at the specification of Stuff we are none the wiser. We have to look at the body of Stuff to see that it has access to The_Stack. Clearly this is against the spirit of separation of specification and body. The specification ought to tell us what something does whereas the body should simply tell us how it does it. Of course the very fine detail is not always relevant but at least we ought to be clear about what is affected by looking at the specification.

Now consider the same example with the minimal SPARK annotations.

```
--# inherit The_Stack;
package Stuff is
  procedure Do_It;
  --# global in out The_Stack.State;
end Stuff;

with The_Stack;
package body Stuff is
  procedure Do_It is
  begin
    ...
    The_Stack.Push( ... );
    ...
    The_Stack.Pop( ... );
    ...
  end Do_It;
end Stuff;

with Stuff;
--# inherit The_Stack, Stuff;
--# main_program;
procedure Main
--# global in out The_Stack.State;
is
begin
  Stuff.Do_It;
end Main;
```

This introduces two more annotations. One is the inherit clause which is required on the specification of a package in order to give access to other packages. The other is the main program annotation. The global annotations now reveal that the state of the package The_Stack is being manipulated by the procedure Do_It and (transitively) by the main subprogram. The fine details of just what is being done to The_Stack are not revealed and indeed it is probably not necessary to know at this structural level.

But the key point is that the side effect of manipulating the state of the stack is revealed. The annotations encourage good design because a bad design will often have a lot of curious unexpected side effects which are embarrassingly revealed by the annotations. Changing the structure in order to reduce the complexity of annotations will simplify the design by increasing coherence and reducing unnecessary cross-coupling.

Design relates to the specifications of components and their interrelationships whereas implementation relates to their bodies. It is interesting to note that most SPARK annotations apply to specifications and this emphasizes that SPARK is primarily about encouraging good design which then in turn leads to correctness of implementation.

An important issue is scalability, that is the ability to cope with large programs as well as small ones. In this context it is important that refinement can be cascaded. Thus if a component C uses a subcomponent S such as the stack as implementation detail then this fact need not be revealed at the top level. The subcomponent S can be embedded in C or (equivalently) be a private child of C. The state of C can then be refined to include the state of S so that S becomes just an implementation detail.

Note carefully that the most benefit will be obtained from SPARK if it is used as early as possible in the design process. It can weed out poor design before energy is spent on implementation. Of course, SPARK is valuable at the implementation stage as well because it will statically detect many errors that the compiler cannot detect. Indeed, SPARK reaches parts of the program that other tools do not reach.

Levels of use

One of the beauties of SPARK is that it can be used at different levels according to the requirements of the project. The simplest level just requires visibility annotations such as global and own annotations. These alone enable the Examiner to detect a great many errors that cannot be found by the compiler and thus have to be found by the tedious process known as testing often at a later stage in the development process and thus both more expensive to find and to fix.

We know that a key strength of Ada is its strong typing which reveals errors that in a pathetic language such as C have to be found by testing. SPARK extends this capability of Ada by finding even more errors without testing.

At the lowest level of annotation, flow analysis detects many typical errors such as uninitialized variables (those read before being given a value), ineffective parameters (whose value has no effect on the outcome), overwritten values (values that are overwritten before being used), nonterminating loops, aliasing, and so on. In addition many of the errors that can be made in Ada (such as inadvertently using the wrong variable because a later declaration hides it) cannot occur in SPARK because of stricter naming rules.

The introduction of the derives annotation will give more detail of the interactions between components and analysis will then often reveal surprising cross-coupling indicative of poor design or coding errors.

Proof may be appropriate for algorithmic applications. Proof can be applied at several levels as well. This paper has described proof whereby the user is required to add proof annotations. Another option is to check for the absence of runtime errors such as those that arise from violating a bound of an array. Since the Examiner knows about the type model it can generate verification conditions which show the absence of such runtime errors without the user having to supply any annotations at all. Proof can be performed with or without the derives annotations so in fact there are really many levels at which SPARK can be used.

These different levels can be mixed up within a single program. The computational leaves of a system might be subject to proof, the derives annotation might be useful for intermediate subcomponents whereas the outermost part of the system might well have the lowest level of annotation. This is a big strength of SPARK; it can be seen as several tools rolled into one each appropriate to a different part of a project.

Conclusion

Abstraction has been the main theme of this paper. Good abstraction is about revealing relevant detail and hiding irrelevant detail. Plain Ada programs typically do not reveal all the relevant detail. But SPARK with its refinement capability can be used to reveal the detail that matters while keeping the irrelevant detail hidden.

The reader should be aware that this paper has only surveyed some of the capabilities of SPARK. Much has been omitted such as how to interface to external parts of a system. Further details will be found in [1] from which many of the examples given here have been taken and which includes a CD containing demonstration versions of the SPARK tools plus full documentation.

Finally it should be noted that SPARK is well-established and has been successfully used on many projects in a variety of application areas; see for example [2, 3].

References

[1] J. G. P. Barnes (1997), *High Integrity Ada - The SPARK Approach*, Addison-Wesley.

[2] R. C. Chapman (2000), *Industrial Experience with SPARK*, Proceedings of SIGAda 2000.

[3] M. Croxford and J. Sutton (1996), *Breaking Through the V and V Bottleneck*, Proceedings of Ada in Europe Conference 1995, Lecture Notes in Computer Science 1031, Springer-Verlag.

Object-Oriented Programming Enhancements in Ada 200Y

S Tucker Taft *

SofCheck, Inc. 11 Cypress Drive, Burlington, MA 01803; USA.; tel:+1 781 750 8068; email: stt@sofcheck.com

Abstract

This article provides an overview of four proposed amendments to the Ada standard for possible inclusion in the revision planned for late 2005 or early 2006. Together, these four amendments can be seen as "finishing" the job of integrating object-oriented programming features into Ada.

Keywords. Ada, Object-Oriented Programming, Amendment

1 Introduction

A new revision of the Ada programming language standard is being prepared, with a scheduled completion date of late 2005 or early 2006. As part of this revision, the Ada Rapporteur Group (ARG), a part of the ISO Working Group 9 (WG9), is developing proposed amendments to the standard. Several of these amendments relate to object-oriented programming (OOP). This paper will describe some of these amendments, and the background and rationale for their development.

When Ada 95 was being designed, there was still a fair amount of controversy whether object-oriented programming features should be included in the language at all, because of their generally dynamic nature, and because of concern about whether some of their perceived negative aspects (difficult to test and verify, "weaker" typing model, etc.) might outweigh their claimed positive aspects.

Over the past decade, object-oriented programming has become the dominant programming paradigm, so much so that it is now simply assumed, and debates have moved on to other language and methodology issues (e.g. aspect-oriented programming, extreme programming, highly scalable programming, etc.). Two major new object-oriented programming languages have appeared on the scene, Java and C#. And most colleges and high schools are now teaching an object-oriented programming language in their introductory programming courses.

Hence, there is no longer any significant debate whether adding object-oriented programming to Ada 95 was a good idea. The question that remains is whether the object-oriented programming features of Ada 95 are as usable, effective, and understandable as they should be.

2 Differences Between Ada 95 and Other OOP Languages

Before attempting to answer this question, it is useful to first identify what makes Ada 95's object-oriented programming features different from those of most other OOP languages, both in a positive and a negative sense. There are several important differences:

a) Ada 95 makes a significant and explicit distinction between class-wide types and specific types. This distinction implicitly exists in essentially all OOP languages, but there is rarely a way to talk about it in the source language itself. Instead, depending on context, a type or class name in such a language might represent a single type in the hierarchy (what Ada 95 calls a "specific" type), or it might represent a type and all types derived directly or indirectly from it (what Ada 95 calls a "derivation class of types").

Only when dealing with class-wide types in Ada 95 is there any possibility of dynamic binding. In most other OOP languages, dynamic binding is the default, and static binding requires additional effort, or is simply not available. This makes it more likely in such languages that dynamic binding will be used in places where static binding would have been preferred, and would have produced a faster, more verifiable, and more maintainable system.

In Ada 95, because static binding is the default, there will generally be significantly reduced coupling between a derived type and its parent type, allowing the parent operations to be treated more like black boxes. In most other OOP languages, you really need to see the source code for all parent operations to know whether it is safe to inherit any one of them rather than override it in a derived type.

b) Ada 95 has no direct linguistic support for type hierarchies involving multiple inheritance. Although there are several other language features (such as "with" and "use" clauses, generic packages, private extensions, and access discriminants), that allow programmers to solve problems in Ada 95 for which other languages might rely on their linguistic multiple inheritance capabilities, there are still some situations where the lack of linguistic support does restrict the ease of solving an important problem.

c) Except for synchronizing operations (such as a task entry call or a protected operation), all operands to an operation in Ada 95 are treated symmetrically in the syntax. That is,

**** Affiliation and contacts as in the original publication***

they are all passed as parameters "inside" the parentheses, independent of whether the operand might control dynamic binding.

This symmetry makes object-oriented abstract data types be a natural generalization of "normal" abstract data types, and makes user-defined binary operators work in a natural way with such types, without any special treatment. The controlling operand of a binary operator could be the right operand or the left operand, depending on what is appropriate. The controlling "operand" can even be provided by context, in the case of a call on a parameterless function like "Empty_Set" which will result in the invocation of the "appropriate" overriding of Empty_Set, depending on the underlying run-time "tag" of the "receiver" of the result of the call.

Unfortunately, this symmetric approach can result in extra verbiage and possible confusion when used with a multi-package type hierarchy. Some "operations" in such a hierarchy might be so-called "class-wide" operations, which are generally declared in the package where the root type of the hierarchy is declared, while others will be "dispatching" operations which are inherited down the hierarchy, and are implicitly declared within each package where a derived type is declared. To call an operation, one has to either have "use" clauses for all packages where it might have been declared, or determine the correct package and put a prefix on the operation name that identifies the relevant package. Although this does not at first glance seem onerous, when working with relatively large type hierarchies, always identifying the package or "use"ing all the relevant packages can make the code less rather than more readable.

With OOP languages that use the "asymmetric" approach, where the (one and only) controlling operand precedes the name of the operation, and the other operands appear inside the parentheses, there is rarely a need to identify the module where an operation is declared, since it is determined by the type of the controlling operand. In C++, the module name is used generally only when overriding the default dynamic binding, and requesting static binding to an operation in a particular class/namespace.

There are some languages, in particular Modula-3, which allow either notation to be used, with the asymmetric "prefix" notation being a short-hand (syntactic "sugaring") for the symmetric notation.

d) Ada separates declaration from implementation, and requires that all types and operations be declared before they are referenced. In some OOP languages, in particular Eiffel and Java, declaration and implementation are not separated in the definition of a class. Furthermore, in these languages, in part because all objects are referenced via pointers and hence are of known "size," there is no need to declare a class before it is referenced.

Because Ada requires declaration before reference, extra work is required to create collections of types that are mutually dependent. In general, an incomplete type declaration is required to allow for such cyclic type structures. However, an incomplete type must be completed within the same package in which it is declared. This precludes such cyclic type structures from crossing multiple packages, and tends to lead to larger-than-ideal packages simply to accommodate such a cycle. The child library unit feature was added to Ada 95 in part to allow packages to remain smaller, with hierarchies (subsystems) of packages being used to represent large multi-type abstractions.

C++ retains the separation between declaration and implementation, while allowing cyclic type structures to cross multiple "namespaces." This is possible because namespaces may be defined in several separate textual pieces, and an incomplete type declaration in C++ may be in one piece of the namespace, while a separate piece contains the full type declaration. In Ada, packages have only two textually separable pieces, namely the package declaration ("spec") and the package implementation ("body"). But putting a full type declaration in the package body is not a solution to the multi-package cyclic type structure problem, because the declarations within the package body are not visible outside the package. By contrast, all the "pieces" of a C++ namespace can contain "visible" declarations.

e) Ada 95 supports 3 levels of visibility for operations and components of a type: fully public, visible to child units, and visible only within the defining package. Most other OOP languages provide special visibility of operations to derived types (subclasses). In C++ and Java this is called "protected" visibility.

An important advantage of the Ada 95 approach to "partial" visibility is that it is provided only to modules whose position within the naming hierarchy implies their special visibility. This creates a strong boundary around the set of units that might be affected by changes to partially visible operations or components. In most other OOP languages, this special visibility is unrelated to the module structure, and a derived type/subclass which might be affected by changes to partially visible operations or components could be in any module, anywhere in the system.

The net effect is that encapsulation and information hiding in Ada 95 is linked more closely to the naming hierarchy, making maintenance of Ada object-oriented systems easier to perform, even when the systems grow large and involve large hierarchies of types.

f) Ada 95 supports both object-oriented programming and multi-threaded programming, but does not directly integrate these two. Tasks and protected objects can be components of an object-oriented "type," or vice-versa, but neither tasks or protected objects can themselves be directly extended. By contrast, in Java, which is one of the very few other languages that have linguistic support for both object-oriented programming and multi-threading, synchronizing operations can be added in subclasses, and the types used to represent threads can also similarly be extended using the normal inheritance mechanisms.

An important advantage of Ada's tasking model is that all operations of a protected type or a task synchronize properly with one another, while in Java, it is possible to have both synchronizing and non-synchronizing operations on the same type, which is an obvious avenue for subtle race conditions to enter a system. Furthermore, because Ada's protected and task types do not allow piecemeal inheritance, all operations that synchronize with one another are defined in the same module, preserving the original advantages of the "monitor" concept introduced many years ago -- analysis and verification of proper synchronization conditions can be performed without having to chase down all critical sections that might be scattered about the system.

Given the above important differences between Ada 95 and most OOP languages, it is appropriate to evaluate these differences, and see whether they represent strengths or weaknesses in Ada's support for object-oriented programming. In some cases, the differences have both positive and negative aspects. Arguably one overall negative aspect of such differences is that they may put Ada 95 out of the mainstream of object-oriented programming, given that more and more programmers are being introduced to OOP, or even programming as a whole, through languages like Java and C#. On the other hand, Ada 95 has an important role in the development of complex, critical systems, and some of the differences are specifically designed to assist in the development of safe, robust, and verifiable systems, while still providing the flexibility and extensibility of object-oriented programming.

3 Areas of Strength, Areas for Enhancement

The challenge for this upcoming revision of Ada is then to preserve Ada's great strengths in its support for the construction of safe, verifiable systems, while enhancing its object-oriented features to take advantage of what has been learned about object-oriented programming features over the past ten years. Areas that have been identified for possible enhancement are support for multi-package cyclic type structures, support for multiple-inheritance type hierarchies, support for the "asymmetric" notation for invoking operations, and support for some kind of extension for protected and task types.

On the other hand, Ada's clear distinction between specific and class-wide types, its default of static binding with dynamic binding only where necessary, and its strong boundary around modules that have visibility on "partially" visible operations and components, are seen as clear advantages to Ada's approach to object-oriented programming, with no need for significant alteration. Furthermore, any changes that are proposed must not compromise Ada's strengths, and if anything, should extend Ada's unique position as the safe and verifiable, real-time object-oriented programming language.

4 Cyclic Type Structures

One item identified as very important for enhancement has to do with allowing cyclic type structures to cross package boundaries. In Ada 95, it is possible to use a combination of class-wide types, type extension, and "downward" type conversion, to overcome the basic Ada limitation to single-package cyclic type structures. However, this approach introduces additional complexity and some degree of run-time overhead and possible sources of run-time errors. Hence, there has been a concerted effort to provide a natural way for cyclic type structures to be safely and securely extended across packages.

Several alternative proposals have been developed and evaluated. Unfortunately, no one proposal has emerged as clearly the best solution in every dimension. The original proposal introduced a new kind of "with" clause called the "with type" clause. This allowed a package to refer to a type that would eventually be declared in some other package, but without requiring that that other package be compiled first. A version of this proposal was actually implemented in the GNAT Ada Compiler from AdaCore Technologies, but was ultimately dropped from consideration by the ARG because of difficulties discovered while working out the lower level details.

Three proposals remain under consideration: one involving type "stubs" (analogous to program unit "stubs", identified by the "is separate" syntax), a second involving a generalization of incomplete type declarations to allow a parent package to declare an incomplete type that will be completed in a child or nested package, and a third proposal involving a new kind of "limited" with clause, allowing one package to gain visibility on the types and nested packages of another package, without requiring "full" compilation of the other package.

Here are examples of the three proposals. They all are based on the Employee/Department problem, where there is a type that represents employees, and a type that represents departments, and employees are members of a department, while a department has a manager who is an employee. The challenge is to define the employee type in one package, and the department type in a separate package, but accommodate the desire to have references to both employees and departments in both packages.

The first example is the "type stub" proposal:

```
limited with Employees;
-- Allow type stubs to refer to this package
package Departments is
   type Employee is separate Employees.Employee;
   -- Type stub
   type Employee_Ref is access Employee;
   type Department is private;


   procedure Set_Manager(Dept: in out Department; Mgr:
Employee_Ref);
```

```
  function Manager(Dep: Department) return
Employee_Ref;
  ...
private
  type Department is record
    Mgr: Employee_Ref;
    ...
  end record;
end Departments;

limited with Departments;
-- Allow type stubs to refer to this package
package Employees is
  type Department is separate Departments.Department;
  -- Type stub
  type Department_Ref is access Department;
      type Employee is private;
      procedure Set_Department(Emp: in out Employee;
 Dept: Department_Ref);
   function Department(Emp: Employee) return
Department_Ref;
   ...
private
  type Employee is record
    Dept: Department_Ref;
    ...
  end record;
end Employees;
```

The second example uses the generalized incomplete type declaration:

```
package Office is
  type Employees.Employee;
  -- Incomplete type completed in child
  type Employee_Ref is access Employees.Employee;
  type Departments.Department;
  -- Incomplete type completed in child
  type Department_Ref is
access Departments.Department;
end Office;
  package Office.Departments is
  type Department is private;
  procedure Set_Manager(Dept: in out Department;
Mgr: Employee_Ref);
  function Manager(Dep: Department)
return Employee_Ref;
  ...
private
  type Department is record
    Mgr: Employee_Ref;
    ...
  end record;
end Office.Departments;

package Office.Employees is
  type Employee is private;
  procedure Set_Department(Emp: in out Employee;
Dept: Department_Ref);
  function Department(Emp: Employee) return
Department_Ref;
  ...
private
  type Employee is record
    Dept: Department_Ref;
    ...
  end record;
end Office.Employees;
```

The third example uses the "limited with" clause:

```
limited with Employees;
-- Gives visibility on types as incomplete types
package Departments is
  type Employee_Ref is access Employees.Employee;
  type Department is private;
  procedure Set_Manager(Dept: in out Department;
Mgr: Employee_Ref);
  function Manager(Dep: Department)
return Employee_Ref;
  ...
private
    type Department is record
      Mgr: Employee_Ref;
      ...
    end record;
end Departments;

limited with Departments;
-- Gives visibility on types as incomplete types
package Employees is
  type Department_Ref  is
access Departments.Department;
  type Employee is private;
  procedure Set_Department(Emp: in out Employee;
Dept: Department_Ref);
  function Department(Emp: Employee)
return Department_Ref;
  ...
private
  type Employee is record
```

```
      Dept: Department_Ref;
      ...
    end record;
  end Employees;
```

All three proposals allow a type defined in one package to be treated as an incomplete type in some other package, without the second package "depending" semantically on the first package. This is the critical capability, because it allows a cyclic type structure to be established without contradicting the partial ordering implied by "normal" semantic dependence relationships. All of the solutions involve a "weaker" kind of dependence, where one package knows that another package "exists" without having full semantic dependence on it. The "limited" with clause proposal approaches this problem by introducing a "limited" dependence on another package. Limited dependences are allowed to be cyclic. They imply some kind of pre-scan of a package to determine the names of the types (and the subpackages) of the package, without doing a full semantic analysis of the package.

The type stub proposal also requires a similar kind of limited dependence, but limits it even further to specific types identified by type stubs. Further, it does not require any kind of pre-scan of the package, because post-compilation checks can be performed to verify that type stubs refer only to types that actually exist in the package.

The incomplete-type-completed-in-a-child proposal introduces a "weak" dependence between a parent package and one of its child packages, requiring that a child package exist and that it declare a type that matches one identified in a generalized form of incomplete type declaration present in the parent's specification.

At this point there is consensus that a solution to this problem will exist in the Ada 200Y standard, and that the form of the solution will be based on one of these three proposals, but the particular approach has not yet been chosen. It is anticipated that the final choice will be made at the ARG meeting immediately following the Ada-Europe 2003 conference.

5 Multiple-Inheritance Type Hierarchies

When Ada 95 was designed, a significant amount of energy was expended in evaluating the possibility of including direct syntactic support for multiple inheritance. At the time, some OOP languages included full multiple inheritance (C++, Eiffel), while others chose single inheritance (Modula-3, Smalltalk). Full multiple inheritance introduces a number of language complexities as well as a somewhat more complicated and/or less efficient run-time model for dispatching calls. Ultimately, we decided to stick with the simplicity of single inheritance for Ada 95, but provide various "building blocks" that could be used to solve problems that in other languages might require multiple inheritance.

Since the Ada 95 design was finalized, a middle ground in the spectrum of inheritance models has become popular that provides multiple inheritance of interfaces (i.e. contracts), but with actual implementation "code" and data components inherited from only a single "primary" parent type. This approach, as exemplified in Java, C#, and to some extent CORBA IDL, eliminates much of the complexity of "full" multiple inheritance, because data components can continue to use the straightforward linear extension approach of single inheritance, and because conflicts due to inheriting code from multiple parent types cannot occur.

The current proposal for adding multiple inheritance of interfaces adds a new kind of type to Ada called an "interface". An interface type is in most respects equivalent to a type declared as "type T is abstract tagged null record;" though the syntax is shortened to be simply "type T is interface;". However, in addition to being usable in all contexts where such an abstract type may be used, the type may also be used as a "secondary" parent type in the declaration of a type extension. Secondary parents ("interface parents") are identified by appearing second or later in a list of the parent types in a record extension. The parent type names are separated from one another by "and", as in:

```
type NT is new Primary and Secondary_1 and
  Secondary_2 and ... with ...;
```

Note that the Primary parent may also be an interface type, since an interface type may be used anywhere an abstract tagged type make be used.

Interfaces may also be used as "parents" of other interfaces, using the following form:

```
type NI is interface with Int_Parent1 and Int_Parent2;
```

As implied above, no code or components are inherited from interfaces, only the specification of operations that must be implemented by the type that has the interface as a parent. If an interfaces has other interfaces as parents, then the union of all the operations of the parents combined with the operations defined on the new interface must be implemented by all (non-abstract) types derived from the new interface.

Here is a larger example which uses interfaces:

```
package MVC is
  -- Set of interfaces that define a model-view-controller
  -- structure.
  type Observer is interface;
  -- "interface" is roughly equivalent to
  -- "abstract tagged null record"
  type Observer_Ref is access all Observer'Class;
  -- An observer waits for changes to a model
  type Model is interface;
  type Model_Ref is access all Model'Access;
  -- A model represents some data structure
  -- that is being viewed and/or manipulated
  procedure Notify(
    Obs: access Observer;
    M: Model_Ref) is abstract;
  -- Notify observer that model it was observing
  --  has changed
  type View is interface with Observer;
  type View_Ref is access all View'Class;
  -- A view is a visual display of some model
  procedure Display_View(
    V: access View;
    M: Model_Ref) is abstract;
  -- Display view of associated model
  type Controller is interface with Observer;
  type Controller_Ref is access all Controller'Class;
  -- A controller supports input device(s) for
  --  manipulating/updating an underlying model
  procedure Start_Controller(
    Ctlr: access Controller;
    M: Model_Ref) is abstract;
  -- Initiate controller for associated model
  procedure Register_View(
    M: access Model;
    V: View_Ref) is abstract;
  -- Register view for given model.
  procedure Register_Controller(
    M: access Model;
    Ctlr: Controller_Ref) is abstract;
  -- Register controller on given model.
end MVC;

with MVC;
with Devices;
package Inputs is
  type Mouse is new Devices.Device with private;
  type Mouse_Controller is new
    Devices.Device and MVC.Controller with private;
  -- Primary parent type, if any, must be listed first
  -- All other parent types must be interfaces.
  procedure Handle_Input(
    MC: in out Mouse_Controller);
  -- Optionally override operations of parent type
  -- (or may inherit those with appropriate defaults)
  procedure Notify(
    MC: access Mouse_Controller;
    M: Model_Ref);
  procedure Start_Controller(
    MC: access Mouse_Controller;
    M: Model_Ref);
  -- Required to define all abstract operations declared
  -- for Observer and Controller
  type Two_Button_Mouse_Controller is new
    Mouse_Controller with private;
  procedure Start_Controller(
    TMC: access Two_Button_Mouse_Controller;
    M: Model_Ref);
  -- May inherit or override operations inherited from
  -- parent type including those that are needed for
  -- interfaces Observer and Controller
  procedure Register_And_Start(
    MC: access Mouse_Controller'Class;
    M: Model_Ref);
  -- Class-wide operation to register the mouse
  -- controller on given model, and then start the
  -- controller going.
private
  ...
end Inputs;
```

Although not illustrated in the above example, the proposal for interface types includes a proposal for "declared-null" procedures. A declared-null procedure is one whose specification ends with "is null;" rather than ";" or "is abstract;". No separate body is permitted for such a procedure. The implicit null body has no effect when executed.

Rather than requiring that all primitive operations of an interface type be abstract, this proposal also allows the primitive operations to be declared null. Such a procedure need not be overridden in a type derived from this interface. If not overridden, its implementation is null. If at least one interface ancestor of a type declares a given operation as null, the type need not provide an explicit overriding of the operation. If a non-interface ancestor type provides a non-null implementation of the operation, that is inherited rather than the null procedure.

Declared-null procedures are useful in that they allow a number of optional capabilities to be supported in an interface, without every derived type having to explicitly define the capability. In addition, if an abstract or interface type with one or more declared-null primitives is used as the ancestor in a generic formal type extension, the formal type is presumed to have non-abstract implementations of these operations. This can be useful when overriding the operations, since it is often desirable to call the parent's operation from an overriding, particular in the case of initializing or finalizing operations.

6 Using Object.Operation Notation

When doing object-oriented programming in Ada 95, the programmer must identify the package in which an operation is declared, along with the various operands. Because dispatching operations are often implicitly declared, identifying the package where they are declared can sometimes be confusing. In addition to dispatching operations, class-wide operations are important in many object-oriented systems. However in Ada, class-wide operations, unlike their equivalent in many other OOP languages, are not inherited along with the dispatching operations. Instead, they are only declared in the original package where they appear.

This distinction in inheritance between dispatching operations and class-wide operations means that it can be a burden to identify the package where an operation of interest is declared, particularly when the choice between making an operation a dispatching operation versus a class-wide operation might be more of an implementation detail than an essential part of the semantics of the operation from a user's point of view. The distinction is generally important when deriving from a type, but may be irrelevant when using the type.

Programmers familiar with other OOP languages that use an "object.operation(...)" syntax rather than Ada's "package.operation(object, ...)" syntax find this added burden an entry barrier to using Ada for OOP systems, which tends to make the language feel less object-oriented than it truly is. The alternative of inserting "use" clauses for every possible package where an operation might be declared has other negative ramifications.

Given these considerations, a proposal has been developed to allow the use of an "object.operation(...)" syntax as a syntactic shorthand for "package.operation(object, ...)". Originally it was proposed that this syntactic shorthand be available to all kinds of types, whether or not the type is tagged. However, supporting this for both access types and tagged types adds to the complexity of the proposal in certain ways due to the desire to allow implicit dereference (implicit ".all") of the "object" if it is designated by an access value. Implicit dereference is provided in all other places where "." is allowed in the syntax, and it would be inconsistent not to allow it here. Furthermore, this notation is specifically intended to simplify object-oriented programming where there may be multiple relevant packages. When using non-tagged types, the object.operation syntax would not provide as much benefit.

The basic idea of this restricted proposal is that any dispatching operation, or any class-wide operation declared in a package where the corresponding specific type is declared, is eligible for calling via this shorthand, so long as the first formal parameter is a controlling parameter, or is of the class-wide type. When the object.operation syntax is used, the "operation" is looked up first as a component, and then as though the packages where the type and any of its ancestors are declared had been made use-visible. If the object were of an access-to-tagged type, an interpretation using an implicit dereference would also be considered. If there are possible interpretations of "operation" among these packages, it is checked to see if any of them are subprograms where "object", "object.all", or "object'access" could be passed as the first parameter, and any actual parameters given in parentheses after "operation" correspond to the remaining formals.

Here are some examples of use of this shorthand:

Given the MVC and Inputs packages given above:

```
   M : MVC.Model_Ref;
   V : MVC.View_Ref;
   C : MVC.Controller_Ref;
   MC : aliased Inputs.Mouse_Controller;
 begin
   V.Display_View(M);
   -- equivalent to MVC.Display_View(V, M);
   MC.Start_Controller(M);
   -- equivalent to Inputs.Start_Controller(MC'Access, M);
   MC.Handle_Input;
   -- equivalent to Inputs.Handle_Input(MC);
   MC.Register_And_Start(M);
   -- equivalent to
   -- Inputs.Register_And_Start(MC'Access, M);
   -- (this is a call on a class-wide op)
```

7 Inheritance of Interfaces for Protected and Task Types

During the Ada 95 design process, it was recognized that type extension might be useful for protected types (and possibly task types) as well as for record types. However, at the time, both type extension and protected types were somewhat controversial, and expending energy on a combination of these two controversial features was not practical.

Since the design, however, this lack of extension of protected types has been identified as a possible target for future enhancements. In particular, a concrete proposal appeared in the May 2000 issue of ACM Transactions on Programming Languages in Systems (ACM TOPLAS[1]), and this has formed the basis for a language amendment (AI-00250).

However, in ARG discussions, the complexity of this proposal has been of concern, and more recently a simpler suggestion was made that rather than supporting any kind of implementation inheritance, interfaces for tasks and protected types might be defined, and then concrete implementations of these interfaces could be provided. Class-wide types for these interfaces would be defined, and calls on the operations (protected subprograms and entries) defined for these interfaces could be performed given only a class-wide reference to the task or protected object.

An important advantage of eliminating inheritance of any code or data for tasks and protected types is that the "monitor"-like benefits of these constructs are preserved. All of the synchronizing operations are implemented in a single module, simplifying analysis and avoiding any inheritance "anomalies" that have been associated in the literature with combining inheritance with synchronization.

The detailed syntax for protected and task interfaces has not been proposed. Here is one possibility:

```
protected interface Queue is
-- Interface for a protected queue
  entry Enqueue(Elem : in Element_Type) is abstract;
  entry Dequeue(Elem : out Element_Type) is abstract;
  function Length return Natural is abstract;
end Queue;
type Queue_Ref is access all Queue'Class;

protected type Bounded_Queue(Max: Natural) is new
  Queue with
-- Implementation of a bounded, protected queue
  entry Enqueue(Elem : in Element_Type);
  entry Dequeue(Elem : out Element_Type);
  function Length return Natural;
private
  Data: Elem_Array(1..Max);
  In_Index: Positive := 1;
  Out_Index: Positive := 1;
  Num_Elems: Natural := 0;
end My_Queue;

task interface Worker is
   -- Interface for a worker task
   entry Queue_To_Service(Q : Queue_Ref)
     is abstract;
end Worker;
type Worker_Ref is access all Worker'Class;

task type Cyclic_Worker is new Worker with
  -- Implementation of a cyclic worker task
  entry Queue_To_Service(Q : Queue_Ref);
end Cyclic_Worker;

task Worker_Manager is
  -- Task that manages servers and queues.
  entry Add_Worker_Task(W : Worker_Ref);
  entry Add_Queue_To_Be_Serviced(Q : Queue_Ref);
end Worker_Manager;

task body Worker_Manager is
  Worker_Array : array(1..100) of Worker_Ref;
  Queue_Array : array(1..10) of Queue_Ref;
  Num_Workers : Natural := 0;
  Next_Worker : Integer := Worker_Array'First;
  Num_Queues : Natural := 0;
  Next_Queue : Integer := Queue_Array'First;
begin
  loop
    select
      accept Add_Worker_Task(W : Worker_Ref) do
        Num_Workers := Num_Workers + 1;
        Worker_Array(Num_Workers) :=
          Worker_Ref(W);
      end Add_Worker_Task;
      -- Assign new task a queue to service
      if Num_Queues > 0 then
        -- Assign next queue to this worker
        Worker_Array(Num_Workers).
          Assign_Queue_To_Service(
            Queue_Array(Next_Queue));
        -- Dynamically bound entry call
        -- Advance to next queue
        Next_Queue := Next_Queue mod
          Num_Queues + 1;
      end if;
    or
      accept Add_Queue_To_Be_Serviced(
        Q : Queue_Ref);
        Num_Queues := Num_Queues + 1;
        Queue_Array(Num_Queues) := Queue_Ref(Q);
      end Add_Queue_To_Be_Serviced;
         -- Assign queue to worker if enough workers
      if Num_Workers >= Num_Queues then
        -- This queue should be given one
        -- or more workers
        declare
          Offset : Natural := Num_Queues-1;
        begin
          while Offset < Num_Workers loop
            -- (re) assign queue to worker
            Worker_Array((Next_Worker + Offset -
              Num_Queues) mod Num_Workers + 1).
                Assign_Queue_To_Service(
                  Queue_Array(Num_Queues));
            -- Dynamically bound call
            Offset := Offset + Num_Queues;
          end loop;
          -- Advance to next worker
          Next_Worker := Next_Worker  mod
            Num_Workers + 1;
          end;
        end if;
      or
```

```
          terminate;
        end select;
      end loop;
    end Worker_Manager;

      My_Queue : aliased Bounded_Queue(Max => 10);
      My_Server : aliased Cyclic_Server;
    begin
      Worker_Manager.Add_Worker_Task(
        My_Server'Access);
      Worker_Manager.Add_Queue_To_Be_Serviced(
        My_Queue'Access);
      …
    end;
```

8 Summary

The four proposed amendments to the Ada standard discussed above are in some sense an attempt to "finish" the job of integrating object-oriented programming into Ada which was started during the Ada 95 revision process. Although the existing OOP features in Ada 95 are both powerful and flexible, eight years of use and ongoing developments in the object-oriented programming language community have suggested opportunities for enhancement.

Although it is likely that some of these amendments will be approved for addition to the standard, it is quite possible that some will not, or that the proposals will be further refined in minor or major ways. Hence it is essential to keep in mind that this is a snapshot of an ongoing revision process, and by no means the final story. For those interested in tracking the progress of these amendments, the website of the Ada Conformance Assessment Authority (ACAA) provides ready access to all of the amendments, as well as minutes of ARG meetings. The URL for this website is:

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

References

[1] Wellings, A.J.; Johnson, B.; Sanden, B.; Kienzle, J., Wolf, Th., and Michell, S.: "Integrating Object-Oriented Programming and Protected Objects in Ada 95", ACM TOPLAS 22 (3), May 2000; pp. 506-539.

P. Leroy, “Memories of a Language Designer”
Originally printed in *Ada User Journal*, Vol. 27, N. 3, September 2006

Memories of a Language Designer

Pascal Leroy *

IBM Rational software; Chairman, Ada Rapporteur Group; email: pascal.leroy@fr.ibm.com

Introduction

On September 27, 2006, ISO/IEC JTC 1/SC 22, the ISO subcommittee in charge of standardizing programming languages, approved the Amendment to the Ada standard (“Ada 2005” in the vernacular) by twelve votes in favour[1], one abstention[2] and two non-voting countries supporting approval[3]. No comments were submitted as part of the vote, so the definition of Ada 2005 is now frozen as only administrative chores remain to be performed before the new standard is officially published by ISO. In parallel to this, the Ada standard will be published this fall in the prestigious Lecture Notes in Computer Science. After working on this Amendment for many years, it is now an interesting time to look back into the mirror.

1 Inception

Shortly after Ada 95 was standardized minor issues and questions concerning the language were addressed to the Ada Rapporteur Group (ARG). Though none of these were earth-shattering, there were anomalies and inaccuracies in the Reference Manual that needed fixing. In 1996 to address those issues the ARG started working on a Technical Corrigendum, which was completed in 2001.

In the meanwhile it became clear that there would not be financial room for another massive revision effort like the Ada 9X project. Future evolutions of the language would thus have to happen as part of a volunteer effort from people and organizations having an interest in Ada. As the user community started to use Ada 95 on real-life projects, they encountered a number of annoyances that could not be fixed by incremental changes. These early annoyances included for instance the impossibility of creating cyclic dependencies among package specifications, and the lack of support for interfacing with C or C++ unions.

Consensus quickly arose that it was necessary to put in place a framework for keeping Ada “alive” through a controlled revision process capable of preserving the benefits of standardization while also allowing room for improvements to the language.

2 History

As a consequence of those considerations, in 1998 the ARG was tasked by WG 9 (the ISO working group in charge of maintaining the Ada standard) to start studying “language enhancements”. It is interesting to notice that by that time some of the most significant issues related to object-oriented programming and program structure were already identified: in addition to solutions to the cyclic dependencies problem, the suggestions made in 1998 included explicit control of overriding, upward-closure for subprogram parameters, and Java-style interfaces.

In fact, actual work on the Amendment did not really start until late 2000, in part because the focus was first on completing the Technical Corrigendum, and in part because more return on experience was needed before deciding what areas of the language actually required improvement.

In the summer of 2001 WG 9 asked for an Amendment to be developed with a target date of 2005 (this being software-related, it should not come as a surprise that we are now running some 9 months behind schedule on a 5-year project). WG 9 later approved a more formal and detailed schedule, as well as directions regarding the kinds of enhancements that the ARG should consider.

By that time it had become apparent that enhancements had to be developed to better support real-time and high-integrity systems, with the Ravenscar profile being the first item on the list. In the following years, numerous proposals relevant to this specific application area were developed by the International Real-Time Ada Workshop (IRTAW) and forwarded to the ARG for inclusion in Ada 2005 (though not without extensive rework in some cases!).

It had been clear from the beginning that expanding the predefined library was an essential goal of Ada 2005. Early on we worked on a package for accessing directories and file systems in a portable way, and we chose to include the matrix and vector facilities described in standard ISO/IEC 13813. Still, the topic that everyone had in mind was a predefined library of containers, though that looked like a daunting effort. To make that happen we decided to harness the help of the user community, and asked for proposals on this subject. After careful study of the two proposals that we received, we felt that neither of them was ideal. We thus decided to craft a third alternative by picking the good ideas in both submissions. This third alternative was initially given a very restricted scope, for we didn’t want to miss the Amendment deadline. As it turned out, however once the core ideas had stabilized it was possible to add more packages, so that the final library may be deemed reasonably complete.

Interestingly the work on containers proved to be a valuable usability test for the new language: as difficulties were encountered in the development of the library, new features had to be added to the core language. Nested type

[1] Canada, China, Denmark, France, Germany, Japan, Romania, Russian Federation, Spain, Switzerland, United Kingdom, United States.

[2] The Netherlands.

[3] Belgium, Italy.

**** Affiliation and contacts as in the original publication***

extensions and partial parameter parts for formal packages originated in this manner.

By the summer of 2004 the scope of the Amendment was pretty much stabilized, and it was clear which of the major proposals were in and which were out. What remained to be done was “mere” integration work. In fact, this proved much more time consuming than we anticipated. This had to do in part with the sheer size of the updated Reference Manual (nearly 1100 pages in its annotated version) and in part with the fact that upon reviewing all the changes “in place” we discovered inconsistencies that required rather extensive rework. For instance, some of the rules related to the inheritance of limited-ness, or to functions returning access results, came very late in the integration process and therefore required considerable attention. Overall, it took very intense work for everyone in the ARG over a period of 18 elapsed months before we had a document that could be submitted to WG 9 for approval.

3 Lessons Learned

It should not be surprising that “real” things take longer to finalize than one may initially expect. Some new features, like the limited_with_clause, proved extremely difficult to design: as many as seven proposals were considered over the course of five years before a satisfactory solution was arrived at: every time we tried a new idea, it seemed like it was breaking a fundamental invariant of the language. Interfaces have had a different story, but an equally complicated one at that: the basic ideas were essentially in place by the end of 2000, yet their entanglement with the rest of the language is so deep that we have been revising them literally until the last minute. Even apparently simple enhancements, like allowing aggregates for limited types, turned out to have unexpected consequences that required heart-wrenching decisions (for example, giving up on aggregates for private types).

It is amusing to notice that the ARG started out fairly timorous in the changes it contemplated: an early proposal for explicit control of overriding entailed making use of pragmas because the notion of adding new syntax was considered heretic. As it became clear that pragmas would be terrible for readability, we slowly warmed to the notion of new syntax. Thus we became increasingly bolder: when nested type extensions were added in 2003, they were easily swallowed, even though we knew that they would have a considerable impact on compilers.

While we were very much driven by the user community, a number of changes appeared out of our own work as we ran either into inconsistencies in Ada 95 or into unpleasant non-uniformities in Ada 2005. For instance, we initially added null_exclusions and improved support for anonymous access types so as to ease programming with access types (especially in the context of OOP). But for long we didn’t want to allow anonymous access types as function results because of the deep language design difficulties that go with them. It was only when we started to use the new language for predefined units and for realistic examples that we discovered that it was an unacceptable limitation and decided to bite the bullet.

4 Regrets?

There are a number of ideas that we discussed for a long time, and on which considerable effort was expended, but which were not included in the final Amendment because we could not find a satisfactory solution. Partial generic instantiations come to mind, as do support for the IEEE 559 floating-point model, and pre- and post-conditions for types and packages. In all cases, integrating the new features into the language was hard and the solutions just didn’t “feel right”. Some of those ideas might in time mature to become valuable additions for a future version of Ada.

I suppose that every member of the ARG has his or her set of favourite features that didn’t make it into the Amendment for one reason of another. Since I have the opportunity to do it here, I shall name my top three. I think it was unfortunate that we could not find a solution to the problem of partial generic instantiation, which hampers the usability of generics to compose abstractions; I guess that the discussion on this problem started too late and that we didn’t have enough time to find the “magic” idea that would solve it elegantly. I also regret that we didn’t have time to revise the exception mechanism: exceptions in Ada are frustratingly limited compared to other languages, but improving them without compromising performance and compatibility is a tough call. Finally, I wish we had had the guts to add **out** and **in out** parameters to functions, although I realize that this is a very controversial topic.

5 With (More Than) a Little Help…

It is often claimed that Ada was designed by a committee. Nothing could be further from the truth. The ARG is made of experts from the user community, the tools vendor community, and academia, who are among the best minds in our industry. It is important to stress that political bickering has no place in the ARG at all, and that although we have had a fair share of heated discussions, proposals were always judged on their technical merits. All the changes that were ultimately incorporated in the Amendment were elaborated by combining the best ideas, and were agreed upon quasi unanimously by the ARG.

I want to conclude by profusely thanking my fellow ARG members[4], who have devoted so much time and energy to bringing this effort to fruition, and the Convener of WG 9[5], who so deftly shepherded the Amendment through the Byzantine ISO administration. I have been lucky and honoured to work closely with all of them during all these years.

[4] Steve Baird, John Barnes, Randy Brukardt, Alan Burns, Robert Dewar, Gary Dismukes, Robert Duff, Kiyoshi Ishihata, Steve Michell, Erhard Ploedereder, Jean-Pierre Rosen, Ed Schonberg, Tucker Taft, Bill Thomas, Joyce Tokar, and Tullio Vardanega.

[5] James Moore.

National Ada Organizations

Ada-Belgium

attn. Dirk Craeynest
c/o K.U. Leuven
Dept. of Computer Science
Celestijnenlaan 200-A
B-3001 Leuven (Heverlee)
Belgium
Email: Dirk.Craeynest@cs.kuleuven.be
URL: www.cs.kuleuven.be/~dirk/ada-belgium

Ada in Denmark

attn. Jørgen Bundgaard
Email: Info@Ada-DK.org
URL: Ada-DK.org

Ada-Deutschland

Dr. Peter Dencker
Steinäckerstr. 25
D-76275 Ettlingen-Spessartt
Germany
Email: dencker@web.de
URL: ada-deutschland.de

Ada-France

Ada-France
attn: J-P Rosen
115, avenue du Maine
75014 Paris
France
URL: www.ada-france.org

Ada-Spain

attn. José Javier Gutiérrez
Ada-Spain
P.O.Box 50.403
28080-Madrid
Spain
Phone: +34-942-201-394
Fax: +34-942-201-402
Email: gutierjj@unican.es
URL: www.adaspain.org

Ada in Sweden

Ada-Sweden
attn. Rei Stråhle
Rimbogatan 18
SE-753 24 Uppsala
Sweden
Phone: +46 73 253 7998
Email: rei@ada-sweden.org
URL: www.ada-sweden.org

Ada Switzerland

attn. Ahlan Marriott
White Elephant GmbH
Postfach 327
8450 Andelfingen
Switzerland
Phone: +41 52 624 2939
e-mail: ada@white-elephant.ch
URL: www.ada-switzerland.ch