ADA USER JOURNAL

Volume 25
Number 4
December 2004

Contents

Editorial Policy for *Ada User Journal*

Publication

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

Aims

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

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

The Journal publishes the following types of material:

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

Further details on our approach to these are given below.

Original Papers

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

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

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

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

Unless the authors state explicitly otherwise, submission of an article is taken to imply that it represents original, unpublished work, not under consideration for publication elsewhere.

News and Product Announcements

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

Reprinted Articles

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

Commentaries

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

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

Announcements and Reports

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

Reviews

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

Submission Guidelines

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

Editorial

The December issue of the Ada User Journal marks the end of the calendar year 2004 with two article contributions that symbolize the relay between the proceedings of the past year and the exciting highlights of the year to come, which promises to be extremely rich with Ada-related events. Both articles are authored by important and valued members of the Ada community. Ben Brosgol, the author of the first article, has kindly offered us an outline of the tutorial that he presented at Ada-Europe 2004. With this contribution, the Journal has provided substantial coverage of the main tutorials that use to complement the core programme of the Ada-Europe conference week. This coverage will continue for the future editions of the conference. The second article is a very precious gift that the Journal is proud to make to its readership at large: thanks to the support of Ada-Europe, starting with this issue, the Journal will be hosting a series of articles by John Barnes (a name that needs no introduction) that cover the preliminary version of the Ada 2005 Rationale. Yes, there will be an Ada 2005 Rationale after all, to accompany the publication of the 2005 amendment of the Ada programming language, and yes, the Rationale will be authored by John Barnes, a promise of very attractive, informative and witty read. We do hope that our readers will enjoy this contribution as much as we do. The readers will also notice that this issue is thicker than usual, which they should take as a token of the health of the Ada community and the wealth of its news and events, for which our News editor, Santiago Urueña, and his long-time precessor Dirk Craeynest offer comprehensive coverage as usual.

A few words are in order to comment on the incident that caused the issue 25-3 of the Journal to come to you with a disappointing number of character and printing errors. As you may have heard from your local distributor, the incident occurred at a stage of the journal production that was beyond our control. We have taken immediate steps with the Printer to make sure that an error-free version of the very same issue be dispatched to you in a timely fashion and that mishaps of this kind do not occur again. We are hopeful that you will have received that correct version of AUJ 25-3 prior to receiving this one.

In closing this editorial at last, I extend my best wishes for New Year to our whole readership, our fellow members of Ada-Europe, its Board and its sibling national organisation

Tullio Vardanega
Padova
December 2004
Email: tullio.vardanega@math.unipd.it

News

Santiago Urueña

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

Contents

Ada-related Organizations

ARTiSAN Software Tools Relocates Corporate Headquarters

ARTiSAN Software Tools Relocates to Eagle Tower, Tallest Office Building in Cheltenham, to Accommodate Projected Growth

Cheltenham, UK - July 1, 2004 - ARTiSAN Software Tools, a global leader for UML-based, real-time systems and software modeling tools, today announced that they have relocated their corporate headquarters to Eagle Tower, one of Cheltenham's most exclusive business addresses.

"The move marks an exciting time in Artisan's history," noted Jeremy Goulding, ARTiSAN's President and CEO. "We're profitable and had double digit quarter-to-quarter growth over the last 6 quarters and it was time that we moved into a building that can accommodate our needs as this trend continues. We're on one floor of the Tower now, and with it having about 100,000 square feet of modern office space on 12 floors, there's plenty of room to meet the demands of our continued growth."

"We've made a significant investment in a state of the art training facility, and will be growing in this area to support the increasing customer demand for SysML, UML 2.0, and specific object oriented tool supported C++ & Ada language training. There is much confusion over the UML 2.0 standard, and people are turning to ARTiSAN, as one of the driving forces behind the standard, for their training."

Eagle Tower represents one of the most prestigious office addresses in Gloucestershire and is a well-known local landmark. Located in the heart of Cheltenham's fashionable Montpellier quarter, the tower is close to the renowned shopping parade, the Promenade, and many of the town's finest bars and restaurants. A photo of the tower can be seen at http://www.artisansw.com/images/eagle.jpg.

The company's new address is:

ARTiSAN Software Tools, Ltd
Suite 701 Eagle Tower
Montpellier Drive
Cheltenham
Gloucestershire
GL50 1TA UK

AdaCore Partners with Praxis Critical Systems on a Joint Academic Initiative

Paris, France and London, UK - October 1, 2004

As part of the Ada Academic Program, AdaCore is pleased to announce a joint initiative with Praxis Critical Systems Ltd. AdaCore's GNAT Academic Program (GAP) will be linked to the Praxis Academic Support Programme - a fully supported professional SPARK toolset offered free-of-charge to university faculty members for teaching and/or research. SPARK is a high-integrity subset of the Ada programming language. The SPARK Examiner checks conformance of code against the rules of SPARK, performs flow analysis and can generate Verification-Conditions for full formal proof of SPARK source code. In conjunction with the SPADE Simplifier and the SPADE Proof Checker, Praxis provides a suite of tools capable of aiding the development, testing and verification of high integrity systems written in SPARKAda.

"The unique properties of the SPARK language and its support tools depend on the solid foundations provided by the Ada language so we are naturally delighted to support AdaCore's far-sighted Ada Academic Programme. The combination of the AdaCore and Praxis programmes provides a one stop resource offering the very best in software engineering tools and teaching materials". Peter Amey, Chief Technical Officer, Praxis Critical Systems.

"Our joint initiative with Praxis reinforces the commitment to Ada within Academia by widening the scope of on-line expertise as well as expanding the wealth of teaching materials available to instructors in Ada, wherever their faculty may be based." Louise Arkwright, Ada Academic Program Manager, AdaCore."

Praxis Critical Systems Limited and High Integrity Systems Limited are merging to create Praxis High Integrity Systems

11 October 2004 - Praxis Critical Systems Limited and High Integrity Systems Limited are merging to create Praxis High Integrity Systems, a company dedicated to delivering the best in high integrity systems engineering. The companies will trade as Praxis High Integrity Systems from 11th October 2004 and will formally merge from 3rd January 2005.

The new company remains part of the 16,000 strong Altran Group, a recognised global leader in innovative engineering and technical partner to the Renault F1 team. Praxis High Integrity Systems will lead the service offering in High Technology Engineering for Altran's US and Northern European business. Altran expects to invest in further growth and international development for the new company, already employing 120 engineers and consultants known for the superlative quality of their contribution to developing vital systems across business and industry.

For more than 20 years, Praxis and HIS have already successfully delivered systems and engineering consultancy across the Aerospace and Defence, Automotive, Rail, Nuclear, Telecommunications and Finance sectors, so the merged company will be a stronger force in these markets in future. Combining proven domain knowledge and experience with new service offerings such as Security Engineering and Delivery Management means more opportunities for new and existing clients to work with the new team.

In a business environment where clients want dependable supply partners able to offer leading-edge services and innovative solutions, Praxis High Integrity Systems delivers. Our client-focussed approach is specifically aimed at understanding client's needs and challenges and adapting our best-in-class principled engineering to meet those needs. Our goal is always to

deliver the right solution at the right quality - fit for purpose first time.

With 'lean' engineering processes understood by all our staff, Praxis HIS brings a team approach that makes rapid strides in building resilient solutions to highly complex problems. And to prove it has ultimate confidence in its abilities to do so, Praxis HIS regularly arranges fixed price and gainshare contracts with clients.

Our mission is to be a successful and agile company founded on technical engineering excellence, capable of creating and developing the best engineering talent. The new organisation is structured to deliver this with a mix of Market Sector specialists and Communities of Expertise (CoEx), including Software Engineering, Systems Engineering (including Requirements Engineering), Project/Operational Risk Management, Programme Delivery Management, Safety Engineering, Security Engineering and Human Factors. Each CoEx is hard at work delivering not just expertise into customer projects but also Intellectual Property in the form of software products, training courses, tool templates and more.

Introducing a new look for ACT Europe

From: Cyrille Comar <comar@adacore.com>
Date: Mon, November 15, 2004 6:33 pm
To: announce@adacore.com
Subject: Introducing a new look for ACT Europe

We are pleased to introduce a new name -- AdaCore - and a new logo. This new name reflects our ongoing commitment to Ada. AdaCore is dedicated to providing quality software and services, and our new look serves to convey the professionalism synonymous with our GNAT Pro package.

What does the new name mean for you?

This change does not affect the products or support we provide our customers. You can use the new report address, report@adacore.com to communicate with our technical team. For sales inquiries please send e-mail to sales@adacore.com. The corporate website is at www.adacore.com, and the GNAT Tracker web server is available at www.adacore.com/gnattracker. Of course the old addresses will continue to work, so you can switch to the new scheme at your convenience.

If you have any questions about the name change, please do not hesitate to contact sales@adacore.com.

DDC-I "SCOREs" Experienced Italian Distributor

Phoenix, AZ - September 15, 2004 - In response to increasing demand in Italian industrial markets, DDC-I today announces the signing of experienced real-time vendor ARTiSAN Software Tools (Srl) as exclusive distributor of DDC-I tools in Italy, including the versatile SCORE (Safety Critical, Object-oriented, Real-time Embedded) integrated development environment.

"Our relationship with ARTiSAN represents an excellent opportunity for real-time software developers in Italy," explains DDC-I President and CEO Dr. Ole N. Oest. "SCORE's participation in the European Open Microprocessor Initiative presents it as a compelling choice for every budget-conscious Italian software developer."

Cheltenham, UK-based ARTiSAN - a global leader in UML-based real-time systems and software modeling tools - appointed Carmelo Tommasi, an industry veteran with over 25 years of multinational management experience at Mentor Graphics, Harris, Viewlogic and Telelogic, to distribute ARTiSAN Software Tools products in Italy.

ARTiSAN Software Tools Srl serves important customers in aerospace and telecom markets, thanks to Mr. Tommasi's market awareness and first-hand knowledge, recognizing increasing customer demand for SysML, UML 2.0, and specific object-oriented tools supporting flexible development in C++ & Ada.

"Our current efforts are focused in aerospace, due to the mission-critical nature of the software development, and on telecom, which has traditionally been strong in Italy. Given the nature of the Italian industrial market, small and medium size companies also represent a fertile field for tool vendors offering flexible products like SCORE," Tommasi concludes.

Aonix Joins Eclipse to Provide Customers a Common Platform

Product road map charts integration of Eclipse plug-ins

San Diego, CA, Paris, France, June 10, 2004

Aonix, an independent global company delivering complete solutions for safety- and mission-critical applications, has joined Eclipse, a community committed to the implementation of an universal platform for tools integration. In addition to porting its Ada95, PERC and Ameos tool suites to the Eclipse platform, Aonix plans to deliver an Eclipse-based IDE targeted to developers of mission- and safety-critical solutions.

"The direction of the Eclipse Platform complements our strategy of delivering common platforms that enable embedded developers to mix tools and secure former and future development through interoperable technologies," noted Jacques Brygier, vice president of marketing at Aonix. "In the mission- and safety-critical market, our customers come from a number of industry sectors, each carrying its own certification standards and specialized tools. Like Eclipse, we have focused on the development of common platforms, particularly with our products based on our SmartKernel real-time executives family, in which our tools - or the next vendors' - work seamlessly, enabling developers to focus on building applications, not on integrating tools."

"Aonix's support for Eclipse affirms the benefits of a universal platform in delivering mission- and safety-critical tools," noted Mike Milinkovich, Eclipse executive director. "We welcome Aonix into Eclipse and look forward to collaboration and future contributions as they extend the Eclipse Platform into their offerings and industry segment."

Aonix already offers different levels of Eclipse integration with its PERC and Ameos products. In the near future, Aonix will integrate its Ada 95-based development environment and the first version of the SmartKernel product family. The Eclipse-based IDE common to all Aonix products is under development and a beta copy will be released in the second half of the year.

About Eclipse

Eclipse has established an open-source eco-system of tools providers and consumers by creating technology and an open universal platform for tools integration. The open-source Eclipse community creates royalty-free technology as a platform for tools integration. Eclipse based tools give developers freedom of choice in a multi-language, multi-platform, multi-vendor supported environment. Eclipse delivers a plug-in based framework that makes it easier to create, integrate and use software tools, saving time and money. By collaborating and sharing core integration technology, tool producers can concentrate on their areas of expertise and the creation of new development technology. The Eclipse Platform is written in the Java language, and comes with extensive plug-in construction toolkits and examples. It has already been deployed on a range of development workstations including HP-UX, Solaris, AIX, Linux, MAC OS X, QNX and Windows based systems. Full details of the Eclipse community and white papers

documenting the design of the Eclipse Platform are available at www.eclipse.org.

Ada-related Events

[The announcements reported below are a selection of the many Ada-related events organized by local groups. If you are organizing such an event, feel free to inform us as soon as possible. If you attended one please consider writing a short report for the Journal. -- su]

Nov 14-18 - SIGAda 2004 Conference

From: Ricky E. Sward <ricky.sward@ix.netcom.com>
Date: 27 Aug 2004 07:57:58 -0700
Subject: SIGAda 2004 Conference
Newsgroups: comp.lang.ada

Conference Announcement - SIGAda 2004

14-18 November 2004, Atlanta, Georgia, USA

Sponsored by ACM SIGAda

The SIGAda 2004 conference offers a top-quality technical program focused on important strengths of the Ada programming language. Three days of technical papers, keynotes, and invited presentations will report on how Ada is achieving success in the challenging realm of software engineering. We are pleased to announce that three leaders in the software engineering community, Pam Thompson from Lockheed Martin Aeronautics, Watts Humphrey from the Software Engineering Institute, and Stephen Cross from the Georgia Tech Research Institute, will provide keynote addresses. We are also fortunate to have key members of the WG9 Ada Rapporteur Group (ARG) who will participate in a 3-hour panel on the significant improvements that WG9 has approved for inclusion in the Ada 2005 Amendment. The Panel will be chaired by IBM Rational's Pascal Leroy, Chair of the WG9 ARG.

Beyond the formal conference of selected papers and presentations, SIGAda 2004 offers two days of outstanding tutorials led by some of the most respected technical leaders in the industry. SIGAda's tutorials and workshops provide full- or half-days for those working the same issues to share with each other and leverage everyone's accomplishments; workshop products are "delivered" to the community. The broad offerings of career-enhancing tutorials include basic Ada 95 introductions for software engineers new to Ada, intermediate and advanced Ada topics for practitioners striving to expand their Ada expertise, and several language-independent technology topics. Join us in understanding how these topics mutually support the disciplined development and evolution of serious, high quality software systems.

The following tutorial was added after the Advanced Program was published:

A#: Programming PDAs and .NET devices with Ada
This tutorial describes A#, an Ada environment for programming the Windows .NET and .NET Compact Frameworks. Attendees will learn how to create Ada applications that take advantage of the rich set of libraries available with the .NET Framework, and also how to deploy Ada applications onto PDAs using the .NET Compact Framework.

For more information on the conference schedule, registration options, and hotel information visit the SIGAda 2004 web site at the following link.

http://www.acm.org/sigada/conf/sigada2004

Oct 06-07 - Ada-Deutschland Tagung 2004

Ada Germany Conference 2004: Automotive Safety and Security

"Security and Reliability for Automobile Information Technology"

October 2004 - SofCheck Chairman and CTO, Tucker Taft, presented the keynote lecture "Ada 2005 - Putting It All Together" to the Ada Germany Conference 2004 on Reliable Software Systems.

View the presentation in PDF format:

[http://www.sofcheck.com/news/adagermany2004rev3.pdf --su]

[See also the announcement (in German) of this conference in AUJ 25-2 (Jun 2004), p.85. --su]

Feb 26-27 - Ada event at FOSDEM 2005

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Date: 5 Oct 2004 21:36:28 +0200
Organization: Ada-Belgium, c/o Dept. of Computer Science, K.U.Leuven
Subject: AdaFosdem mailing list created for Ada event at FOSDEM 2005
Newsgroups: comp.lang.ada,fr.comp.lang.ada

Recently, there was a discussion on the Ada-Belgium members' mailing list about the possibility of organizing an Ada event at FOSDEM 2005 in Brussels (Sat-Sun 26-27 February 2005, www.fosdem.org).

FOSDEM is the "Free and Open Source Software Developers' European Meeting" held annually in Brussels; the 4th edition attracted some 2000 participants earlier this year.

I'm pleased to announce that to coordinate among potential speakers and attendants, an AdaFosdem mailing list has been created. Several relevant pre-list postings have been included in the list's archives.

We hereby invite everyone interested in an Ada event at FOSDEM 2005 to subscribe (*) to the AdaFosdem mailing list in order to contribute ideas and suggestions. We recommend new subscribers to browse the archives for September and to offer comments.

To subscribe and to consult the list archives, use URL http://listserv.cc.kuleuven.ac.be/archives/adafosdem.html

To post to the list (after subscribing), send e-mail to adafosdem@listserv.cc.kuleuven.ac.be

Looking forward to see much activity on the new list,

Ludovic Brenta, Ada at FOSDEM coordinator, Ludovic.Brenta@insalien.org

Dirk Craeynest, Ada-Belgium & AdaFosdem list owner, Dirk.Craeynest@cs.kuleuven.ac.be

(*) AdaFosdem is an open mailing list (anyone can subscribe), though subscriptions have to be confirmed by e-mail (to avoid spammers subscribing). The list is not moderated, though only subscribers can post (again to reduce the risk of spamming). We hope this setup will be a good compromise between high availability of the list and low spam risk.

[See also "Libre Software Meeting" in AUJ 25-3 (Sep 2004), p.117. --su]

Jun 20-24 - Ada-Europe 2005 Conference

From: Dirk Craeynest <dirk@heli.cs.kuleuven.ac.be>
Date: 7 Oct 2004 22:06:48 +0200
Organization: Ada-Europe, c/o Dept. of Computer Science, K.U.Leuven
Subject: 2nd CFP Conf. Reliable Software Technologies, Ada-Europe 2005
Newsgroups: comp.lang.ada,fr.comp.lang.ada

10th International Conference on Reliable Software Technologies - Ada-Europe 2005

20 - 24 June 2005, York, UK

http://www.ada-europe.org/conference2005.html

Organized, on behalf of Ada-Europe, by the University of York, in cooperation with ACM SIGAda (approval pending)

Ada-Europe organizes annual international conferences since the early 80's. This is the 10th event in the Reliable Software Technologies series, previous ones being held at Montreux, Switzerland ('96), London, UK ('97), Uppsala, Sweden ('98), Santander, Spain

('99), Potsdam, Germany ('00), Leuven, Belgium ('01), Vienna, Austria ('02), Toulouse, France ('03), Palma de Mallorca, Spain ('04).

General Information

The 10th International Conference on Reliable Software Technologies (Ada-Europe 2005) will take place in York, UK. Following the usual style, the conference will span a full week, including a three-day technical program and vendor exhibitions from Tuesday to Thursday, along with parallel workshops and tutorials on Monday and Friday.

Schedule

7 November 2004: Submission of proposed contributions

17 January 2005: Notification to authors

7 March 2005: Camera-ready papers required

20-24 June 2005: Conference

Topics

In the last decade the conference has established itself as an international forum for providers and practitioners of, and researchers into, reliable software technologies. The conference presentations will illustrate current work in the theory and practice of the design, development and maintenance of long-lived, high-quality software systems for a variety of application domains. The program will allow ample time for keynotes, Q&A sessions, panel discussions and social events. Participants will include practitioners and researchers from industry, academia and government organisations interested in furthering the development of reliable software technologies. To mark the completion of the technical work for the Ada language standard revision process, a special session will be devoted to the presentation and discussion of the prospects of the revised language in the landscape of mainstream language technologies.

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

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

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

- Enabling Technology: CASE Tools, Software Development Environments and Project Browsers, Compilers, Debuggers and Runtime Libraries

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

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

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

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

Call for Industrial Presentations

[Cf. the Forthcoming Events section in this AUJ issue, pages 221-222. – su]

Submissions

Authors are invited to submit original contributions. Paper submissions shall be in English, should be complete and should not exceed 20 double-spaced pages in length. Extended abstracts and outlines of presentations that provide a sufficient insight into the intended contents of the contributions will also be considered. Authors of accepted extended abstracts and/or presentations shall submit a consolidated version of their contribution to the Program Co-Chair Tullio Vardanega for final acceptance, by *February 21, 2005*. Authors should submit their work via the Web submission system accessible from the conference Home page. The preferred format for submission is PDF. Postscript can also be accepted, as long as it was generated selecting the "optimize for portability" option in the used printer driver. Submissions by other means and formats will *not* be accepted. If you do not have easy access to the Internet, or you do not have an appropriate Web browser, please contact the Program Co-Chair Tullio Vardanega, whose address details are on this call as well as on the conference Home page.

Proceedings

The authors of accepted papers shall prepare their camera-ready submissions in full conformance with the LNCS style, not exceeding 12 pages and strictly by *March 7, 2005*. Authors should refer to: http://www.springer.de/comp/lncs/authors.html for format and style guidelines. Failure to comply will prevent the paper from appearing in the conference proceedings. The conference proceedings including all accepted papers will be published in the Lecture Notes in Computer Science (LNCS) series by Springer Verlag, which will be available at the start of the conference. All other accepted contributions will appear in the Ada User Journal using the relevant format and style.

Awards

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

Call for Tutorials

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

Call for Workshops

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

Exhibition

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

Reduced Fees for Students

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

Organizing Committee

Conference Chair

Alan Burns, University of York, UK burns@cs.york.ac.uk

Program Co-Chairs

Tullio Vardanega, University of Padua, Italy, tullio.vardanega@math.unipd.it

Andy Wellings, University of York, UK, andy@cs.york.ac.uk

Tutorial Chair

Iain Bate, University of York, UK, ijb@cs.york.ac.uk

Exhibition Chair

Rod Chapman, Praxis Critical Systems, rod.chapman@praxis-cs.co.uk

Publicity Co-Chairs

Ian Broster, University of York, UK, ianb@cs.york.ac.uk

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

Local Administrator

Sue Helliwell, University of York, UK, sue@cs.york.ac.uk

Ada-Europe Conference Liaison
Laurent Pautet, ENST Paris, France, pautet@enst.fr

Program Committee

Asplund Lars, Mälardalens Högskola, Sweden
Alonso Alejandro, Universidad Politecnica de Madrid, Spain
Barnes Janet, Praxis Critical Systems, UK
Bernat Guillem, University of York, UK
Blieberger Johann, Technische Universität Wien, Austria
Burgstaller Bernd, Technische Universität Wien, Austria
Burns Alan, University of York, UK
Cederling Ulf, Vaxjo University, Sweden
Craeynest Dirk, Aubay Belgium & K.U.Leuven, Belgium
Crespo Alfons, Universidad Politecnica de Valencia, Spain
Dencker Peter, Aonix GmbH, Germany
Devillers Raymond, Université Libre de Bruxelles, Belgium
González-Harbour Michael, Universidad de Cantabria, Spain
Hately Andrew, Eurocontrol - CEATS Research Development Simulation Centre, Hungary
Hommel Günter, Technischen Univesität Berlin, Germany
Kauer Stefan, EADS Dornier, Germany
Keller Hubert, Institut für Angewandte Informatik
Kermarrec Yvon, ENST Bretagne, France
Kienzle Jörg, McGill University, Canada
Kordon Fabrice, Université Pierre & Marie Curie, France
Leroy Pascal, IBM, France
LLamosi Albert, Universitat de les Illes Balears, Spain
Lundqvist Kristina, MIT, USA
Mazzanti Franco, Istituto di Scienza e Tecnologie dell'Informazione, Italy
McCormick John, University of Northern Iowa, USA
Miranda Javier, Universidad Las Palmas de Gran Canaria, Spain
Morere Pierre, Aonix, France
de la Puente Juan A., Universidad Politecnica de Madrid, Spain
Pautet Laurent, ENST Paris, France
Plödereder Erhard, Universität Stuttgart, Germany
Romanovsky Alexander, University of Newcastle upon Tyne, UK
Rosen Jean-Pierre, Adalog, France
Schonberg Edmond, New York University & ACT, USA
Vardanega Tullio, Università di Padova, Italy
Wellings Andy, University of York, UK
Winkler Jürgen, Friedrich-Schiller-Universität, Germany

Ada and Education

Real-Time Ada programming

From: Stephane Richard <stephane.richard@verizon.net>
Date: Thu, 16 Sep 2004 11:29:08 GMT
Subject: Re: RTeal-time Ada Programming Rules ?
Newsgroups: comp.lang.ada

Armand Puccetti wrote:

> Does someone know about some good guides with programming rules for real-time Ada? I'm looking for detailed rules used for developing embedded systems within large companies or national authorities (FAA, DoD,...), that can give concrete hints to RT programmers. Also, what reference text books on RT Ada programming are worthwile reading?

Here's one called "Real time scheduling theory and Ada."
http://www.sei.cmu.edu/pub/documents/88.reports/pdf/tr33.88.pdf

And another called "Realtime Software Engineering in Ada: Observations and Guidelines"
http://www.sei.cmu.edu/pub/documents/89.reports/pdf/tr22.89.pdf

I think they'll make to good night reading books for you :-). Might wanna pay a visit to my website too (ada world http://www.adaworld.com)

- In the Learning Center/ Free books and references for another PDF file about the Ravenscar tasking model.

- In the Ada Advocacy section for yet another pdf file entitled "Targeting Ada95/DSA for Distributed Simulation of Multi Protocol Communication Networks"

Free Seminar about SPARK

From: Joyce L. Tokar <tokar@pyrrhusoft.com>
Date: Fri, 27 Aug 2004 00:12:09 GMT
Subject: FREE SEMINAR: SPARK: A Safer Way to Program
Newsgroups: comp.lang.ada

SPARK: A Safer Way to Program

Friday 1 Oct 2004 13:00-16:00

Mustang Library Discussion Room
10101 N 90th St, Scottsdale, AZ 85258

There is an increasing need to produce software that is safe, secure, and reliable. As systems become more complex and safety and security issues gain ground in the world of real-time applications, there is a greater demand for tools and techniques that will enable the generation of robust software in a cost-effective and timely manner. Much of the emphasis on the development of high integrity software is placed on the testing cycle near the end of the development process. This methodology results in the need for a large amount of time at the end of the development process to test components as well as integrated systems.

Yet, it is well known and documented that the best time to find errors is at the start of the development cycle when the problems are smaller and better understood. The cost of correcting errors that are discovered early in the software life cycle is considerably less expensive than those that are found during integration and test. The SPARK approach to developing software provides the tools and technology needed to construct correct software from the start of the process. This methodology reduces the cost of development because the components are built correctly from the start leading to a shorter integration and test cycle with fewer bugs discovered in this phase.

This free seminar will provide an overview of the SPARK language and the tools in the SPARK toolset that facilitate the generation of safe and secure software. The course is suitable for senior software and systems engineers; the seminar does not presume prior knowledge of SPARK. The seminar is also useful to software and systems managers responsible for the development and integration of complex critical systems. The attendees should have an understanding of the fundamentals of the development of complex, critical real-time software applications.

If you would like to join us for a three-hour presentation by Dr. Joyce L Tokar, please complete the registration form on our website and email it to training@pyrrhusoft.com.

See our website for other dates and locations of the seminar www.pyrrhusoft.com.

Public SPARK Training in 2005

From: Rod Chapman <rod.chapman@praxis-cs.co.uk>
Date: 11 Nov 2004 07:42:30 -0800
Subject: ANN: SPARK Training in 2005
Newsgroups: comp.lang.ada

I'm pleased to say that dates for public SPARK Training for 2005 are now available on www.sparkada.com

Note that there's a new, advanced "Black Belt SPARK" course that concentrates on the use of the SPARK Proof Tools and the Proof-Directed design of SPARK programs. Public course dates for this course will be announced soon.

Public Ada 95 Courses

From: Ed Colbert <colbert@abssw.com>
Date: 12 Nov 2004 09:45:37 -0800
Subject: [Announcing] Public Ada 95 Courses 13-17 December 2004 in Carlsbad CA
Newsgroups: comp.lang.ada

Absolute Software will be holding a public Ada 95 course during the week of 13 [December] 2004 in Carlsbad, CA. You can find a full description and registration form on our web-site, www.abssw.com. Click the Public Courses button in the left margin. (We also offer courses on object-oriented methods and other object-oriented languages.)

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

Ada-related Resources

New AdaPower.com

From: David Botton <david@botton.com>
Date: Wed, 27 Oct 2004 19:06:14 -0400
Subject: New AdaPower.com
Newsgroups: comp.lang.ada

I have begun work on a new version of AdaPower.com powered by PHP and MySQL using portal code that runs a number of my personal sites.

When complete, site updates can easily be made from a control panel I wrote, etc. etc.

More information to follow. You can watch progress over the next few weeks as things are put up, etc. at of course http://www.adapower.com

The old AdaPower site is accessible via http://www.adapower.org during the transition.

From: David Botton <david@botton.com>
Date: Thu, 28 Oct 2004 02:27:05 -0400
Subject: AdaPower.com Links
Newsgroups: comp.lang.ada

The AdaPower Links Page is complete now. So....

Are you a Compiler Vendor, Tool Vendor or Offering Training and Consulting?

Then you should be on the AdaPower.com Links page.

http://www.adapower.com/links

Please check to see if you are listed. If not, please e-mail to david@botton.com

From: David Botton <david@botton.com>
Date: Thu, 28 Oct 2004 05:19:24 -0400
Subject: AdaPower.com Call For Projects
Newsgroups: comp.lang.ada

Do you have an active Ada project that other people can join and be a part of?

There are many people that would love to work with you on your Ada dreams.

The *new* AdaPower.com features a section called "Ada Projects You Can Join" where your Ada project can be listed.

See http://www.adapower.com/projects

Contact me: david@botton.com to have your project listed! Please include Name, Description and URL

From: David Botton <david@botton.com>
Date: Fri, 29 Oct 2004 00:53:42 -0400
Subject: AdaPower.com Home Pages
Newsgroups: comp.lang.ada

Is your Ada home page listed on AdaPower.com?

Take a look http://www.adapower.com/index.php?Command=HomePages

If not let me know!

From: David Botton <david@botton.com>
Date: Fri, 29 Oct 2004 00:52:25 -0400
Subject: AdaPower.com RSS News Feeds
Newsgroups: comp.lang.ada

Did you know that the new AdaPower.com has RSS News Feeds?

You can subscribe to either the main news feed at http://www.adapower.com/rss/index.xml or the update feed for the last 20 updates made to the site at http://www.adapower.com/rss/latest/index.xml

From: David Botton <david@botton.com>
Date: Fri, 29 Oct 2004 00:55:24 -0400
Subject: Source Code Example, Articles, Etc.
Newsgroups: comp.lang.ada

Do you have Ada Packages for Reuse that are not yet on AdaPower.com?

Most packages have been entered in to the new AdaPower.com site. Make sure yours is there.

If not, contact me and it will be in instantly.

From: David Botton <david@botton.com>
Date: Tue, 9 Nov 2004 00:24:16 -0500
Subject: The NEW AdaPower.com NOW COMPLETE
Newsgroups: comp.lang.ada

The entire contents of the old AdaPower.com has now been moved in to the databases running the new AdaPower.com, links and code updated when known and already a ton of new stuff in.

From the AdaPower.com home page:

Ada Programming Articles, Examples and Packages on AdaPower as of Nov 8, 2004: 433. So now there is a count :-)

Help me break 500 this week!

I am looking for examples of Ada programing. Be it simple basic code, advanced techniques or sample programs to demonstrate the use of Ada libraries, GUI programming with Ada, Web programming with Ada, etc. etc.

Thanks!

From: David Botton <david@botton.com>
Date: Fri, 5 Nov 2004 01:46:49 -0500
Subject: Ada FAQ
Newsgroups: comp.lang.ada

AdaPower.com is almost done in the transition to the new design and database orientation and I've now picked up on an old project, the Ada FAQ. As we all know, the old Ada FAQs have been kindly locked up in (c)s and the author unresponsive to release them for many years.

AdaPower had an FAQ project for a little while and the FAQs from that project are being imported in to the new Ada FAQ at: http://www.adapower.com/faq

If you have an Ada question and answer, please forward it to David@Botton.com

If you maintain an FAQ for an Ada related site or product, please send a link to the above address so I can add it to the list of "Other Ada Related FAQs"

BTW - On November 7, AdaPower will be 6 YEARS OLD! see http://www.adapower.com

Ada-related Tools

AdaCL 4.2.0 - Ada Class Library

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Thu, 26 Aug 2004 21:29:49 +0200
Subject: [Announcement] AdaCL 4.2.0 released.
Newsgroups: comp.lang.ada

Notes:

First Release with Björn Persson's EAStrings and Orto. It's a Team now so do use Tracker for bug reports.

Changes:

EAStrings support for non Latin 1 strings.

Orto Commandline parsing for non Latin 1 commandlines.

Extension to XML/Ada Unicode support. And yes I do submit them to libre as well and one is already approved.

Abstract:

AdaCL provides library services for scriptig in Ada:

- A text search and replace library.

- a complete cgi binding.

- execution of external programms inclusive I/O redirection with Ada.Text_IO. Unlike GNAT.OS_Lib Ada CL lets you wait on a given asynchronous process instead of just the fi rst to end.

- a garbage collector.

- booch components for indefinite elements.

- trace feature - very handy for CGI (no debugger, no console output).

- Control cdrecord and makeisofs.

- some cvs tools.

With Regards

Martin Krischik

mailto://krischik@users.sourceforge.net

http://www.ada.krischik.com

Simple components 1.6

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sat, 9 Oct 2004 20:02:47 +0200
Subject: ANN: Simple components 1.6
Newsgroups: comp.lang.ada

I have uploaded version 1.6.

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

Changes made are:

1. Added set of GC objects: the child package Object.Handle.Generic_Set;

2. Erase operation was added to all unbounded arrays;

3. Bug fix in Generic_Set. Now, binary operations should work correctly when both arguments are the same set.

Lego Mindstorms

From: Jerry Petrey <jdpetrey@raytheon.com>
Date: Wed, 29 Sep 2004 10:00:10 -0700
Subject: Re: Ada and Robotics (for fun)?
Newsgroups: comp.lang.ada

Chris Humphries wrote:

> Just for fun, I thought it would be interesting to do hobby like ada programming on robots. From searching google, most I got was references to using Ada for lego mindstorms. That seems like fun, anyone have experience in this? Here are some of the links for Lego Mindstorm:
> 1) http://www.faginfamily.net/barry/Papers/AdaLetters.htm
> 2) http://www.usafa.af.mil/dfcs/bios/mcc_html/adagide.html
> 3) http://www.usafa.af.mil/dfcs/Ada_Mindstorms_manual.htm
> [4) http://www.faginfamily.net/barry/Papers/ITICSEWeb/using_ada.htm --su]
> Hope this wasn't off-base with this, but all programmers like to play in programming that isn't "work"? Plus, robots are cool.

Mr. Fagan has done a nice job with Ada Mindstorms. I hope it continues to evolve.

You can get the AdaGide add-on for Ada Mindstorms at:

ftp://ftp.usafa.af.mil/pub/dfcs/fagin/adamindstorms-install.exe

[See also same topic in AUJ 23-3 (Sep 2002), pp.127-128. --su]

ECLAT Status

From: Nick Roberts <nick.roberts@acm.org>
Date: Thu, 07 Oct 2004 23:52:57 +0100
Subject: ECLAT [was: Ada memory management?]
Newsgroups: comp.lang.ada

Luke A. Guest wrote:

>> http://sourceforge.net/projects/eclat

> Erm, how far are you with this project Nick? I've known about it for a while and your page doesn't really have anything on it (yet), how about an update?

Honest truth is, not very far yet. There are a few docs under the 'Docs' link.

> Also, are you using GCC at all? That would be interesting; it might spur some competition with ACT ;-D If you're not using GCC, maybe you should try using it to speed up development.

I do want ECLAT to be (friendly :-) competition for GNAT/GCC, both in the front and and the back end. I think this is a case where it makes sense to 'reinvent the wheel'. If people who are new to Ada ask "Are there any free compilers?", I think it would sound a lot better to be able to suggest two, rather than just one. (I know there is ObjectAda, but it is really only a demo version.)

There are some technical reasons why GNAT and GCC are unsuited to what I want to achieve.

GNAT's library model is based on source code files directly representing the library; this model isn't always ideal. I want to provide a compiler that has the more traditional model of a library being stored in a set of files which contain all the necessary information (to generate executables) in a binary form.

I want ECLAT to be able to target the AdaOS native executable format for the IA-32 (NEAI/IA-32), which is segmented. GCC emits code which is suitable (only) for a 'flat' memory model, and I think adapting it to generate code that supports the NEAI/IA-32 segmented architecture would be difficult.

[See also "ECLAT - Open Source Ada 2005 Compiler" in AUJ 25-3 (Sep 2004), p.120. --su]

Distributed compilation

From: Martin Dowie <martin.dowie@baesystems.com>
Date: Fri, 29 Oct 2004 11:50:05 +0100
Organization: BAE SYSTEMS
Subject: Re: Distributed compiles?
Newsgroups: comp.lang.ada

> A long shot, I know, but are there any distributed compilers for Ada? Our build process takes a couple of hours and it would nice if we could spread the load a bit...

Not quite what you're after but check out the "-j" option for GNAT (under '6.2 switches for gnatmake'). On a twin Xeon machine this should give you 4 compilations in parallel.

From: Jeff Creem <jcreem@yahoo.com>
Date: Fri, 29 Oct 2004 11:05:58 GMT
Subject: Re: Distributed compiles?
Newsgroups: comp.lang.ada

I've never tried this but perhaps cook would be helpful

http://www.vaxxine.com/pegasoft/homes/5.html#5.2

From: Georg Bauhaus <sb463ba@l1-hrz.uni-duisburg.de>
Date: Wed, 3 Nov 2004 13:22:09
Subject: Re: Distributed compiles?
Newsgroups: comp.lang.ada

Not suggesting that you haven't already done so, still, could you split the software into independent subsystems that are built separately and then use some distributed file system?

From: Wes Groleau <groleau+news@freeshell.org>
Date: Wed, 03 Nov 2004 19:48:10 -0500
Subject: Re: Distributed compiles?
Newsgroups: comp.lang.ada

With GNAT, you can specify gnatmake -j 5 and have five compiles running at once.

If you set it up (it's possible, check the docs) so that when it calls 'gcc' it actually gets a script that runs the job on another machine with the real gcc.

And, since gnatmake is written in Ada, and compiled with GNAT, which complies with the distributed systems annex, it shouldn't be too hard for someone with some time on their hands to modify gnatmake as follows:

-j n

where n is a positive number, just like now

-j "host1 host2 host3 host4"

farms out the compiles on all those hosts (assumes mount points and paths are the same)

Programming a PDA with Ada

From: Stephane Richard <stephane.richard@verizon.net>
Date: Tue, 26 Oct 2004 11:54:29 GMT
Subject: Re: PDA's
Newsgroups: comp.lang.ada

> I'm looking for a PDA that I can program. So I can basically put my own buttons and information on.
> Any information on a PDA that does that, or specific software and PDA that I can use, would be great.

As far as programmable PDAs goes, There's more than one model. I've found some programming language or another on pretty much all of Palm's series of PDAs. I have a PSION REVO+ which I happen to like alot because of it's layout and that one is programmable as well in OPL (a somewhat BASIC like language) C++ and Java and of course Assembler.

For palm: http://mobile.eric-poncet.com/palm/tutorial.html

For PSION PDAs you can go to the Symbian OS website http://www.symbian.com/developer/

The good think about a Symbian OS machine, is that Ada can be compiled to Java bytecode and Symbian can/does run Java.

From: Wes Groleau <groleau+news@freeshell.org>
Subject: Re: PDA's
Date: Tue, 26 Oct 2004 22:27:00 -0500
Newsgroups: comp.lang.ada

Ada to J-code to Palm OS has also been done.

From: Rob Veenker <veenker@xs4all.nl>
Date: Fri, 29 Oct 2004 09:12:38 +0200
Subject: Re: PDA's
Newsgroups: comp.lang.ada

Besides JGNAT, which works well on most PDA's (the trick is to get a JVM :-) there now is MGNAT for PDA's as well. There will be a tutorial by Martin Carlisle in the next SigAda conference in Atlanta. I have already successfully ported (well copied and compiled :-) some applications on the iPAQ. It integrates even into the Visual Studio .Net so you can debug the Ada code as well on the emulated PDA. For this, MGNAT generates debug info into the .IL files

There should be an official release soon

From: Pascal Obry <pascal@obry.org>
Date: 26 Oct 2004 19:40:38 +0200
Subject: Re: PDA's
Newsgroups: comp.lang.ada

Stephane Richard wrote:

> The good thing about a Symbian OS machine is that Ada can be compiled to Java bytecode and Symbian can/does run Java.

Do you have a step-by-step doc to do that? I have a P900 and like to program it in Ada. The JGNAT is ok for me, but I have never installed a JVM into a Symbian device. I know that there is some information on the Sony Ericsson Web site but this is far from being a step-by-step approach!

From: Stephane Richard <stephane.richard@verizon.net>
Date: Tue, 26 Oct 2004 17:50:09 GMT
Subject: Re: PDA's
Newsgroups: comp.lang.ada

Maybe this website can help. http://www.wirelessdevnet.com/channels/pda/training/symbian2/

Fuzzy sets for Ada

From: Dmitry A. Kazakov <mailbox@dmitry-kazakov.de>
Date: Sun, 17 Oct 2004 14:19:36 +0200
Subject: ANN: Fuzzy sets for Ada version 3.7
Newsgroups: comp.lang.ada

Fuzzy sets for Ada is distributed under terms of GMGPL. It provides implementations of:

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

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

- Intuitionistic fuzzy sets with the operations on them;

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

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

- Linguistic variables and sets of linguistic variables with operations on them;

- String-oriented I/O is supported.

The version 3.7 is based on the latest version of simple components.

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

[See also same topic in AUJ 24-4 (Dec 2003), p.204. --su]

Infinite number sequences

From: Georg Bauhaus <sb463ba@l1-hrz.uni-duisburg.de>
Date: Wed, 22 Sep 2004 21:15:54 +0000 UTC
Subject: [ANN] "functional" number sequence device
Newsgroups: comp.lang.ada

A small programming device that has helped me make "infinite" sequences based on numbers in Ada is online at http://home.arcor.de/bauhaus/Ada/fada.txt

It is modelled, in ML terms, after datatype chain = Link of (int (int -> chain)).

Ada and OpenGL

From: Etienne Baudin <pfox@free.fr>
Date: Sun, 17 Oct 2004 17:30:17 +0200
Subject: Exemple d'application Ada+Opengl
Newsgroups: fr.comp.lang.ada

[Translation from French – su] I am offering code that implements a sort of rudimentary viewer for 3d objects. It is not really professional, but it shows that one can put together nice toys with Ada. http://pfox.free.fr/AdaOpenGL-Exemple.tar.bz2

The software compiles on Linux as well as on Windows (good live portability!).

Do not hesitate to let me have your comments and to report any compilation problems (cf. the relevant procedure within the distribution).

Win_IO 2.0.1

From: Michael Gonzalez <mgh@unican.es>
Date: Wed, 13 Oct 2004 11:50:46 +0200
Organization: Universidad de Cantabria
Subject: Win_IO 2.0.1 is now available
Newsgroups: comp.lang.ada

Version 2.0.1 of Win_IO is now available in: http://www.ctr.unican.es/win_io/

This version is developed with GtkAda 2.2.X and is free software. It is the same as version 2.0 except that it corrects a bug in the function used to read an image from disk.

Win_IO is a set of packages for graphical input and output. It is designed specially for students or Ada users who do not want to spend their time learning a complex graphical user interface, but who are "tired" of the old-fashioned text-oriented input and output. Win_IO has the same goals as JEWL (John English Windows Library, http://www.it.bton.ac.uk/staff/je/jewl/), but is simpler (and less powerful) and is portable within UNIX, Linux and Windows platforms. JEWL is currently only provided for Windows.

See how you can code nice-looking windows for input & output in a really minimal program:

output:
http://www.ctr.unican.es/win_io/ow_example.html

input :
http://www.ctr.unican.es/win_io/iw_example.html

Win_IO is composed of the following modules:

- Input_Windows: Provides a simple window with I/O capabilities for data of the types Integer, Float, and String. Several data can be displayed and/or retrieved on the same window.

- Output_Windows: Provides a simple window with Output capabilities for data of the types Integer, Float, and String. Several data can be displayed on the same window.

- Message_Windows: Provides a simple window for displaying a short message. It provides an OK button for closing the window.

- Menu_Windows: Provides a simple window with several buttons that enable the user to select from a number of options.It is a generic package that must be instantiated with an enumeration type. One button will be created for each value in this type.

- Graphics_Windows: Provides a simple window with drawing capabilities, including the display of picture files.

- Plot_Windows: Provides a simple window for drawing two-dimensional graphs from sets of points.

A community Windows binding

From: Stephen Leake <stephen_leake@acm.org>
Date: 05 Oct 2004 19:28:03 -0400
Subject: A community Windows binding
Newsgroups: comp.lang.ada

I'd like to step back from the "GWindows vs CLAW" debate and take stock.

David Botton has made several points in favor of using GWindows as the starting point for a community Ada binding. He has also pointed out the existence of a GWindows user/developer group on the GNAVI mailing list; perhaps those people have not been represented in the discussions here.

My personal interest is in establishing a community supported Free Software Windows Ada binding. I can easily go with either GWindows or CLAW as a starting point.

I think the crucial issue is "how many developers will actually join the project?".

I have volunteered to be the first sysadmin of a SourceForge project for a community Ada Windows binding. I'd like to get started :).

[...] As the list stands now, there is a clear preference for GWindows.

From: Frank Piron
Date: Thu, 07 Oct 2004 08:38:30 +0200
Subject: Re: A community Windows binding
Newsgroups: comp.lang.ada

Our position here at KonAd:

1) Every effort which leads to a broader use of Ada should be supported.

2) Building a library or binding which for easy construction of production quality GUI interfaces on the windows operating system is an effort in the sense of 1).

3) Last year we decided to extend GWindows because we had to decide something (otherwise no money) and GWindows was there: free, available, easy to use and to extend.

4) Of course we will participate in every community effort concerning the further development of GWindows.

From: Steve <steved94@comcast.net>
Date: Wed, 06 Oct 2004 14:04:04 GMT
Subject: Re: A community Windows binding
Newsgroups: comp.lang.ada

Georg Bauhaus wrote:

> I think the point is Windows programming, not general GUI programming. There are things that make OSs and GUIs look and feel different. In particluar, if your program uses Windows specific features, how do you define an interface that uses Windows specific features on *BSD/X, say?
> How would you define an interface that is capable of being ported to _both_ Windows and 8 1/2 (the Plan 9 graphical interface)?

IMHO,

The community Windows binding should not get bogged down in building a system that is compatible with other OS or windowing systems.

I am a strong advocate of Ada. Ada is the best general purpose programming language I have met to date. I have met several programming languages to date, and continue to meet them (C# being the latest).

I have come to the realization about making the most economic and logical choices: it doesn't matter.

When OS/2 was a more stable and uniform OS, the bulk of new development went to Windows and NT, which at the time was comparatively unstable. OS/2 was a better choice... it doesn't matter.

Using Ada is the best choice for much of the development done today. It doesn't matter, development is rarely done in Ada. I think that's why David Botton has been away from this list for a long period of time.

For the community windows binding, I think it's best that we focus on Windows and only Windows (even though it sucks).

From: Ross Higson <rosshigson@optusnet.com.au>
Subject: Re: A community Windows binding
Newsgroups: comp.lang.ada

[...] I wish there was a good platform independent GUI toolset, and I support the goals of GtkAda in trying to fill this gap even though I personally find it a bit limited and more difficult to use than GWindows. But I thought we were talking here about a Windows specific binding, and I think GWindows is the right choice for that. It would also help promote the use of Ada on the Windows platform because it would be familiar territory to any Windows programmers who decided to try Ada.

From: Stephen Leake <stephen_leake@acm.org>
Date: 10 Oct 2004 14:05:57 -0400
Subject: Re: A community Windows binding
Newsgroups: comp.lang.ada

Given the response so far, it is clear there is much more support for using GWindows as the base of a "community Windows binding" than CLAW.

Also, it seems clear that SourceForge is the platform to use. Using Gnu Arch is appealing, but it requires that each developer open their machines to outside access. I'd rather let SourceForge worry about the security implications of that. The other hosts suggested don't seem as well supported as SourceForge.

Before I actually create a SourceForge project, I'd like to get at least one other developer to commit to being a sysadmin on the project. I don't think being sysadmin will take much time; it mostly consists of setting up accounts for developers. When we get to doing releases, there will be more work, but we can deal with that then. [...]

From: David Botton <david@botton.com>
Subject: Re: A community Windows binding
Date: Sun, 10 Oct 2004 20:39:31 -0400
Newsgroups: comp.lang.ada

Stephen Leake wrote:

> Another minor issue is the name of the project. We want to host GWindows code (no need to change the name of that package).

I recommend we use the original name of the GWindows/GNATCOM project, ie. GNAVI

> Also the Gnatcom code? I'm not clear if this is available anywhere other than AdaPower, or if it needs a CVS project.

CVS for GNATCOM and GWindows was actively maintained by ACT when they were supporting it for their customers. As

far as I know this is not officially the case any more.

> And possibly a version of Ada.Collections, until they become widely available from vendors. We should start using that for any collection needs in GWindows. Or we could just redistribute a .tar.gz of Matthew Heaney's current stuff.

I think just linking / redistributing Matthew Heaney's current stuff is ideal.

Comparison of GUI libraries for Ada

From: David Botton <david@botton.com>
Date: Sun, 3 Oct 2004 12:32:56 -0400
Subject: Re: GWindows and David Botton
Newsgroups: comp.lang.ada

[Discussing the development of a community Windows binding (see above thread) --su]

Stephen Leake wrote:

> We should compare Windex, Gwindows, and Claw, and decide which one makes the best starting point.

I have a comparison between GWindows and CLAW at:
http://www.adapower.com/gwindows/GWindowsVsClaw.html

Stephen, I would suggest looking at the event models in GWindows in particular to see the major reason (in addition to doing what was needed to allow for COM/.Net/ActiveX controls which may be the chief reason in the end) why I didn't build off of Windex which is very tightly bound to the Windows way of doing things. Windex while in general a smaller library than GWindows does have some areas that it is better fleshed out in.

GWindows is also tuned toward the larger project I was working on for a GNU Delphi replacement. (GWindows + GNATCOM + GNAVI IDE). So it is very easy to target for a GUI builder based on its dispatching model combined with inheritance.

From: Marius Amado Alves <amado.alves@netcabo.pt>
Date: Wed, 06 Oct 2004 19:56:55 +0100
Subject: Re: GWindows and David Botton
Newsgroups: comp.lang.ada

An update of our little table incorporating the latest contributions. I have also put it in the Universal Casbah wiki at

http://www.liacc.up.pt/~maa / Casbah / Miscellany / GUIs_
For_Ada or
http://www.liacc.up.pt/~maa/cgi-bin/casbah/casbah.cgi?operation=view&pagename=GUIs_For_Ada so that contributions can go there directly.

GUI/Windows products for Ada comparison
(C) Guys de Cla

Product	Binding	License/OSes/Install			Support/Remarks
AdaBindX	C/Lesstif	GPL*	B,L	d	b
IntroClaw	C/Win32	GPL*	W	easy free	f
FullClaw	C/Win32	prop	W	easy com	f
Glade	GTK+	GPL	B,L,?	d	a
GtkAda	C/GTK+	GPL*	B,L,M, S,W	d com	
GWindows	C/Win32	GPL*	W		
JEWL	Win32	free	W	easy	
RAPID					c
TASH	Tcl/Tk	GPL*	B,W, L	e	
Windex	Win32	GPL*	W	free	
Win32	C/Win32	free	W		

GPL* = GMGPL
(a) visual GUI builder and code generator (gate) for GtkAda
(b) binding to Xlib, Xt, and Xm (motif/lesstif)
(c) visual GUI builder
(d) Debian package
(e) Debian package no longer updated
(f) visual GUI builder available

Binding:
Underlying programming language / library.

License:
prop=proprietary

OSes:
B = BSD
L = Linux
M = MacOS
S = Solaris
W = Windows

Install:
easy = installs out of the box
hard = requires a guru

Support:
free = dedicated volunteer structure e.g. a maillist (not CLA)
com = commercial support
no = no support (but of course CLA continues to exist)

Append "/doc" if documentation exists (free)

Threads in comp.lang.ada *

GTK
GWindows and David Botton

URLs *

RAPID
ftp://sunsite.informatik.rwth-aachen.de/pub/mirror/ftp.usafa.af.mil/pub/dfcs/carlisle/usafa/rapid/index.html

XIA - XPath In Ada

From: Marc A. Criley <mc@mckae.com>
Date: Sat, 06 Nov 2004 15:22:19 GMT
Subject: Announce: XIA (XPath In Ada)
Newsgroups: comp.lang.ada

A limited capability beta version of XIA, a native implementation of XPath In Ada, is now available on the Mckae Technologies website (http://www.mckae.com/xia.html).

Version: 0.10

This current version of XIA is a limited-capability beta release. It implements the axes and the node tests, but does not yet implement any predicate filtering. So an XPath query returns all the nodes meeting the node test along the given axes. Predicate processing will be incrementally added to XIA in subsequent releases, but right now any additional node filtering must be done by the XML-enabled application itself.

As this is a beta release, reports of errors (either in operation or in the nodes retrieved) would be appreciated. Please provide the XML document (or readable fragment), the query that was submitted, and a description of what was expected.

Send bug reports and questions to xia-info@mckae.com

DOM and SAX parsing in Ada

From: Steve <steved94@comcast.net>
Date: Tue, 09 Nov 2004 03:14:13 GMT
Subject: Re: DOM and SAX parsing in Ada
Newsgroups: comp.lang.ada

Tim Roede wrote:

> There is considerable interest in the group I am working with to provide XML parsing using both DOM and SAX parsing. Does anyone have a name or a link to a suitable Ada library (similar to xerces)?

XML/Ada

http://libre.act-europe.fr/xmlada/

Includes a Unicode module, a SAX 2.0 module, and a DOM 2.0 module. The code is released under the GMGPL.

Support is available for a fee from AdaCore. In addition to the platforms listed at the Libre site, I have successfully used XML/Ada 1.0 with ObjectAda 7.2.2.

From: David Botton <david@botton.com>
Newsgroups: comp.lang.ada
Date: Mon, 8 Nov 2004 19:24:41 -0500
Subject: Re: DOM and SAX parsing in Ada

There are 2 XML libs listed here and I will be adding another shortly:

See:
http://www.adapower.com/index.php?Command=Class&ClassID=AdaLibs

See: The Ada Source Code treasury on http://www.adapower.com for an example of using the Ada/XML DOM (more examples to follow)

From: David Botton <david@botton.com>
Date: Mon, 8 Nov 2004 19:56:59 -0500
Subject: Re: DOM and SAX parsing in Ada
Newsgroups: comp.lang.ada

There are so many XML and Web related tools for Ada that I broke them now in to their own category, so please see http://www.adapower.com/reuse

From: David Botton <david@botton.com>
Date: Tue, 9 Nov 2004 08:25:32 -0500
Subject: Re: DOM and SAX parsing in Ada
Newsgroups: comp.lang.ada

> AdaCL.GCI is missing on "Ada Web and XML"

Not any more.

Ada Binding to Pipes

From: Marc A. Criley <mc@mckae.com>
Date: Tue, 10 Aug 2004 07:58:58 -0500
Subject: Re: SSH Sessions?
Newsgroups: comp.lang.ada

Adam Ruth wrote:

> I'm doing this very thing myself, and I'm just using the command line version of ssh. I execute it using the system call (the c library system (`const char *command`) call). It's pretty straightforward, since I don't need to do much that's complicated, just execute a remote command line program as though it was being executed on my local command line.

Just want to reacquaint everyone about the existence of Jim Rogers' "Ada Binding to Pipes" (http://www.adapower.com/reuse/pipes.html). You can issue a command and all the output is caught and easily read back in for post-processing.

It's incredibly easy to use, effective and reliable, and I use it almost exclusively whenever I need to issue commands from within an Ada program. Works great, highly, highly recommended. (Thanks Jim!)

PGSQL - PostgreSQL Minimal Binding

From: "Alex R. Mosteo" <devnull@mailinator.com>
Newsgroups: comp.lang.ada
Subject: Ada & Postgresql
Date: Fri, 03 Sep 2004 19:09:58 +0200

Anyone doing the above? I've found three bindings:

Apq, Pgada, Gnade

I suppose I could test the three, but has anyone already done that? I just need to do fairly simple insertions.

From: Marius Amado Alves <amado.alves@netcabo.pt>
Newsgroups: comp.lang.ada
Subject: Re: Ada & Postgresql
Date: Sat, 04 Sep 2004 12:41:09 +0100

>>Pgada

> Is this the pg2-stuff from Mario Amado Alve? If so, I use this[0] ;) [0] http://adi.thur.de/?show=adabill

Pgada /= Pg2. Nice to know Pg2 is being used. Let me take the chance make a long overdue update on this. On 2001 I've replaced Pg2 with Pgsql, for use in the European project SolEuNet, which ended 2003. I now make Pgsql available, at http://www.liacc.up.pt/~maa/files.

There's no need to go rushing upgrading your existing Pg2 application to Pgsql. But if you're starting a new application, I recommend Pgsql.

Pg2 was in my Adlib site which went down around that time. I'll try to set up a new shop soon with all the updated stuff as well as an historical copy of Adlib.

From: Marius Amado Alves <amado.alves@netcabo.pt>
Newsgroups: comp.lang.ada
Subject: Re: Ada & Postgresql
Date: Mon, 06 Sep 2004 12:36:09 +0100

GNADE vs. Pgsql.

Pgsql is a simple Ada binding to Postgres. GNADE is a very big [1] Ada library encompassing various database systems (Postgres, MySQL, etc.) and protocols (ODBC). Pick the proper thing for your application. Use Pgsql when you simply want to bind to Postgres and don't need ODBC.

Now comparing oranges with oranges, i.e. considering the Postgres binding of GNADE, Gnu.Db.Postgres, vs. Pgsql.

As noted, Gnu.Db.Postgres has a leak in the connection area. But has controlled queries that automatically reclaim memory on finalization. Pgsql has no leaks per se, but in its current version the types are not controlled, so the user has to reclaim memory him self. Pick you poison.

However, I think the main difference is in the style of the Ada spec. Unfortunately I'm not able to look at Gnu.Db.Postgres right now [2], but from what I recall from my examination of the landscape some time ago, which has probably included Gnu.Db.Postgres, they are much longer and harder to use than Pgsql.

[1] The RPM has 6M, but I don't know what's in it. See note 2.

[2] The Sourceforge site seems to only provide RPM files. How do I read this in my Windows XP laptop? Also, when I click the version number on the site (hoping to browse the files), I get XML garbage.

Mine Detector Game 4.4

From: PragmAda Software Engineering <pragmada@earthlink.net>
Date: Sun, 26 Sep 2004 18:40:31 GMT
Subject: Ann: New release of Mine Detector for Windows
Newsgroups: comp.lang.ada

PragmAda Software Engineering is proud to announce Version 4.4 of Mine Detector. This version appears to correct the errors that occurred with Version 4.3 under Windows 2000 and XP, but not under 9X, ME, and Linux.

The Windows binary and source versions are Version 4.4. The Linux binary version remains at Version 4.3, since there is no difference in appearance or behavior between the two versions under Linux.

Mine Detector is available from http://home.earthlink.net/~jrcarter010/mindet.html

War in Ada (the card game)

From: John B. Matthews <jmatthews@wright.edu>
Date: Thu, 23 Sep 2004 03:19:02 GMT
Subject: Re: best ada integrated development environment
Newsgroups: comp.lang.ada

Phil wrote:

> Hi, I have been out of the ada stuff for a while and are just about to embark on a project to simulate a blackjack card game using Ada.

[...]

You might also be interested in my simulation of war (the card game) in Ada: <http://www.wright.edu/~john.matthews/war.html>.

Documentation Tools for Ada Sources

From: Alex R. Mosteo <devnull@mailinator.com>
Date: Thu, 07 Oct 2004 12:40:55 +0200
Subject: Javadoc-like for Ada
Newsgroups: comp.lang.ada

I'm interested in some doc generation tool à-la javadoc; i.e. that not only creates a listing like gnathtml but also capable of extracting explanations from properly formatted comments in the specification.

I've heard good things about doxygen and I'm going to review it to see if it's suitable for Ada.

Any other options I should consider?

From: Stephane Richard <stephane.richard@verizon.net>
Date: Thu, 07 Oct 2004 11:46:16 GMT
Subject: Re: Javadoc-like for Ada
Newsgroups: comp.lang.ada

You might want to look at Ada Browse which you can find here http://home.tiscalinet.ch/t_wolf/tw/ada95/adabrowse/

You might also want to look at AdaDoc which you can find here. http://sourceforge.net/projects/adadoc/

From: Lionel Draghi <Lionel.Draghi@Ada-France.org>
Date: Tue, 12 Oct 2004 00:53:16 +0200
Subject: Re: Javadoc-like for Ada
Newsgroups: comp.lang.ada

http://www.naturaldocs.org/ does require neither special tags in the code, nor ASIS. Building a doc from your sources, or whatever file that can be displayed in a browser is easy.

NaturalDocs is far from being as good as gnathtml/AdaDoc/AdaBrowse to extract procedure parameters and so on, but as this is clearly one of the most uninteresting thing to put in a doc, I don't care. I can use Emacs or GPS for this kind of navigation.

On the other hand, NaturalDocs is very good at extracting explanations from comments, and this seems generally much more interesting to me.

Campaign to get full language support in Natural Docs

From: Martin Dowie <martin.dowie@baesystems.com>
Date: Thu, 21 Oct 2004 12:50:04 +0100
Organization: BAE SYSTEMS
Subject: Re: NaturalDocs - sponsor Ada support
Newsgroups: comp.lang.ada

Martin Dowie wrote:

> Put your $10 where your mouth is and get Ada support at www.naturaldocs.org http://www.naturaldocs.org/languages.html

Actually, even if you don't want to pay, you could consider voting for Ada support for free!

[At this moment, Ada would be the second in the ranking for full support, only behind C/C++ and before languages like Java, PHP or Python --su]

ASIS2XML

From: Simon Wright <simon@pushface.org>
Date: 24 Oct 2004 08:05:42 +0100
Subject: ANN: ASIS2XML 20041024a
Newsgroups: comp.lang.ada

ASIS2XML converts a unit's ASIS representation into XML, so as to make it easier to develop transformational tools using (for example) XSLT.

As supplied, it relies on GNAT; the only ASIS-for-GNAT feature it relies on is that Data_Decomposition.Size has been extended to work for Subtype Indications; and that only so that it can work out how many bytes a record component will occupy when streamed (this part is in progress)

Not every ASIS feature is supported yet.

There is no XML Schema as yet (however, the output's structure follows that of ASIS as determined from the Ada specs -- I'm not at all sure this is the Right Thing for an XML representation).

This is an alpha release, and you can find it at http://www.pushface.org/asis2xml/

Cheddar - Real-Time Scheduling Simulator

From: Frank Singhoff <singhoff@beru.univ-brest.fr>
Date: 8 Sep 2004 15:05:11 GMT
Organization: Universite de Bretagne Occidentale
Subject: ANN : New release of Cheddar : a real time scheduling simulator
Newsgroups: comp.lang.ada

The EA 2215 team is pleased to announce a new release of Cheddar.

Cheddar is a free real time scheduling tool. Cheddar is designed for checking task temporal constraints and buffer sizes of a real time application/system. It can also help you for quick prototyping of real time schedulers. Finally, it can be used for educational purposes.

Cheddar is developed and maintained by the EA 2215 Team, University of Brest.

Cheddar is composed of two independent parts: an editor used to describe a real time application/system, and a framework. The editor allows you to describe systems composed of several processors which own tasks, shared resources, buffers and which exchange messages. The framework includes many feasibility tests and simulation tools. Feasibility tests can be applied to check that task response times are met and that buffers have bounded size. When feasibility tests can not be applied, the studied application can be analyzed with scheduling and buffer simulations. Cheddar provides a way to quickly define "user-defined schedulers" to model scheduling of ad-hoc applications/systems (ex : ARINC 653).

Cheddar is written in Ada. The graphical editor is made with GtkAda. Cheddar runs on Solaris, Linux and Win32 boxes and should run on every GNAT/GtkAda supported platforms

The current release is now 1.3p3. If you are a regular Cheddar's user, we strongly advice you to switch to the 1.3p3 release due to the large amount of 1.3p2 bugs that we fixed.

Cheddar is distributed under the GNU GPL license. It's a free software, and you are welcome to redistribute it under certain conditions; See the GNU General Public License for details. Source code, binaries and documentations can be freely downloaded from http://beru.univ-brest.fr/~singhoff/cheddar

1) Summary of features:

- Do scheduling simulations with classical real time schedulers (Rate Monotonic, Deadline Monotonic, Least Laxity First, Earliest Deadline First, POSIX queuing policies: SCHED_OTHERS, SCHED_FIFO and SCHED_RR) with different type of tasks (aperiodic, periodic, task activated with a Poisson process law, ...)

- Extract information from scheduling simulation. (buffer utilization factor, task response times, task missed deadlines, number of pre-emption, ...)

- Apply feasibility tests on tasks and buffers (without scheduling simulation) :

  + Compute task response time bounds.

  + Apply processor utilization tests.

  + Compute bound on buffer size (when buffers are shared by periodic tasks)

- Shared resources support (scheduling and blocking time analysis). Supported protocols: PIP, PCP.

- Tools to express and do simulations/feasibility tests with task precedence:

  + Schedule tasks according to task precedence

  + Compute Tindell end to end response time.

  + Apply Chetto and Blazewicz algorithms.

- Tools to run scheduling simulation in the case of multiprocessors systems

- Do simulation when tasks are randomly activated.

- Can run scheduling simulation on user-defined scheduler and task arrival patterns.

- Run user-defined analysis on scheduling simulation.

- ...

2) Most of new features provided by 1.3p3:

- Fix many bugs of the previous release (see BUGS file)

- Add a new user interface of the scheduling simulation service. With 1.3p3, Cheddar provides two different scheduling simulations: customized or un-customized scheduling.

  + Un-customized simulation draws time line and computes worst case response time from simulation. This service is called from the "Scheduling Simulation" pixmap.

  + Customized simulation draws time line and can compute many others measures (eg. Worst/Best/Average cases of shared resource blocking and response time from simulation).

This service is called from the menu "Tools/Scheduling/Scheduling simulation" (F. Singhoff)

- Add a way to display or export event tables produced by the scheduling simulator engine. Event tables are XML formatted. An event table is a set of data

which stores a computed scheduling. (F. Singhoff)

- Add a way to import event tables computed by other tools. This service allows you to run analysis on scheduling produced by operating system, object request broker or any applications. (F. Singhoff).

- Add Partitioning tools for multiprocessor systems scheduled with Rate Monotonic. Several partitioning strategies are provided (RM Best Fit, RM Next Fit, RM First Fit, RMGT and RMST) (M. Nivala)

- Fix errors on utilization factor feasibility tests. In the previous release, preemptive EDF and RM tests were applied by error on other schedulers. (H. Martin, S. Bothorel)

- Add user-defined event analyzers. User-defined analyzer can be run on a given scheduling to look for user specific properties. User-defined event analyzers are pieces of user code which scan and do analysis on event tables. (F. Singhoff)

- Add user-defined task arrival pattern. This feature should allow us to easily define new task activation patterns (ex: bursty task activation; jitter constraint activation, sporadic activation,...) (H. Huopana, F. Singhoff)

- Add a simple message scheduling. Actually, message scheduling is limited with constant communication delay messages and with sending tasks which send messages at the start of their activation. This service have to be extended in the next release to be really usefull. (G. Oliva, F. Charlet)

- Add a sub-program To detect priority inversion from scheduling simulation (F. Singhoff)

- Add a C interface to call the framework from C programs. (F. Singhoff)

- Shared resource states are displayed on the time line. (E. Vilain)

3) Work in progress:

During the next year, we plan to improve the tool with the following features:

- Update the user's guide according to the new 1.3p3 features

- Improvement of the buffer analysis features with queuing theory analysis tools.

- Provide a way to import/export application specifications in AADL.

- Improvement of message scheduling with:

+ Allowing message sending at any time of a task capacity

+ Providing a way to user-defined message delay communication by specification of user-defined message scheduling (as user-defined scheduler)

- Fixing a buggy service which should detects deadlock from simulation.

- Completing available services on event tables.

[See also same topic in AUJ 24-4 (Dec 2003), p.207. --su]

RTLGnat - GNAT Port for RT-Linux

From: Maurizio Ferracini <maurizio.ferracini@gmail.com>
Date: 19 Oct 2004 03:21:13 -0700
Subject: Re: Gnat and priority level
Newsgroups: comp.lang.ada

[Getting more than 32 priorities in Windows2000 --su]

> You can't (at least easily). Not only the GNAT implementation uses just 32 priorities, but these are mapped to underlying windows priorities, which are only 7, if I remember correctly (yes, there's overlapping).

And with a Real-Time Linux version like RT-Linux? [...]

From: Jerome Hugues <hugues@nephilim-pouet.enst.fr>
Date: Tue, 19 Oct 2004 11:50:15
Organization: ENST, France
Subject: Re: Gnat and priority level
Newsgroups: comp.lang.ada

Alex R. Mosteo wrote:

> mferracini wrote:
> Firstly, I think that GNAT doesn't target the RT capabilities of RT-Linux, so you'll really have regular non-RT linux threads. Unless someone has a patch for it.

There exists one project on this subject: http://bernia.upv.es/rtportal/apps/rtl-gnat/

Note that I've never tried it, so I cannot comment on which GNAT version it uses, ACATS results, etc

> Second, you can choose two threading implementations with Gnat for Linux: linux native threads, whose particularities I don't know but some Linux expert could tell you (BTW, using the old 2.4 thread implementation, not the new 2.6 one). fsu-threads (I think) which if I remember correctly are POSIX-compliant or something-else-compliant which native Linux's aren't.

Strict Annex D (real time) compliance. See http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/Choosing-between-Native-and-FSU-Threads-Libraries.html

for more details

About VAD - Visual Ada Developer

From: Stephane Richard <stephane.richard@verizon.net>
Date: Wed, 08 Sep 2004 17:29:12 GMT
Subject: About Visual Ada Developer (VAD).
Newsgroups: comp.lang.ada

I've been trying to find the author of this project (in the subject line) to see what's going on with the project. The link to the project doesn't work right now and I'd like to know either where it is, or if it is still actively developed somewhere. Anyone have any information on this?

If the author of VAD, Leonid Dulman, should happen to read this, please get in touch with me either here or by email (you should have my email address already :-). And let's talk.

From: Stephane Richard <stephane.richard@verizon.net>
Date: Thu, 09 Sep 2004 12:00:42 GMT
Subject: For those that wondered about V.A.D.
Newsgroups: comp.lang.ada

Big announcement, lots to read, but I think it's well worth it :-). I'm posting this from a request I got from Leonid himself he can't post to comp.lang.ada for some reason and asked me to post this for him:

Visual Ada Developer (VAD)

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 6.4 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/9x, Unix(Linux), Mac and OS/2. You may use it as IDE for any Ada-95 (C, C++, TCL) project. You may use it to build TCL script only. VAD generated ada sources you may compile and build executable with GNAT in Windows and Unix(Linux) or Aonix ObjectAda 7.2 in Windows.

2. Used software

GNAT 3.15p (3.4.0) Ada-95 compiler ftp://cs.nyu.edu/pub/gnat

TCL/TK 8.3.5.1 http://tcl.activestate.com/software/tcltk/

TCL/TK 8.4.7.0 http://tcl.activestate.com/software/tcltk/

TCL/TK 8.5.b2 http://tcl.activestate.com/software/tcltk/

W A R N I N G ! VAD 6.4 has three realization for tcl/tk8.3.5, tcl/tk 8.4.6 and tcl/tk 8.5x , you need to install and to check tcl/tk before using of VAD.

[For a complete (and long) list of used components and a description of the program files, see the original posting from Stephane Richard at comp.lang.ada --su]

In WEB Browser you may run VAD Tutorial from vad/tutorial/vadtutor.htm VAD 6.4 is available in http://www.websamba.com/GUIBUILDER

You may dowload sources vad64scr.tar.bz2, vadhlp.tar.bz2, vadtutor.tar.bz2, vadsmp.tar.bz2,vadaonix, adahlp.tar.bz2, vadtcl.tar.bz2, adastyle.tar.bz2,philosofers.tar.bz2,vadidl.tar.bz2 and binaries vad64win.tar.bz2 (Windows 9x/NT) vad64lin.tar.bz2 (i386)

Any questions, any ideas, any problems, any help Leonid Dulman(leonid_dulman@yahoo.co.uk)

[See also "VAD 6.2 - Visual Ada Developer" in AUJ 24-4 (Dec 2003), pp.204-205. --su]

IDE for Ada on Mac OS X

From: Stephane Richard <stephane.richard@verizon.net>
Date: Sat, 30 Oct 2004 02:43:44 GMT
Subject: Re: IDE for Ada on Mac OS X?
Newsgroups: comp.lang.ada

Christopher J. Henrich wrote:

> What integrated development environments that run on Mac OS X 10.can handle Ada? Does such a thing exist?

http://macada.org/ is probably the best place to start your search. Not sure if there's an IDE right there but there's one on the main picture so perhaps there they could help you find a Mac IDE.

From: James E. Hopper <hopperj@macconnect.com>
Date: Sun, 31 Oct 2004 01:42:47 GMT
Subject: Re: IDE for Ada on Mac OS X?
Newsgroups: comp.lang.ada

In addition to Apple's Xcode (which works very nicely with Ada) both Codeforge and VisualSlickEdit works on OS X and supports Ada. Oh, and of course Ada mode in emacs is also supported on OS X.

CodeForge http://www.codeforge.com/products/

VisualSlickEdit http://www.slickedit.com/purchase/pu_regtrial.php.

From: John B. Matthews <jmatthews@wright.edu>
Date: Tue, 02 Nov 2004 14:38:44 GMT
Subject: Re: IDE for Ada on Mac OS X?
Newsgroups: comp.lang.ada

Marius Amado Alves wrote:

> According to this you can use Xcode: http://pyrite.truman.edu/~millerj/AdaonOSX/
> [...]

Well, several of the contributors, posting in this thread, have been using Ada on Mac since 1994. The present incarnation for Mac OS X does indeed run with Xcode, as shown on Dr. Miller's site above. The integration is excellent: edit, compile, debug, step, trace, breakpoint, profile - all from the GUI. I also use gnatmake from the command line regularly. [...]

Auto_Text_IO & SAL

From: Stephen Leake <stephen_leake@acm.org>
Date: 07 Aug 2004 12:22:56 -0400
Subject: new SAL, Auto_Text_IO releases
Newsgroups: comp.lang.ada

SAL version 1.70 is released.

Auto_Text_IO 3.03 is released.

SAL is my Ada library; it has containers, spacecraft math, config files, other misc stuff. New in this release is a small collection of GtkAda widgets, some of which work :). More importantly, there is a test harness for automating GUI tests. Also misc fixes in other SAL packages.

Auto_Text_IO is an ASIS-based tool that generates Text_IO packages for types in Ada packages; very useful for persistent storage. New in this release are support for Standard.Duration, Standard.Character, misc other fixes.

For full information, and to download the latest versions, see my web page http://www.toadmail.com/~ada_wizard/

[See also same topic in AUJ 25-3 (Sep 2004), p.119. --su]

XAda - A Simple OS in Ada

From: xavier <xavier@ipnnarval.in2p3.fr>
Date: Wed, 25 Aug 2004 13:46:35 +0200
Subject: [annonce] XAda un os simple en Ada
Newsgroups: fr.comp.lang.ada

[Translated from French – su]

I have started to adapt to Ada the series of articles on the Linux Magazine « Do it yourself » and this has given rise to XAda (short for uniXAda).

XAda is a utility built for acquainting oneself with the internals of a kernel and for building them in Ada. This exercise has already provided [me] with contracts for a more substantial and longer-term project: "Lovelace" a UNIX written in Ada, which natively provides the Ada runtime (among other initial objectives).

Should that be of your interest: http://ipnnarval.in2p3.fr/~xavier/

All sorts of contributions and comments are most welcome.

Ada-related Products

AdaCore - Ada Answers

From: Ed Falis <falis@verizon.net>
Date: Tue, 26 Oct 2004 21:28:01 GMT
Subject: Ada Answers
Newsgroups: comp.lang.ada

www.ada-answers.com launches!

As the need for robust and reliable software systems increases, Ada continues to prove to be an excellent answer for many of today's most complex programming challenges. "Ada Answers" is dedicated to keeping developers and project managers informed about Ada and showcasing the particular strengths and benefits of this extremely powerful programming language. "Ada Answers" is part of AdaCore's ongoing commitment to promote the qualities of the Ada programming language.

Features of the new site include:

- Real world examples of Ada in use, including a comprehensive list of companies and organizations that are using Ada everyday.

- A growing collection of video interviews in which developers and managers speak about why they have chosen Ada.

- Technical features of the Ada programming language are explained, showing how they translate into bottom-line business benefits.

- A section dedicated to the wide range of up-to-date Ada materials and resources available on the web.

AdaCore is always looking for interesting and innovative Ada stories. Please contact us with yours. We plan to produce videos of some of the stories, but all contributors whose story is published will receive a stylish Ada Answers T-shirt!

AdaCore - GPS 2.1.0

AdaCore is pleased to announce the immediate release of GPS 2.1.0

GPS, the GNAT Programming System, is an advanced IDE that streamlines your software development process - from the initial coding stage through testing, debugging/verification and maintenance. Designed by programmers for programmers, GPS is an IDE that offers the experience of designing software in a uniquely comfortable environment.

The 2.1.0 version is a major release and features many improvements, notably:

- Improved customization, in particular:

  + Extended python support

  + Ability to define new project attributes

+ New commands for accessing GPS internals and manipulating menus/toolbars

Emacs compatibility mode

New help menus

C/C++ improvements

+ Better integrated multi-language build/link

+ More powerful indentation

GPS is available on an ever-growing number of platforms including:

Solaris SPARC
GNU/Linux x86
Windows (NT, 2000, XP)
Tru64
HP-UX
IRIX

Aonix - Eclipse-based Ada IDE for Mission- and Safety-Critical Development

AonixADT opens Ada developers to a gamut of third-party tools

San Diego, CA, Paris, France, July 26, 2004

Aonix®, an independent global company delivering complete solutions for safety- and mission-critical applications, is pleased to announce the Beta release of AonixADT - an Eclipse-based Integrated Development Environment (IDE) for the Ada language. Because Eclipse offers a common platform for which many companies have developed plug-ins, Aonix has extended the wealth of interoperable technologies and tool flexibility available to Ada developers by ensuring that AonixADT (Ada Development Toolkit) builds on such a universal platform.

In the mission- and safety-critical market, developers come from a number of industry sectors, each carrying its own certification standards and specialized tools. By delivering the power of the standard Eclipse IDE with Ada programming language awareness, Aonix provides developers with built-in support for the project manager, editor, difference capability, compile, debug, and command history. Developers can focus on building applications, not on integrating tools since AonixADT also retains a large set of existing plug-ins for third-party tools, including support for source-code configuration management.

"Aonix is committed to providing the mission- and safety-critical developer with an integrated, flexible tool chain," noted Greg Gicca, Director of Product Marketing, Ada Products, "Notably, the release of AonixADT celebrates two first-time achievements - it's the first Eclipse-based Ada IDE and marks the first time that Ada developers have been offered such a wealth of third-party tools already integrated through a universal platform."

AonixADT provides Ada-project awareness, an Ada-language sensitive editor, Ada-language compile and build capabilities, along with a complete Ada debugger interface. ADT project awareness allows full library hierarchy manipulation and Ada program units can be conveniently inserted or removed from Ada projects. The language-sensitive editor provides complete language awareness with syntax color coding and template completion. Symbolic debugging is integrated within the Ada-language sensitive editor. The build interface offers complete access to the Aonix ObjectAda compile and build capabilities.

This first release of AonixADT supports self-hosted development for the Windows platform. Support for Solaris and Linux development and cross compilation for key embedded platforms will be announced subsequently.

Shipping and Availability

AonixADT is available for free with Aonix products. To purchase AonixADT stand-alone, the package is available at $995 per seat.

DDC-I - Comprehensive Migration Assessment Packages

A Valuable Resource for Migrating Legacy Systems to Current Technology

Phoenix, AZ - Aug 2, 2004 - Experienced embedded system software development tools provider and safety-critical, real-time industry leader DDC-I today announced the availability of project-specific Migration Assessment Packages for current programs facing the daunting task of stewarding valuable legacy code into the future. Through a thorough assessment of difficulty, cost and a project timeline for migration to modern technology, DDC-I can design and implement a custom-tailored package for each program's needs.

"Migrating legacy systems to current technology requires careful planning and support. Whether the situation demands legacy system upgrades, application retargeting, rehosting or language migration, DDC-I brings over twenty years of embedded system software development experience to every project," explains David Mosley, DDC-I Engineering Manager and Product Champion for the SCORE® IDE.

According to Mosley, development tools on outdated hosts can be migrated to newer technology and can drastically improve productivity and future flexibility. For example, SCORE® from DDC-I can instantly offer multi-language, multi-target capability, placing fewer restrictions on future work. Thousands of lines of FORTRAN code, migrated to C, can also generate substantial cost savings in reuse - with no shortage of experienced programmers. Why waste resources and time discovering old tool limitations? Process improvements realized using modern development tools and languages consistently reach straight to the bottom line.

Positioning programs with five, ten and twenty year lifecycles for the future also requires replacement of obsolete target hardware. While past custom solutions were prohibitively expensive to develop and unsuitable for mass production, today's COTS solutions frequently meet project requirements at a fraction of the cost of in-house development.

Offering a full spectrum of consulting, training and support services, individually priced and shaped to help customers with complex applications achieve project goals on-time and on-budget, DDC-I possesses broad knowledge and hands-on expertise in every part of the application/development/certification and migration process.

DDC-I's Migration Assessment Package begins with on-site migration needs assessment and application/data/infrastructure evaluation culminating in a complete migration assessment report. Once the assessment is complete, the customer can make an informed choice of how to move forward. They can also choose to do the work themselves, or let DDC-I handle it for them.

DDC-I - WEB based customer support: "Web Pass" Atlas Support

Secure, Personalized Customer Service Portal Provides Highly Targeted Customer Information and Direct Access to Developer Knowledge

Phoenix, AZ - Aug 3, 2004 - Software developers designing and maintaining safety-critical embedded system software require powerful tools demanding powerful support. Already providing a robust back-line for DDC-I's complete range of programming tools, their flexible Atlas Support package now provides personalized access for maintenance customers via a password-protected web portal with a dramatically expanded list of features.

"Web Pass is designed to extend the reach and responsiveness of the DDC-I support department, providing Atlas members a technical database to solve problems and a direct line to the right person in engineering as soon as they need one," explains David Mosley, DDC-I Engineering Manager and SCORE® Product Champion.

The expanded library of product-specific FAQ files, white papers and technical tips in the new Web Pass area are joined by web-based issue submission and t racking and STR tracking features. Via Atlas, project leaders and pre-assigned staff gain direct access to DDC-I's lead product developers - "Product Champions" - experienced engineers directly responsible for fielding technical questions or product-specific problems.

Already serving a "who's who" clientele distributed across the safety-critical, real-time embedded systems software industry, DDC-I's Atlas Support program offers flexible support options to suit the changing needs of customer programs throughout the lifecycle, from development to maintenance. As a software development partner, DDC-I offers a full range of services designed to help maximize productivity at any stage of the development cycle.

Atlas Premium offers special attention during critical stages of development, as software architecture is established and prototyping takes place, when immediate customer support response can make or break a project. Atlas Advantage suits customers past the critical project path, often entering implementation, when timely service and regular version upgrades are essential. Atlas Choice members custom-build a service package combining options to best fit their maintenance and migration needs.

DDC-I - TADS 1750A Solaris Development System Upgrade

TADS Ada Development System for 1750A Processors

Phoenix, AZ - Aug, 31 2004 - DDC-I today announces the availability of the maintenance upgrade release of Sun/Solaris® hosted TADS-1750A Ada Development System (v6.1.1) adding additional run-time, linker and compiler improvements to the full complement of upgrades integrated during the recent TADS rehost to PC/Windows® (v6.0).

"Responding to client needs by consistently upgrading the tools our clients' programs require for success is a primary focus of DDC-I's 'Customer Care' philosophy," explains DDC-I Senior Software Engineer and TADS Product Champion Harold "Bud" Blum. "A wide range of programs depend on TADS for safety-critical embedded system development, alongside DDC-I's flexible customer support and engineering services."

A mature software development solution, TADS generates the most compact code available via the highly optimizing compiler, selective linking, modular run-time systems and an unsurpassed toolset. Improvements to TADS-1750A Run-Time System include: more reliable handling of very rapid interrupt sequences from peripheral hardware; more reliable handling of critical timing delays in multitasking configurations; and assured protection of critical code sections from interrupts under all memory model configurations.

Version 6.1.1 also adds specific optional Ada 95 extensions to the Ada 83 compiler, and linker improvements for more reliable handling of memory allocation by application programs. Also, effective with TADS v6.0, the AdaTrak Profiler is no longer offered, though AdaTrak support for previous TADS versions is naturally still available.

Pre-packaged and custom-built engineering services are designed to help customers maximize program productivity at any stage in the development cycle -- installation and training with full user documentation, basic hot-line support and additional high-level engineering services, full program consultation, and custom software development to specialized requirements. Leveraging extensive experience, DDC-I specializes in migration support services, particularly in the initial assessment phase.

"Keeping TADS performance and support at the highest level demonstrates DDC-I's commitment to their customers, while ongoing market demand and Ada's advantages in code reliability, reusability, readability, and portability continue to prove the superiority of TADS for real-time embedded system developers in aerospace, avionics and defence - or any safety- critical application where failure is not an option," Blum concludes.

I-Logix - UML Model-Driven Development Now Available For SPARK Ada Applications

Rhapsody in Ada Now Offers Rules Based SPARK Ada Code Generation From UML Designs and Integration With the SPARK Examiner.

September 29, 2004 - Embedded systems and software solutions provider I-Logix and leading SPARK analysis tool developer Praxis Critical Systems, announced today a new capability to I-Logix' UML Model-Driven Development (MDD) product, Rhapsody that facilitates the development of fully SPARK compliant Ada applications. A joint development effort between I-Logix and Praxis Critical Systems now enables the generation of SPARK Ada code directly from UML models. This capability is especially valuable for safety critical and other high integrity applications, where SPARK Ada is the language of choice.

SPARK Ada is a language specifically designed to support the development of software used in high integrity applications. As an annotated and semantically rigorous subset of Ada, SPARK exploits the strengths of Ada while eliminating potential ambiguities. These properties, which include rigorous definition, simple semantics, security, expressive power, verifiability and bounded resource requirements, allow SPARK, in both its '83 and '95 variants, to meet all the requirements for software used in critical systems as well as facilitating exceptionally deep and efficient forms of analysis.

With this integration, Rhapsody users are now able to introduce and fully integrate safety critical elements very early on into their design process. Pre- and post-conditions can now be captured as part of a UML design, further augmented to reference proof functions that are tested through the SPARK Examiner. In effect, designers can "think SPARK" while simultaneously taking advantage of the benefits that UML Model-Driven Development (MDD) provide.

Following are the key components of the integration between Rhapsody in Ada and the SPARK Examiner:

Complete SPARK Profile: Rhapsody in Ada provides the system modeller with a UML profile specific for SPARK. This profile allows for an intuitive and highly visible method of modelling and communicating any SPARK design. Rhapsody in Ada then fully generates the SPARK source code and complete SPARK annotations including preconditions, post-conditions, and proof functions. Additionally, Rhapsody in Ada offers the flexibility and control of directly capturing the SPARK annotations, thus providing the end user with an entry mechanism that best fits their working style.

Seamless integration: Rhapsody in Ada works seamlessly with SPARK Examiner from Praxis Critical Systems. From Rhapsody in Ada, the modeller can invoke the Examiner to verify the conformance of the model with the SPARK rules, perform data and flow analysis, and verify the proof contexts. SPARK Examiner results are displayed within Rhapsody in Ada, and a double-click on an error or warning will bring the modeller to the appropriate portion of the design, increasing workflow efficiency.

Fine-tuned control: Rhapsody in Ada allows the modeller to work incrementally on any SPARK design. The modeller has full control over the model elements and the level of examination to be performed by the Examiner. The design can begin with no examination, and then have certain model elements examined for data flow, followed by a full information flow analysis.

The user has the choice to examine each model element individually, or a group of model elements at once, or the entire model. This enables an incremental, iterative and flexible design process that is consistent with the recommended workflow for SPARK while being driven directly by the model.

Customizable SPARK Source Code Generation: Rhapsody in Ada offers a rules-based source code generator allowing the customer total control over every aspect of the model-to-code transformation. From modifications of the SPARK profile itself to changes in the generated source and annotations, Rhapsody maintains a seamless integration for the system developer.

"We are very pleased to see the comprehensive capability offered by Rhapsody in Ada extended to support the SPARK language. The editable rules feature provides for a well-integrated UML environment where designers can leverage the process benefits offered by SPARK Ada very early on in the development process," said Rod Chapman, SPARK Product Manager of Praxis Critical Systems. "Additionally, the generated code can be easily and iteratively analyzed through the SPARK Examiner. The result is a powerful integration of SPARK and UML, generating SPARK compliant code from UML models that is readable, testable and deployable. This combination of individually leading technologies provides much more than the sum of its parts."

"We are very excited about extending our UML based design environment to the SPARK Ada community," said Neeraj Chandra, I-Logix Senior Vice President of Marketing and Corporate Development. "Designers using SPARK Ada are now able to leverage powerful capabilities such as visual modelling with UML 2.0, visual execution and debugging, both on host and target, and flexible, rules based automatic application generation. This new capability allows users to design even larger systems at a reduced cost and higher quality. Combining the benefits of UML with Praxis Critical Systems' excellent SPARK analysis tools lead to a significantly higher level of reliability, safety and security as well."

For more information on any of the company's embedded software products, please feel free to contact us.

About Praxis: Praxis Critical Systems has developed a global reputation in the fields of systems and requirements engineering, software development, safety assurance, information security and risk management and works with some of the leading aerospace companies and other organisations including; Boeing, Lockheed Martin, BAE Systems, ALSTOM Transport, Westinghouse Rail Systems, The Civil Aviation Authority, The Federal Aviation Administration and National Air Traffic Services. The Company's roots are in the application of sound engineering principles to the development of high-integrity software systems whether safety-, business- or security-critical. Its unique tools and products have evolved from practical experience in the most effective approaches to developing such systems. The Company has now diversified into several new markets including financial services, telecommunications, utilities and automotive. For more information, please visit www.praxis-cs.co.uk.

About I-Logix: Founded in 1987, I-Logix is a leading provider of Model-Driven Development (MDD) solutions for systems design through software development focused on real-time embedded applications. These solutions allow engineers to graphically model the behaviour and functionality of their embedded systems, analyze and validate the system, and automatically generate production quality code in a variety of languages. I-Logix also offers iNotion®, a product lifecycle management portal designed for software; coupling product development, quality assurance, marketing and the customer by providing instant, controllable, web-based access to development artifacts and product management services 24/7 worldwide. For more information, please visit our website www.ilogix.com.

McKae Technologies - DTraq

From: Marc A. Criley <mc@mckae.com>
Date: Mon, 11 Oct 2004 08:03:03 -0500
Subject: ANN: DTraq 0.986 is available
Newsgroups: comp.lang.ada

McKae Technologies announces the release of version 0.986 of DTraq, an Ada 95 data logging and review tool.

DTraq is a data logging and playback debugging tool providing near realtime data logging and analysis to aid debugging and validation. Captured, or 'tapped' data from a program can be viewed live while the program is running or, since it is being logged to a file, played back or printed out later for off-line review and analysis.

DTraq differs from other logging and playback tools in that no data layout maps or byte interpretations or "data dumpers" need to be manually created. Nor is the application responsible for converting the raw binary data to text form before logging it. DTraq handles all conversion automatically by scanning the application's source code, identifying tapped data items, and extracting the information it needs to properly convert and display the logged items-simple scalar items as well as arrays and records. When the layout of data items change, rescanning automatically picks up the changes.

DTraq requires GNAT 3.15p due to its reliance on the Ada Semantic Interface Specification (ASIS) and has been validated on Red Hat 9 Linux.

Source and executables are available on the DTraq home page: http://www.mckae.com/dtraq.html, along with the comprehensive and up-to-date user manual -- http://www.mckae.com/dtq_common/DTraq.pdf.

DTraq usage is described, and screenshots provided, starting at http://www.mckae.com/dtq_usage/tapping.html.

Updates to DTraq 0.986 (versus 0.985):

- Improved the handling of data tap Suspension, particularly in situations where tapped data is being received sporadically.

- Reevaluated the generation of informative messages to reduce text clutter.

Ada and CORBA

Windows Support of GLADE

From: Pascal Obry <pascal@obry.org>
Date: 14 Sep 2004 22:43:27 +0200
Subject: Re: GNAT and GLADE
Newsgroups: comp.lang.ada

Tom Bolick wrote:

> GNAT is cross platform, and seems to work fairly well on Windows, can GLADE work on windows? (We want the option of running on one machine in Windows or multiple machines in Linux). There will always be one machine in Windows for the User Interface.

Yes, GLADE works perfectly well on Windows. Also it is possible to have partitions on different computers some on GNU/Linux, some on Solaris and others on Windows. All those partitions will be able to talk to each others without trouble.

> We'd like to test on one machine and then scale up, does GLADE do this? (It looks like it does this very well.)

Yes.

> What do I need to use GLADE?

Well, GLADE obviously :) And GNAT of course.

From: Pascal Obry <pascal@obry.org>
Date: 16 Sep 2004 20:50:06 +0200
Subject: Re: GNAT and GLADE
Newsgroups: comp.lang.ada

Tom Bolick wrote:

> That all sounds great. So how do I get started with GLADE under windows? Do I have to recompile it?

Yes, I think so. This is not trivial but it is not something very hard either. Download GNAT 3.15p binary package and the GLADE 3.15p sources from http://libre.act-europe.fr/.

Ada and GNU/Linux

A# for Mono

From: David Botton <david@botton.com>
Date: Thu, 14 Oct 2004 20:01:12 -0400
Subject: A# for Mono / Linux
Newsgroups: comp.lang.ada

Martin Carlisle has been able to confirm that the A# compiler (Martin's DotNet / GNAT compiler) will compile under Linux.

(See: http://www.usafa.af.mil/dfcs/bios/mcc_html/a_sharp.html)

It is possible to create A# applications that run under Windows, Linux and Mac OS X using martins compiler be it on Windows, Linux, Mac OS X or whatever, since on Linux and Mac OS X, Mono, the GNU DotNET implementation is available.

The current version of Mono comes with Gtk+.NET and therefore it is possible to write GUI applications that run on Windows and Linux using A#. A# automatically generates the Ada bindings needed from .NET / Mono components. It is possible to also use Windows.Forms on Windows which is also being ported to Linux and Mac OS X for cross platform GUI with native look and feel.

He doesn't have ready access to Linux machines to maintain the port of the A# compiler for Linux, but would be happy to work with some one interested in doing so.

If you can, please contact him at carlislem@acm.org

Ada and Microsoft

GNATCOM - COM/COM+/DCOM/Active X for Ada 95

From: David Botton <david@botton.com>
Date: Sun, 24 Oct 2004 15:47:31 -0400
Subject: ANN: GNATCOM 1.4a Released
Newsgroups: comp.lang.ada

GNATCOM, the Professional Open Source Ada 95 COM/COM+/DCOM/Active X binding has been updated to GNATCOM 1.4a. This release incorporates extensive support of directly embedding and controling ActiveX controls for GWindows based applications.

Thanks to GNATCOM, the Ada 95 COM/DCOM/COM+ Development Framework and Tools open every facet of the Windows platforms to Ada 95 development. Never again will the cries be heard, "but there are no bindings" on the Windows platform!

GWindows is being made available under the GNAT modified GNU GPL used by GNAT's runtime library making it availble for use in both GPL and proprietary applications.

For more information on GNATCOM, to view the on-line documentation, and to download the product, please visit http://www.gnavi.org/gnatcom

More information on GWindows can be found at http://www.gnavi.org/gwindows

Microsoft & Ada

From: Stephane Richard <stephane.richard@verizon.net>
Date: Tue, 07 Sep 2004 01:03:23 GMT
Subject: Re: Microsoft & Ada ???
Newsgroups: comp.lang.ada

Stephen Thompson wrote:

> Couldn't help but notice that MS is a platinum sponsor of SigAda 2004. As far as I know this is the first time this has happened. What can we read into this ???.

That is interesting,

For Ada and for Microsoft too I would say. I don't think they would sponsor if they didn't think there was something there for them, maybe, but I don't think so :-). Microsoft Visual Ada Anyone? ;-)

But hey, if Microsoft is sponsoring, I think it's great for both worlds :-).

From: Stephane Richard <stephane.richard@verizon.net>
Date: Tue, 07 Sep 2004 02:39:03 GMT
Subject: Re: Microsoft & Ada ???
Newsgroups: comp.lang.ada

Frank J. Lhota wrote:

> I recall that MS looked into developing such a compiler, but decided not to go forward with it.

Yep that's the way I remember it too.

So then, either they're thinking of hitting the Ada market and/or (probably both :-) See what they can do with that A# project?

From: Jeff <jeff.huter@bigfoot.com>
Date: 10 Sep 2004 00:48:26 -0700
Subject: Re: Microsoft & Ada ???
Newsgroups: comp.lang.ada

Did you notice the following on the A# website?

"Negotiations are in progress with Microsoft to include Ada in Visual Studio .NET"

I personally would love to see this happen. I think it would greatly increase the use of Ada. I'd also like to see the A# project ported to *nix so that it could run on the Mono and Portable DotNet. I think this would fill a big void in using Ada for desktop applications. Namely, the lack of application frameworks and support libraries.

From: Stephane Richard <stephane.richard@verizon.net>
Date: Fri, 10 Sep 2004 12:07:46 GMT
Subject: Re: Microsoft & Ada ???
Newsgroups: comp.lang.ada

Yeah I noticed it (though quite recently) I don't think it's been there that long. I think you're right, if this was to happen, it would be a good boost for Ada. I think there's many other possibilities with Microsoft too that could play a big role for Ada aside A# too :-)....so I think it's all good :-).

From: Pascal Obry <pascal@obry.org>
Date: 12 Sep 2004 09:56:38 +0200
Subject: Re: Microsoft & Ada
Newsgroups: comp.lang.ada

Ludovic Brenta wrote:

> I would be happy if Microsoft provided an Ada compiler, provided that this is really an Ada compiler (as defined by the standard) and not a compiler for a different language. If they call this different language "Ada" or "Microsoft Visual Ada", we're in trouble.

Visual C++ is just this: a different language, and C++ is far from being in trouble... I would love to see Ada in the same level of troubles ! As I said earlier Microsoft Visual Ada will be a good way to introduce Ada in some projects... even if you stick to Ada not the Visual version of it coming from Microsoft :)

From: Ben Brosgol <brosgol@world.std.com>
Date: Mon, 06 Sep 2004 23:39:37
Subject: Re: Microsoft & Ada ???
Newsgroups: comp.lang.ada

Rick Conn from Microsoft is the local arrangements chair for SIGAda 2004, and he has done an excellent job in getting his company's support for the conference (and also in publicizing the event to local companies and colleges). Microsoft's platinum sponsorship is one example of Rick's success, but alas it should not be read as evidence of a sudden language transition within the company :-)

References to Publications

Ada Article at ACM Queue

From: Rom Moran <tmoran@acm.org>
Date: Sun, 10 Oct 2004 17:51:02 GMT
Subject: ACM Queue Ada article
Newsgroups: comp.lang.ada

"There's Still Some Life Left in Ada", Alexander Wolfe, ACM Queue Oct 2004

IEEE Spectrum mentions Ada

From: Marc A. Criley <mc@mckae.com>
Date: Thu, 16 Sep 2004 08:11:36 -0500
Subject: Crash proof software -- when it HAS to fly!
Newsgroups: comp.lang.ada

IEEE Spectrum Online has as their weekly Feature Article "Crashproof Code", "Flying an experimental supersonic aircraft requires rock-solid flight software ".

It's an interesting article, with the Ada code getting a couple mentions.

http://www.spectrum.ieee.org/WEBONLY/publicfeature/sep04/0904air.html

Ada at the Embedded Systems Conference

From: Jim Gurtner <jgurtner@mindspring.com>
Date: Thu, 16 Sep 2004 23:57:09 GMT
Subject: Embedded Keynote Speaker Mentions Ada
Newsgroups: comp.lang.ada

Is this an Ada put down?

Dan Saks said in a keynote speech at Embedded Systems Conference in Boston on Tuesday (Sept. 14):

"In embedded programming, learning a less-popular language like Ada or Eiffel is critical not so much because it is a marketable skill but because it helps programmers see what is possible with more mainstream languages like C, C++ or Java."

Full article at:

http://www.embedded.com/showArticle.jhtml?articleID=47208416

Mention of Ada at Embedded Systems Programming Magazine

From: Peter Hermann <ica2ph@sinus.csv.ica.uni-stuttgart.de>
Date: Fri, 24 Sep 2004 16:45:01 +0000 UTC
Subject: fire
Newsgroups: comp.lang.ada

Great article of Jack Ganssle in Embedded Systems Programming" September 2004 pages 54-56 "Codifying Good Software Design"

exciting

thoroughly investigated

compliment

http://www.embedded.com/showArticle.jhtml?articleID=26806185

From: Jeff Creem <jcreem@yahoo.com>
Date: Sat, 04 Sep 2004 14:00:30 GMT
Subject: Embedded Systems Programming Magazine - Ada Mention
Newsgroups: comp.lang.ada

Embedded Systems Programming September 2004, page 56 in article

"Codifying Good Software Design" makes a very brief Ada mention:

"Just as certain software technologies lead to better code (for instance, C code is generally at least an order of magnitude buggier than code written in Ada), the technology of fireproofing was well understood long before ordinances required their use."

[...]

DDC-I Online News

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

From: jc <jcus@ddci.com>
Date: Tue, 3 Aug 2004 17:22:34 -0700 (MST)
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I
To: 7D August 2004 Online News US <jcus@ddci.com>

DDC-I Online News, August 2004, Volume 5, Number 8 - [http://www.ddci.com/news_vol5num8.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

Migration Assessment Packages Now Available! An assessment of difficulty, cost & timeline for migrating legacy systems to modern technology

Web Based Customer Support - Announcing the New & Improved. Atlas Web Pass. Personalized portal provides highly targeted customer info & direct access to developer knowledge

Thoughts from Thorkil: Exception Handling (1)

We're All in This Together: Practical, down-to-earth ways to help us all work and play well together

From: jc <jcus@ddci.com>
Date: Wed, 1 Sep 2004 17:09:59
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I
To: 14D September 2004 Online News US <jcus@ddci.com>

DDC-I Online News. September 2004, Volume 5, Number 9 - [http://www.ddci.com/news_vol5num9.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

Employee Experience Defines #1 in Customer Care. Success a Direct Result of Talented Staff and No-Nonsense Customer Care Philosophy

Maintenance Upgrade for TADS-1750A. Adds Additional Run-time, Linker, and Compiler Improvements

Thoughts from Thorkil - TADS Ada Static Dumper. Helpful Tool Offers Info About Variables, Constants, Packages, Tasks & Subprograms

Self-Organizing Teams. An Innovative Approach for Any Team Situation

From: jc <jcus@ddci.com>
Date: Fri, 01 Oct 2004 12:53:53 -0700 (MST)
To: 17D October 2004 Online News US <jcus@ddci.com>
Organization: DDC-I
Subject: Real-Time Industry Updates - News from DDC-I

DDC-I Online News. October 2004, Volume 5, Number 10 - [http://www.ddci.com/news_vol5num10.shtml] A monthly news update dedicated to DDC-I customers & registered subscribers.

DDC-I "SCORE's" Experienced Italian Distributor. DDC-I tools now available through ARTiSAN Software Tools (Srl) new Milan sales and service facility.

Migration Assessment ... A Big Hit: Enthusiastic response reveals many customers looking into legacy upgrades.

Thoughts from Thorkil - Exception Handling (2): SCORE's approach to exception handling

It's A Relationship, Not A Sale: As a customer, we want someone who cares and will take action!

Ada Inside

Indirect Information on Ada Usage

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

URL: http://www.adaic.com/jobs/
Date: September 15, 2004 at 18:00:00
Subject: Ada Software Engineer

Job Description:

The job will include development of new, and modification of existing, system software and support software components.

Software engineers will work through the entire software development life cycle from requirements analysis and through integration, test and delivery.

An object-oriented based, open architecture with integrated COTS applications are applied in the development of new system software.

Opportunities exist in several domains from device drivers, network communications, infrastructure, application development, tools, simulations, and GUI.

Education:

Bachelors degree in Electrical Engineering, Computer Engineering, Computer Science, Mathematics, or related field and minimum 5 years of software development experience strongly preferred.

Required Skills:

Software development experience in several of the following areas is preferred: Ada, Ada-95, UNIX, C++, C, GUI design, UML, and Object Oriented Methodology.

Knowledge of, Solaris architecture and near-real-time asynchronous operations is desirable for most positions.

A minimum of 5 years of software engineering experience, is strongly preferred.

Job offer is contingent upon obtaining acceptance by the customer for program access.

Active Top Secret clearance mandatory. Will consider applicants with previously held TS clearance.

Salary: 70k - 85k

Ada in Context

Advantages of Strong Typing

From: Richard Riehle <adaworks@earthlink.net>
Date: Fri, 20 Aug 2004 17:15:53 GMT
Subject: Re: Static vs. Dynamic typing (big advantage or not)---WAS: c.programming: OOP
Newsgroups: comp.programming,comp.object,comp.lang.smalltalk,comp.lang.ada

Cristiano Sadun wrote:

> Dave Harris wrote:

> > I notice you mention, "different primitive types". I suspect a fair fraction of type errors in C/C++ are due to mixing up the integers, int versus short versus long versus bool versus char versus wchar_t versus unsigned. Sometimes it's useful to have that much control, but often it just gets in the way and makes opportunity for mistakes.

> Hm. It could be - but just why you should want to declare something as, say, short, if it isn't necessary? If int or long fit all your situations, just declare everything as int or long.

1) In languages that support automatic type promotion, one is always at risk.

2) Type casting needs to be more disciplined in some of those languages.

3) When I declare a type, with a given bounds, it allows the compiler to check whether I any objects of that type, when used within my program, are likely to be out of bounds (above or below the given range). For example, (in Ada)

```
type T1 is range -473 .. 451;
type T2 is digits 7
   range -30_000.0 ... 100_000.0;
type Unsigned_Small_Integer is
   mod 256;
for Unsigned_Small_Integer'Size
   use 8;
```

I could give many more examples some of which would illustrate the ability to create types of quite sophisticated properties.

In type T1, I have defined a type with a given upper and lower bounds. The compiler can use this information. Also, every Ada program sits on top of a small run-time executive. The RTE is tiny, but it does its job well. When something occurs that causes an object of type T1 to go out of bounds (not common, but possible), the RTE raises a constraint error.

In the case (contrived, I admit) of Unsigned_Small_Integer, we have an unsiged integer that can be represented in eight bits. I use a representation clause (for ... use 8) to force the size of the representation eight. I could just as easily force the size to 16 bits or 32 bits, and then specified the alignment I want for objects of that type. If I make an error, the compiler immediately notifies me of it.

Granted, when writing programs for less critical systems, these features can be overkill. For a large category of embedded, real-time systems, particularly those targeted to bare-board environments), these features are a blessing.

As several people have noted, it is not (or should not be) a debate about the virtues of static typing over non-static typing. Rather the discussion should be about when it is appropriate to use static typing (if often is) and when it is appropriate, as it often is, to use non-static typing. It should not be a debate about Smalltalk or Lisp, at one extreme, versus Ada, at the other the other extreme. Instead, it should be the case that we all learn the tools of our profession, understand when to use which tool, and not be so dogmatic about one tool over another, that we fail to understand the advantages of each tool in different circumstances.

I personally like Smalltalk. I like, but am not proficient in Lisp. I have seen enough successful software in both languages to have respect for both the developers and the languages. At least one Smalltalk aficionado of my acquaintance is smart enough to understand that Smalltalk has its limitations for certain classes of problems. I readily admit that Ada may not always be the right solution. Of course, I continue to believe that there is no situation where I would deliberately choose C++, so I guess I am a bit inflexible in that regard. That is, if I have a choice of Ada versus C++, I would almost always choose Ada. On the other hand, if there were choice between Ada and Smalltalk, Ada and Eiffel, Smalltalk and Lisp, Perl and Eiffel, etc., I would want to evaluate the context, the circumstances, the architecture, and many other considerations before making a decision. Does that not seem a sensible approach?

From: Richard Riehle <adaworks@earthlink.net>
Date: Thu, 19 Aug 2004 00:37:08 GMT
Subject: Re: Static vs. Dynamic typing (big advantage or not)---WAS: c.programming: OOP and memory management
Newsgroups: comp.programming,comp.object,comp.lang.smalltalk,comp.lang.ada

> > Please give us an example of a type definition that is difficult to catch and fix.

> I don't understand what you're asking him. Are you saying that you've never seen a complicated model with a set of types that was ever so slightly /off/ the mark so that it was difficult to discover, and also laborious to fix?

There are several parts to your question.

1) Complicated model
2) Incorrect types ("ever so slightly /off/ the mark
3) Difficult to discover
4) Laborious to fix

My experience is primarily with Ada, so these questions are not quite as relevant as they might be in some other language. Nevertheless, I will answer them.

I have seen designs where the there were too many types defined. In particular, people sometimes design too many floating point types. For example,

```
package Real_Numbers is
   type Real is digits 8
      range -2000.0 .. 2000.0;
   type Degree is digits 6
      range 0.0 .. 360.0;
   -- and many more such
   -- definitions
end Real_Numbers;
```

In the above example, it might be more useful to derived Degree from Real, or create an Ada subtype (I will not define the difference here, but it is different) from Real for Degree. When there are too many variations on floating point, one often has to do too much type conversion elsewhere within the program. However, Ada never lets you get this wrong.

As to incorrect types, this will most often occur in composite types, especially record types. On might forget to include a component of the type in the definition. Since record types, in Ada, are most frequently defined as "limited" types, they are not part of the public part of a specification. This makes it quite easy to correct them without disturbing the integrity of the underlying specification.

As to being difficult to discover, I have not seen this very often in programming with Ada. The language is designed so the compiler will quickly highlight any inconsistencies. The compilation process is not based on textual information alone. Each unit that depends on another unit requires the unit on which it depends is successfully compiled first. The compiler will not even begin to compile a unit unless its dependency relationships have been resolved. This lends itself to rapid discovery of problems with type definitions.

When we consider laborious to fix, I rarely find that to be a problem. In fact, I cannot think of the last time (in nearly 20 years) where the problem with a type was laborious to fix. Certainly, when I was a novice there were problems I could not easily resolve. With experience, I have found that well-designed Ada does not entertain me with the kind of mysterious errors I sometimes encounter in other languages.

I realize that most readers do not benefit from the Ada static compilation model. Still, that model does have the appeal, to those who do enjoy it, of making the type system a blessing rather than a nuisance.

[...]

From: Stephen Leake <stephen_leake@acm.org>
Date: 19 Aug 2004 21:27:25 -0400
Subject: Re: Static vs. Dynamic typing (big advantage or not)---WAS: c.programming: OOP and memory management
Newsgroups: comp.lang.ada

Thomas G. Marshall wrote:

> [...]
> Because it was done in Java, a statically typed language, there was a great deal of energy spent in just turning the crank of making sure that the types shifted from prior strategy to the current one. I hope I've been clear enough here.

I gather you believe it would have been easier to do this redesign in some other language? What language, and why?

If I had done the same thing in Ada (and I have done similar redesigns (I call it "refactoring")), I would expect to "turn the crank" to get all the types right. That's part of what static typing is for; the compiler lets you know where stuff has to be changed. But at each point, you need to make sure that nothing _else_ needs to be changed; often it does.

So I don't see why you feel "turning the crank" is a Bad Thing here.

Ada Compiler Differences

From: Magnus <koma@lysator.liu.se>
Date: 18 Oct 2004 05:47:47 -0700
Subject: Ada compiler differences
Newsgroups: comp.lang.ada

Could someone please point me to a list of things that may be implementation dependent in different Ada compilers and/or on different platforms?

I know that things like 'Access and structures without [representation clauses] may differ. But what else?

Or rather: How can I write code that really is platform (and compiler) independent in Ada?

From: Stephen Leake <stephen_leake@acm.org>
Date: Tue, 19 Oct 2004 21:32:55 -0400
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Nick Roberts wrote:

> Generally, you cannot.

This is a gross overstatement, and a horrible disservice to Ada.

The real answer is: "Generally, for applications that do not deal directly with hardware or other target-dependent features, you can".

Part of the point of Ada is to allow users to write portable code!

> However, you generally /can/ write Ada programs that require very little changes to be ported. Typically, the best technique is to move everything that might need to be changed during a port into separate (library) packages, so that the places where changes are required are isolated from the rest of the code.

That is good advice in any language.

> There are lots of subtle gotchas, unfortunately, that you may need to watch out for. I can list a few. Re-entrancy of pre-defined subprograms: on some implementations, if two tasks make simultaneous calls to a subprogram in certain pre-defined library packages, one or both will not work correctly.

This is also true independent of language. When writing multi-threaded code, you need to be aware of thread issues.

> If you are lucky, your program will just crash; if you are unlucky, it will be a source of subtle, intermittent, infuriatingly uncatchable bugs. Theoretically, this should never happen unless both subprogram calls make reference to the same variable (or file). In practice, I think you'll find some implementations are less than perfect in this area.

Hmm. If the Language Reference Manual does not _explicitly_ state that a particular function is safe for calling from multiple tasks, then you must assume it is not, and provide your own layer of task protection for it. I suspect Nick has been violating this rule.

> If you are not careful, you can run into re-entrancy problems with your own subprograms, too. Since different implementations can multitask in very different ways, its often the case that a potential re-entrancy problem doesn't manifest itself until a program is ported.

That is bad design, in any language. One of the reasons Ada defines tasking in the language is to allow people to write portable multi-tasking code.

So the correct statement here is "If you follow good multi-tasking design principles, Ada lets you easily write portable multi-tasking code".

> Aliasing: implementations are sometimes allowed to choose whether to pass a parameter by value or by reference (indirectly). If it so happens that a call to a subprogram effectively passes the same variable as two different (formal) parameters, the subprogram has two different 'paths' to the variable, without knowing it. The order in which updates to the variable occur could change from one implementation to another, in this situation. This can be a source of some really mysterious bugs, when porting.

True. Also easy to avoid, once you are aware of it. Hmm. There aught to be an ASIS based tool to check for this.

> Order dependency: implementations are generally allowed to choose in what order they evaluate the expressions passed as (actual) parameters to a subprogram call. If more than one of these expressions has a side-effect, and the side-effects could interact in some way, it possible that the order the implementation chooses could affect the behaviour of a program. This can be a source of subtle and nasty bugs when porting.

While technically true, I don't recall anyone posting such a bug here. And I have never encountered such a bug.

Typical code just doesn't have this problem.

> The values of everything declared in the predefined package 'System' are all implementation defined, as well as in its children, and the subprograms will all work differently. There may be extra imp-def declarations, and some declarations may be omitted or different to what the standard states. So use of these packages needs care, from a portability perspective. It's best to avoid using anything here if you can.

True.

Another area of non-portability is GUI interfaces. Since the Ada standard does not define a GUI library, code you write using a GUI library is only as portable as that library.

From: Mark H Johnson <mark_h_johnson@raytheon.com>
Date: Mon, 18 Oct 2004 16:48:38 -0500
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

I would start with chapter 13 of the Ada Reference Manual. Simple things like Integer'Size (e.g., 32) might not equal Natural'Size (e.g., 31) can trip you up in a variety of different ways. I also had problems with some code assuming 'Size of the object was the same as 'Size of the corresponding type.

I would also suggest a review of the Annotated ARM as well. The annotations are quite enlightening when trying to determine the possible implementation details of the language. [...]

To do that fully would require an extensive list of issues and is often constrained by the types of platforms you expect to run on. For a simple example, network order (for TCP/IP) is "big endian". If you are on a little endian machine, several values need to go through htonl, htons, and similar functions to be converted. If you "know" you will always on a big endian machine (same as network order), you might get away without them, but that isn't portable.

That particular issue has nothing to do with Ada; you have the same problem with C or other languages.

From: Jacob Sparre Andersen <sparre@nbi.dk>
Date: 18 Oct 2004 16:01:50 +0200
Organization: CRS4, Center for Adv. Studies, Research and Development in Sardinia
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

One advice:

Don't depend on predefined types (Integer, Float, Natural, etc.) where it makes more sense to declare a specific type for your specific use.

(you'll probably get more)

From: Nick Roberts <nick.roberts@acm.org>
Date: Mon, 18 Oct 2004 20:55:15 +0100
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Luke A. Guest wrote:

> Yeah, you'll need to define your own types, but surely these are going to be "derived" from the default types, i.e.
> ```
> type Chutney is new Integer
>    range 1 .. 5;
> ```
> Are you saying not to even do this?

Yes. Normally it is better to simply put:

```
type Chutney is range 1..5;
```

and allow the implementation to select the best base type.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Tue, 19 Oct 2004 02:11:45 GMT
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

I rarely do this. What is the point of specifying the representation (except at the edges of the application)? Why use 32 (or 64) bits for 5 values? Let the compiler choose the best representation for the target.

From: Martin Dowie <martin.dowie@btopenworld.com>
Date: Mon, 18 Oct 2004 21:03:36
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Leave as many aspects of the underlying representation to the compiler. It knows what the target is, so in this case it may choose a 8-bit representation as opposed to say a 32-bit one. Perhaps the target has faster instructions for 8-bit data items.

From: Jeffrey Carter <jrcarter@acm.org>
Date: Wed, 20 Oct 2004 01:16:54 GMT
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Luke A. Guest wrote:

> Even if you use a use clause to specify how many bits you want it to take up (i.e. even in a record)?

The only reason to do that is because you need to map to something like hardware where this is required, so it's a fairly rare case (what I called the edges of software). Even then, there's no advantage to using a derived type. Indeed, using a derived type is giving contradictory information to the compiler: "use the same representation as Integer" and "use only N bits".

From: Rod Chapman <rod.chapman@praxis-cs.co.uk>
Date: 20 Oct 2004 01:02:48 -0700
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Write the code in SPARK, which has only 2 known implementation-defined features - the range of predefined base types (which can be added by way of annotations), and the finer details of floating point (signed zeros, rounding mode etc.)

Example: the port of the SPARK Examiner (written in SPARK of course...) from Solaris to Linux took...about 20 minutes, most of which was spent installing GNAT. The Examiner is about 70kloc, so hardly a trivial program.

From: Rod Chapman <rod.chapman@praxis-cs.co.uk>
Date: 22 Oct 2004 01:13:59 -0700
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Larry Kilgallen wrote:

> I suppose an _unknown_ implementation-defined feature is defined by the term "bug" :-)

If you can find _any_ implementation-dependent, implementation-defined, or erroneous behaviour in SPARK (other than the 2 I mentioned above), then please report it to sparkinfo@praxis-his.com

From: Stephen Leake <stephen_leake@acm.org>
Date: Wed, 20 Oct 2004 09:05:50 -0400
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

Simon Wright wrote:

> Stephen Leake wrote:

> > Hmm. If the Language Reference Manual does not _explicitly_ state that a particular function is safe for calling from multiple tasks, then you must assume it is not, and provide your own layer of task protection for it. I suspect Nick has been violating this rule.

> What, even Generic_Elementary_Functions.Arctan ?!

Well, you have a point. Although, on a system without floating point hardware, this _could_ use a global cache of previously computed results, which _could_ be not task safe.

> On the other hand, anyone who called Float_Random from two tasks *with the same generator* would be entitled to expect truly random results. I think you have to be prepared to use your wits sometimes ..

It could also be argued that the LRM _should_ say more about what is guarranteed to be task safe.

From: Randy Brukardt <randy@rrsoftware.com>
Date: Mon, 25 Oct 2004 19:28:57 -0500
Subject: Re: Ada compiler differences
Newsgroups: comp.lang.ada

This is wrong. As long as the tasks are not passing the same objects, the language requires that multiple tasks can call predefined routines at the same time. See A(3). (That's the third paragraph of the introduction to Annex A.)

Any implementation that fails to do that for any predefined subprogram is incorrect. Of course, it is not unusual for implementations to be incorrect,

especially in this area; and that seemed to be Nick's original point.

It will be very important that this property is true for the containers, for instance.

Ada and the GNU Build System

From: Lutz Donnerhacke <lutz@iks-jena.de>
Date: Wed, 4 Aug 2004 02:05:12
Subject: Re: GNAT and GNU build system
Newsgroups: comp.lang.ada

Tapio Kelloniemi wrote:

> I'm planning to seriously develop free software packages in Ada. The problem I have is GNAT's fitness for GNU program building standards.

You don't need that. Ada comes with portable interface definition capabilities.

> I would like to autoconfig my project (not very bad, some M4 macros can be found in GtkAda). But I would like to use automake (my projects are in Ada and C, because almost every free software library is written in C).

The monkey argument is no argument for C. You do not need autoconfig for Ada.

> If anyone has had any experience or interest regarding this, please drop a line.

I prefer redefining the standard C-header in Ada. This is necessary, but nearly impossible, because the C-library and kernel-call interface generated depends on the C-compiler version and flags used when compiling the kernel and the libs.

That's why there is no generic way to automatically determine the interface for Ada. In order to do it in a portable way you have to study the interface deeply and redefine it in Ada yourself. Luckily the ABI does not change this hard, but depends on CPU and OS-version.

I'd recommend a libportable written in C, generated with autoconfigure and all those nifty workarounds about a missing interface definition. This libportable should convert the CPP-headers into C-headers, so that preprocessor definitions become linkable. The Ada code will be constant and portable, because the value of the day for a system constant can be linked.

From: Ludovic Brenta <ludovic.brenta@insalien.org>
Date: Wed, 04 Aug 2004 21:21:40 +0200
Subject: Re: GNAT and GNU build system
Newsgroups: comp.lang.ada

I also tend to think that autoconf and automake are poor hacks meant to circumvent the deficiencies of C; just like make in fact. However, I see one situation where it makes sense to use them in Ada, and that is when you want to build a binding to a C library that has platform-dependent parts. This is best illustrated by Florist, which has a rather convoluted build process:

1) It uses configure to generate a Makefile and a config.h
2) The Makefile compiles a number of C source files, and links them into an executable, named c-posix
3) The Makefile executes c-posix, which writes a gnatprep configuration file and a C file, c-posix-signals.c
4) The Makefile runs gnatprep, with the configuration file, to generate several Ada files
5) The Makefile compiles and links c-posix-signals.c
6) Executes c-posix-signals, which creates posix_signals.ads
7) Compiles all the Ada files
8) Links the compiled files into a library

See http://buildd.debian.org/fetch.php?&pkg=libflorist-3.15p-1&ver=3.15p-3&arch=sparc&stamp=1069154570&file=log&as=raw

But the conclusion I draw from all this is: do your very best to avoid such a complicated build process if possible! Unless you are interfacing to the operating system, there is very little reason to use the autotools. GNAT and Florist together should provide you all the portability you need (and yes, GNAT does use autoconf/automake, because GCC does).

From: Stephen Leake <stephen_leake@acm.org>
Date: 04 Aug 2004 18:06:19 -0400
Subject: Re: GNAT and GNU build system
Newsgroups: comp.lang.ada

Sounds reasonable. Others have said "you don't need automake with Ada". That's almost true, but any _real_ project also has tests, documentation, and distribution needs which are beyond Ada's scope. autoconf/automake help with those tasks, and Ada needs to at least cooperate.

I'm using automake in an Ada project (a GtkAda interface to a books database). It's not ready for prime time yet, but I'd be happy to exchange notes about autoconf use.

From: Ludovic Brenta <ludovic.brenta@insalien.org>
Date: Wed, 04 Aug 2004 22:34:25 +0200
Subject: Re: GNAT and GNU build system
Newsgroups: comp.lang.ada

Tapio Kelloniemi wrote:

> I would really prefer typing: ./configure --prefix=/usr --enable-goldobj

I understand your concern completely. My concern, though, is the sheer complexity of the ./configure script itself, and that of the Makefile it generates. Most of that complexity is unnecessary with Ada programs, and in fact really gets in the way. With Ada, you would normally want to take full advantage of GNAT project files. The only places where ./configure may be of help are for the installation target (--prefix), and finding any Ada libraries you depend on.

<plug mode=shameless>

This problem is solved nicely in Debian GNU/Linux. Each library has a GNAT project file in a well-known location (/usr/share/ada/adalib/library.gpr). Your program just "withs" them as required.

</plug>

Furthermore, on non-Debian systems, the GNU Ada Environment Specification[1] says where library files should be installed. Here again, you would not need a ./configure at all; just use -aI and -aO as necessary.

<plug mode="really_shameless">

Since Debian follows the GNAE, your GNAT project file can be portable without the need for ./configure.

</plug>

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

So, you could consider writing a ./configure script by hand, which would generate a minimal Makefile containing only the value of --prefix. Something along the lines of:

```
PREFIX=/usr
all: my_program
my_program:
        gnatmake -Pmy_program.gpr
install: my_program
        cp my_program
$(PREFIX)/bin
```

If you need package-specific options (e.g. --enable-gold-objects), then you can extend the ./configure and generate something like:

```
PREFIX=/usr
GOLD_OBJECTS=false
all: my_program
my_program:
        gnatmake -Pmy_program.gpr
-XGOLD_OBJECTS=$(GOLD_OBJECTS)
install: my_program
        cp my_program
$(PREFIX)/bin
```

From: Tapio Kelloniemi <spam12@thack.org>
Date: Thu, 05 Aug 2004 08:50:37 GMT
Subject: Re: GNAT and GNU build system
Newsgroups: comp.lang.ada

And I think that autotools should also cooperate. FSF says that free software packages in GNU system should cooperate with others. Luckily GNAT is part of GNU project and I think autotools writers could very well take a step towards Ada from the complete C centralism. (This C centralism is unfortunately accepted by GNU projet and very unfortunately even required by the GNU coding standards.)

[...] There are still things that cannot be solved with Ada, consider such a real time programming need as memory mapped IO, which is very non-portable.

Configure could perhaps generate a Makefile, which runs automake with the appropriate GNAT project file (assuming that GNAT is the only compiler to support). The project file used could be choosed by configure (eg. if --enable-maintainer-mode is specified, choose debug.gpr, instead super_optimise.gpr).

> This is true. So we need to start generating patches to auto* that take advantage of Ada's simplicity. Not an easy task, but in the long run it is the right solution.

Note that automaintainers are not allowed to accept our patches (if longer than few lines), if writer doesn't sign a paper that he doesn't own the code (this is the FSF's requirement). I really think that this should be done in tight cooperation with the GNU developers, because they have deep knowledge of their tools and we have that of ours.

> Hmm. Maybe rewriting auto* in Ada first would be easier; I'm not sure :).

I think that using GNU tools is a good idea not to couple efforts. Pure Ada version would of course be easier, but in mixed language projects, tools such as autoscan, autoheader and others are also needed.

Things that I think should be done to auto* are:

- Autoconf should be able to generate gnatprep definition files, perhaps from template like config.adp.in:

-- Define this to true, if libfoo support should be included.

```
HAVE_libFOO := @LIBFOO_SUPPORT@
```

Better even could be that configure would generate this from scratch.

- Automake should be able to compile Ada programs (in a way or another).

- Automake should know how to install Ada libraries (unfortunately installing .so and header is not enough.)

- Libtool (never used, don't know what should be done to it)

- Gettext and autopoint (gettext should be able to scan Ada sources for strings)

> Only if you have a pure Ada project. I find this unlikely; I like LaTeX or Texinfo for documentation, and I want my makefiles to run test drivers.

You speak truly. Because almost every library is written in C, C interfacing is required, as ugly and unwanted as it is. It is very often better to share than write giant size programs which introduce bugs, because somebody understood something incorrectly.

From: Ludovic Brenta <ludovic.brenta@insalien.org>
Date: Sun, 08 Aug 2004 16:45:15 +0200
Subject: Re: GNAT and GNU build system
Newsgroups: comp.lang.ada

The usual way to do configuration management in Ada in these situations is to have several bodies for a single spec, and choose one body when configuring. This can be done in several ways. The GNAT way is to symlink the file containing the particular body to the canonical name. For example, one body is "5msystem.ads" and it is symlinked to "system.ads" before building. This however is done from the Makefile and does not, per se, require a configure script.

Another way is to have one target-dependent directory, and pass a variable to a GNAT project file to select which target-dependent directory should be added to the Source_Dirs.

A third way, used by GPS, is to write a GNAT configuration file containing configuration pragmas. The compiler-specific pragma Souce_File_Name can associate a body with the selected file.

A fourth way could be to use gnatprep (like Florist does).

[...] I would probably be better to use a single project file, and pass variables on the command line (you can have a case statement in a project file).

As you can see, there is a lot you can do to influence the build. Some of this has to be done at configure time; some can be done from within the Makefile.

I think that the general idea is to have a minimal configure script that only does what is strictly necessary, i.e. generates the Makefile. I do not like GNU configure because it tries to be everything to all developers, and ends up in unmaintainable and unnecessary complexity.

If the Makefile can be written in a portable way, then I would prefer not to have a configure script at all (or a no-op configure script, to please people with pavlovian reflexes).

Memory Management in Ada

From: Martin Krischik <krischik@users.sourceforge.net>
Date: Thu, 07 Oct 2004 14:06:41 +0200
Subject: Re: Ada memory management?
Newsgroups: comp.lang.ada

> Since there is an allocator 'new' in Ada, I was wondering if there is a 'delete' too. I've heard there is a special technique called Storage Pool in Ada to write own memory managers, but what is the default deallocator?

Two option open: Your Ada has garbage collection or you use Ada.Unchecked_Deallocation.

AFAIK: Only Ada [platforms] targeting the JVM have garbage collection. However for GNAT you can use the Boehm Collector which is part of the GCC.

Mind you: A good collection class library will free you from almost all memory management.

From: Nick Roberts <nick.roberts@acm.org>
Date: Thu, 07 Oct 2004 18:24:21 +0100
Subject: Re: Ada memory management?
Newsgroups: comp.lang.ada

There are some Ada implementations where memory never leaks (unless a bug in the compiler or run time system causes one, or the programmer causes it by poor unchecked or external programming). Automatic deallocation combined with comprehensive memory reclamation is called 'full garbage collection'. I think, in reality, there is no native code Ada compiler which supports full garbage collection.

Personally, I regret this situation. The compiler vendors all say they don't support full garbage collection because there is no demand, which is true considering the paying Ada compiler market is almost entirely made up of the embedded, real-time, and safety-critical arenas. But the fact that the Ada language is capable of protecting the programmer from the possibility of memory leaks used to be, many years ago, one of the reasons cited for its superiority. The complete lack of support for full garbage collection seems unfortunate to me. It is a proud capability of many languages which have since become far more popular than Ada.

I am trying to build my own (open source, GPL) Ada compiler, and I hope to be able to build in full garbage collection support, if only to prove that it can be done. But I do not have recourse to huge resources, so don't hold your breath.

http://sourceforge.net/projects/eclat

I am a (moderate) fan of full garbage collection, but I concede that it is not, in reality, as useful as you might think. One could guess that 90% of 'desktop' Ada programs would not benefit from the availability of full garbage collection, for one of the following reasons: the program does little or no dynamic memory allocation; the program does not give any (or much) opportunity for automatic deallocation until (near) the end of program execution; more control over the management of (most) dynamically allocated variables is required; the program must be written to be portable to other Ada compilers (which do not or may not support full garbage collection); the program only uses pre-written encapsulated 'containers' for dynamic data

storage, which all do their own memory management anyway.

Many programs will only dynamically allocate variables of a fixed size (known at compile time). By having a separate pool for each different size, the necessity for memory reclamation (moving things around) is obviated; only automatic deallocation is useful. I do not know if any existing native code Ada compiler actually supports this technique (doing it automatically). It could be done manually, by implementing one's own storage pool.

Probably, full garbage collection would never be used for an embedded or real-time program, because it is a big-memory technique, and does not conform to strict timing requirements. I'm not quite sure about safety-critical programs.

However, for the remaining 10% -- or whatever the figure really is -- of desktop programs, full garbage collection would certainly be a useful facility, and I am certain it is why many good programmers often choose languages such as Python, Ruby, and various others which support it, in preference to a language which (in practice) doesn't.

Ada Movies on AdaPower.com

From: David Botton <david@botton.com>
Date: Tue, 9 Nov 2004 00:43:18 -0500
Subject: Ada Movies on AdaPower.com
Newsgroups: comp.lang.ada

Oldies but goodies :-)

Requires quick time.

AdaPower Cinema, http://www.adapower.com/index.php?Command=Media

David Botton

Ada and Malicious Software

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Newsgroups: comp.lang.ada
Subject: Ada and malicious software
Date: Wed, 22 Sep 2004 09:21:42 GMT

Tom wrote:

> One question that I would like an answer for is: Is Ada less susceptible to computer virii than C++ and Java on the Windows XP operating system? Now that is a question that would come up more often where I work.

If you mean true viruses then I can't see that there would be any difference. The choice of programming language doesn't affect a program's ability to modify the code of another program. (On the other hand, choice of operating system does.)

If you mean malicious code in general, then yes, Ada programs would be less subjected to worms, cracking tools and other things that exploit security holes. Security holes are often buffer overflows, and as Jean-Pierre Rosen said, Ada programs do not have buffer overflows. Arithmetic overflows also occur, and Ada protects well against those too.

From: Jean-Pierre Rosen <rosen@adalog.fr>
Subject: Re: Ada and malicious software
Date: Thu, 23 Sep 2004 09:33:02 +0200
Organization: Adalog
Newsgroups: comp.lang.ada

Warren W. Gay wrote:

> That should probably be tempered with "not as susceptable" to buffer overflows. If there was poor design and/or testing, then a production mode program that is compiled with the checks "off", is still vulnerable, although admitedly, much less likely.

Of course. You can write badly in Ada. But the difference is, you have to ask for it!

Scripting Languages and Ada

From: Marius Amado Alves <amado.alves@netcabo.pt>
Date: Fri, 03 Sep 2004 17:42:07 +0100
Subject: Re: Learning Ada83
Newsgroups: comp.lang.ada

> It would also be useful and practical to learn one of the advanced scripting languages like Perl or Python or Ruby....

Wow, they're *advanced* scripting languages now! And I thought real man did all their scripting in Ada these days :-) Now, less jestly, I find this a strange advice in this list. If you want to go the Great Ball of Mud way why not just recommend PAM (PHP + Apache + MySQL) and get done with it? (Sorry, every now and then I can't resist a good language battle.)

From: Randy Brukardt <randy@rrsoftware.com>
Date: Fri, 3 Sep 2004 18:47:58 -0500
Subject: Re: Advanced scripting languages (was: Learning Ada83)
Newsgroups: comp.lang.ada

That's funny, because if it's complicated enough that I can't write a batch file to do it, I'll generally write it in Ada. Bat has If and Goto, and that is enough for simple tasks. Beyond that, I want to be able to fix it and be able to insure that it works...

From: Kevin Cline <kevin.cline@gmail.com>
Date: 4 Sep 2004 20:28:16 -0700
Subject: Re: Advanced scripting languages (was: Learning Ada83)
Newsgroups: comp.lang.ada

Strong typing is handy, but it's not enough to ensure that something works. Ada has it's strengths, but it's not the tool for every job. Why would you spend an hour writing 50 or 100 lines of Ada code when five minutes and a five-line Perl script would do the job?

From: Randy Brukardt <randy@rrsoftware.com>
Date: Tue, 7 Sep 2004 19:07:36 -0500
Subject: Re: Advanced scripting languages (was: Learning Ada83)
Newsgroups: comp.lang.ada

Because it would take a week to learn Perl well enough to use it that way. And it is very rare that such a program is never used again; my one experience with Perl showed it to be a write-only language (it took me two weeks to successfully modify the code that runs the CVS on the ada-auth.org site). And it is very rare that I need to write a 100 line throwaway program; most of my code has much more permanence than that. Summary: It doesn't make sense for me. YMMV.

From: Ken O. Burtch <kburtch@sympatico.ca>
Date: Wed, 08 Sep 2004 09:38:12 -0400
Subject: Re: Advanced scripting languages (was: Learning Ada83)
Newsgroups: comp.lang.ada

The documentation for the AdaScript Business Shell (http://www.pegasoft.ca/bush.html) contains several arguments against tools like Perl in favor of Ada-based scripting languages, including:

1. Lower learning curve (a BUSH for loop is an Ada for loop)
2. Lower overall cost for maintaining a script project over its lifetime
3. Error messages that work for you, not against you
4. Better readability when debugging other people's scripts.
5. Sensible syntax shortcuts designed to be typo-resistant.
6. Better portability.
7. Code reuse.

When you look at the big picture, Perl projects increase costs and development time. As always, use the right tool for the job. But my experience with Perl development over the past 2 years has been negative. The idea of a 5 line Perl script to replace 100 lines of Ada is a myth.

Ken B.

Author of "Linux Shell Scripting with Bash"

Mantainability of Ada, C++ and Perl

From: Kevin Cline <kevin.cline@gmail.com>
Date: 21 Sep 2004 17:21:38 -0700
Subject: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

As promised in the Ada popularity thread, I have taken one of the Charles examples and reimplemented it in C++. I used only

the standard C++ language and libraries. The Ada/CHARLES main program body is 118 (non-blank) lines of code, plus an additional 40 lines of instantiations in eight other specification files, for a total of 158 lines and 9 files. The C++ implementation is 76 (non-blank) lines of code in a single file. For grins, I also wrote the program in Perl. That took 14 lines.

Summary:
Ada/Charles 158 lines, 9 files
C++ 76 lines
Perl 14 lines

You can compare the implementations at http://www.geocities.com/kc0a/wordcount.html

From: Björn Persson <rombo.bjorn.persson@sverige.nu>
Date: Wed, 22 Sep 2004 08:50:05 GMT
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

Neat. Now let's take three identical persons who haven't seen this program before, give them one version each, and see how long it takes them to understand it.

In other words: What matters is how easy the program is to read and maintain. The line count is interesting only to the extent that it affects the readability.

From: Benjamin Ketcham <bketcham@drizzle.com>
Date: Wed, 22 Sep 2004 13:38:33 -0000
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

Well, the line count affects readability rather profoundly, in this example! Not being an expert in any of the languages concerned, I can take a more objective view of readability perhaps; and I can say that if I had to figure out and maintain any of these programs, the one that easily fits in entirety on an 80x24 screen wins my approval. I.e., this might not scale to larger projects: I'm not at all sure I'd rather read a 1400-line Perl program than a 7600-line C++ program or a 15800-line Ada program. Actually, "none of the above" sounds most appealing.

OTOH, it does not appear that the three programs actually implement the same spec. E.g., error messages are different (and missing in the Perl version), and if I'm not mistaken the Perl version reads input from stdin, the other two take a file argument -- or is while(<>) in Perl smart/twisted enough to read from a filename in argv[] if present, else stdin? I certainly recall that it has extensively overloaded behaviour. Anyway, this is not even a vaguely fair test until all three actually have the same behaviour in detail.

From: Cesar Rabak <crabak@acm.org>
Date: Wed, 22 Sep 2004 12:16:22 -0300
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

I second this. And for a quick and dirty comparison we have to figure out some (unique for all implementations) test suite, so we can check the overall behaviour (not only performance).

From: James Alan Farrell
Date: Wed, 22 Sep 2004 11:27:01 -0400
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

Does this prove that perl is the best language? If so, perhaps we should all switch our avionics projects over to perl. Must be it will be easier to code and maintain, safer to fly, and run just as fast.

The things that make a language a "good" language are many and complex, and are different in different circumstances. If I need a small utility that compares lines in different text files, I find perl much easier than Ada. On the other hand, if I have a very large project that must always perform "correctly", I've not found a language that can beat Ada.

But then I've never used Eiffel.

From: Chris Humphries <chris@unixfu.net>
Date: Mon, 27 Sep 2004 09:22:06 -0400
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

Kevin Cline wrote:

> Again, the original question was "Why isn't Ada more popular?"
> The answer I am giving is that most programmers don't have such an overriding concern for safety that they are willing to write twice as much code to get the additional safety.

[...]

Thanks, and good answer. Since then asking that question, I have learned a lot more about Ada. John Barnes' book, "Programming in Ada 95, 2nd Edition" helped tremendously. A very good book. It is now nice to understand just what Ada is, and not just another programming language, but more an attempting at progressing software engineering.

I also think I understand why Ada is not more popular. Though I am very thankful for open source, and feel very strongly about about it, something bad has grown out of it: bad programmers and programs. Most open source projects seem to have no formal software development process. There is more an attitude of "shut up and hack", which typically goes by whatever they get features to do from, which may be from whatever they want, if there is a TODO list, then you are lucky. There is usually no goals or anything actually defined, no use cases (I would put money on most do not even know what one is, and if they did, what they would use it for), no unit tests or any automated framework for making sure the code does as designed (which brings me to my next one), no design documentation or even a design process.

Many young/inexperienced programmers do not even see why this is important, as their code works, and generally they are done with it then and there. In the real world, and in my job, most of the time is spend updating code. Good times are when I get to design something new, most the time is spent doing grunt work.

In a time when most programs are written by having a 1-2 thought process of 1) I want my program to do this and 2) type-type-run-repeat, such languages that allow you to do this easier are more popular, such as php and perl.

Granted, I do code perl for some of the legacy projects here (no I am not a web developer, heh), and if standards are in place, perl code can be very readable and understandable. Coding standards are nice, yet seem to be rarely used.

[...]

From: Kevin Cline <kevin.cline@gmail.com>
Date: 27 Sep 2004 14:31:07 -0700
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

Bad programmers didn't come from open source. There have always been bad programmers, and there always will be. Just as it is with music and writing and film and art, 90% of all programming sucks.

[...] On a good project, design is a continuous process. Unfortunately, most programs reach the point where any change is so risky that developers spend all their time trying to figure out how to squeeze in a new feature without breaking the existing features.

From: Brian May <bam@snoopy.apana.org.au>
Date: Tue, 28 Sep 2004 09:51:48 +1000
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

I know of commercial programs that are developed (designed, implemented, and tested) very badly just as I know of open source projects that appear to be handled very well.

Commercial projects can be put under pressure to cut corners, e.g. because management sees a formal software development process as an academic exercise that is a complete waste of time "in the real world"(TM), despite protests from programmers. The same management see software development as a tedious and unreliable process, and don't make the connection that the lack of a proper development process could be

leading to the difficulties. They often give solutions (that may be inadequate) rather then requirements, turning the task of programming into a guessing game.

Open source developers aren't under these pressures, and have more scope for doing the right thing.

(This isn't to imply that all open source projects are developed correctly; some are bad too; this doesn't imply all commercial projects are developed badly either).

From: jayessay <j-anthony@rcn.com>
Date: 23 Sep 2004 15:30:11 -0400
Subject: Re: Ada Popularity: Comparison of Ada/Charles with C++ STL (and Perl)
Newsgroups: comp.lang.ada

[...] Ole-Hjalmar Kristensen:

> Kevin Cline wrote:

>> But large projects in Ada or C++ or Java or C# might be small or medium-sized projects in a higher-level language.

> Maybe. It depends on how large the project is, and how good you are to create reusable components/abstractions within the project, and how good a fit your problem is to the facilites provided by the "higher-level" language.

This is key. The robustness, maintainability, evolvability, and "correctness" of a program is directly and closely related to how close its expression is to the language of the problem domain.

In the case of Lisp, this is where the "Lisp is a programmable programming language" comes in. You can always create a (possibly hierarchical set of) domain specific language that directly supports the most natural, clean, and direct means of expressing the definitions and process within the domain. People often talk of "hoisting the language up to the level of the domain".

It used to be (and I suppose in many places still is the case) that people thought this sort of thing could be done with components and component libraries. But this is an error. No library (no matter how good it is - and most are very poor) will ever have the level of expressiveness as even a decent (let alone good) domain specific language.

>> If you have a project that must always perform "correctly", then you better prove it correct. Strong typing can help, but is not absolutely necessary to that effort.
For the rest of us, the most important thing is to get tested code done quickly. I don't know about you, but I can write and test 14 lines of code a whole lot faster than I can write and test 80 or 160.

> I usually find that my ability to create programs is limited by my thought process, not my typing speed :-)

He's not referring to the same limitation, and he's right.

Artistically creative expression has no role in software design

From: Marc A. Criley <mc@mckae.com>
Date: Mon, 19 Jul 2004 13:46:58 -0500
Subject: Artistically creative expression has no role in software design
Newsgroups: comp.lang.ada

(Okay, now that I have your attention... :-)

In spectating the "SCO vs Linux" lawsuit (www.groklaw.net), a lot of documents of various types get posted. One of the recent references was a paper titled, "The case against Copyright Protection of Non-literal Elements of Computer Software" http://tinyurl.com/3tjqj, by Christopher Heer of the University of Toronto.

The paper analyzes something called the "Abstration-Filtration-Comparison test (AFC test), which is a court created means for attempting to determine copyright infringement of software where no _literal_ copying was involved, or perhaps was obscured.

One of the interesting conclusions of this paper is this:

"Since the design of computer software is forever driven by its intended functionality and efficiency concerns, the room for artistically creative expression never arises. [...] It is more appropriate to consider the software objects of a computer program as analogous to the gears, pulleys, and levers of a mechanical invention, as by its very nature, the design of computer software is intended to optimize functionality by making a program run faster, use less memory, or be easier for the programmer to modify."

Since programming in Ada has been sneered at as requiring that a programmer "lose their programming freedom" and likened to "programming in a straitjacket", this article argues that those are in fact proper characteristics in the development of correctly functioning, optimized software!

Software is pure function; there are inputs (data and time), transformations, and then outputs; although the software's design and implementation may be formulated in a specific way that promotes one or more of Heer's three functionality optimization axes. Arguably then, for the first two axes, more speed and less memory, there should be an optimal design and an optimal implementation.

Add in the overriding requirement that the software must be _correct_, and Ada shines--its definition supports establishing and verifying programming correctness--and subsets with supporting tools like SPARK (www.sparkada.com) even take that a serious step further.

Ada's case for the third axis of optimization, "easier for the programmer to modify" (which is subjective), can be strongly made as well. With its design goal of readability, fully object-oriented capabilities, and the strong typing that makes it easier to modify software _correctly_, again Ada shines.

After you get past the knee-jerk reaction to Heer's conclusion (which I'll admit to), sit back and really think about software, its function, and how to achieve _correctness_ and _efficiency_ in design and programming. Software development starts to become less about creative expression, and more like a quest, trying to find the elegant implementation of functionality. Refactoring, anyone?

From: Alexander E. Kopilovich <aek@VB1162.spb.edu>
Date: Tue, 20 Jul 2004 05:45:33
Subject: Re: Artistically creative expression has no role in software design
Newsgroups: comp.lang.ada

It seems that the author of that paper knows far too little about art and artistically created expressions. Perhaps he thinks that art is overwhelmingly not functional, but decorative... and that true artists never worry about restrictions and consequences, being driven by mystical revelations.

I'd like to recall here an interesting (and not rare) kind of art - propaganda art, which from time to time thrives both in literature and in movies (especially in war or tension times). There are plenty of examples of true art of this kind - and certainly the ultimate purposes of those things were and are functionality and efficiency.

Another well-known generic example is architecture - one may recall that so beloved by many in software world "design patterns" were largely originated from the Christpher Alexander's work on architectural patterns - and then read the first book in that series - "Timeless Way of Building" by Christopher Alexander - and see the roles of functionality and effectiveness in that art.

Then, the author of that paper holds awfully narrow view for computer software. It seems that he recognizes very specific-purpose software only - because he spoke about functionality as about a compact and well-defined thing, which does not need such artisitic features as fine balancing between contradictory criteria. He surely did not ask himself: what is the functionality for a text editor of MS Word kind, and how it differs - not in general, but in all important details - from the functionality of a text editor of, say, Emacs kind.

So I think that the quoted paper does not deserve further reading -;

Conference Calendar

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

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

2005

January 03-06	Software Technology Track of the 38th **Hawaii International Conference on System Sciences** (HICSS-38), Big Island of Hawaii, USA. Includes mini-tracks on: Distributed Object and Component-based Software Systems; Strategic Software Engineering; Adaptive and Evolvable Software Systems: Techniques, Tools, and Applications; etc.
January 12-14	32nd Annual ACM SIGPLAN-SIGACT **Symposium on Principles of Programming Languages** (POPL'2005), Long Beach, CA, USA. Topics include: fundamental principles and important innovations in the design, definition, analysis, transformation, implementation and verification of programming languages, programming systems, and programming abstractions.
January 21	1st **International Workshop on Abstract Interpretation of Object-oriented Languages** (AIOOL'2005), Paris, France. Topics include: Abstract Domains for the analysis of Object-oriented Languages, Semantics of Object-oriented Languages, Static Analysis of Object-oriented Languages, Verification, etc
☺ February 26-27	**Free and Open Source Software Developers' European Meeting** (FOSDEM'2005), Brussels, Belgium. Topics include: events on Ada and Open Source (tentative)
March 07-10	17th **Software Engineering Process Group Conference** (SEPG'2005), Seattle, Washington. Topics include: building quality products on cost and on schedule, establishing and maintaining continuous improvement efforts, etc.
March 09-11	11th **International Conference on Languages and Models with Objects** (LMO'2005), Bern, Switzerland. Topics include: object-oriented programming, components and distributed objects, object engineering, etc.
March 13-17	20th ACM **Symposium on Applied Computing** (SAC'2005), Santa Fe, New Mexico, USA. Includes tracks on: Embedded Systems (topics include: RTOS for embedded systems, Hardware/software support for real-time applications, Compilation strategies for performance enhancement vs. footprint control, Program parallelization for embedded systems, Case studies, etc.); Programming Languages (topics include: Compiling Techniques, Language Design and Implementation, Practical Experiences with Programming Languages, Program Analysis and Verification, etc.); Software Engineering (topics include: Software Reuse and Component-Based Development, Software Reliability and Software Fault Tolerance, Reengineering, Reverse Engineering and Software Maintenance, Real-Time Embedded Systems, etc.), etc.
March 14-18	4th **International Conference on Aspect-Oriented Software Development** (AOSD'2005), Chicago, Illinois, USA
March 21-23	9th **IEEE European Conference on Software Maintenance and Reengineering** (CSMR'2005), Manchester, UK. Theme: "Maintaining for Integration". Topics include: Evolution, maintenance and reengineering; Experience reports (successes and failures); Migration, wrapping and interfacing legacy systems; Reverse engineering of embedded systems; etc.
April 02-08	**Conference on Design, Analysis, and Simulation of Distributed Systems** (DASD'2005), San Diego, California, USA. Theme: "Making simulation and analysis successful through novel distributed system design, methodologies, and management". Topics include: Distributed Systems, Design and implementation approaches for complex systems using Petri nets, Distributed real-time

systems, Formal concepts and methods for validation and testing, High level architecture in distributed systems, etc.

April 02-10 **European Joint Conferences on Theory and Practice of Software** (ETAPS'2005), Edinburgh, Scotland, United Kingdom. Event includes: conferences from 4-8 April, 2005, satellite events on 2-3 and 9-10 April, 2005

- April 02-03 3rd **Workshop on Quantitative Aspects of Programming Languages** (QAPL'2005). Topics include: the design of probabilistic and real-time languages; of semantical models for such languages; the discussion of methodologies for the analysis of probabilistic and timing properties (e.g. security, safety, schedulability); applications to safety-critical systems; etc.
- April 02-10 8th **International Conference on Fundamental Approaches to Software Engineering** (FASE'2005). Topics include: Systematic approaches towards evolution management in large scale systems, continuous software engineering, and improvement and adaptation of legacy systems to altered requirements; Rigorous approaches to the design, testing, and maintenance of reactive, mobile, and distributed software systems; Integration of formal concepts and current best practices in industrial software development; Experience reports on best practices with development tools, software development kits, ...; etc.
- April 03 4th **International Workshop on Compiler Optimization Meets Compiler Verification** (COCV'2005). Topics include: optimizing and verifying compilation, and related fields such as translation validation, certifying and credible compilation, but also programming language design and programming language semantics.
- April 03 5th **Workshop on Language Descriptions, Tools and Applications** (LDTA'2005). Topics include: Program analysis, transformation, and generation; Formal analysis of language properties; Automatic generation of language processing tools.
- April 04-08 14th **International Conference on Compiler Construction** (CC'2005). CC is undergoing an expansion. Traditionally, CC has focused on compiler construction; CC now seeks to become a conference for research on a broader spectrum of programming tools, from refactoring editors to checkers to compilers to virtual machines to debuggers. Topics include: compilation techniques, incl. program representation and analysis, code generation and code optimization; run-time techniques, incl. memory management; compilation techniques for embedded code; compilers for parallel and distributed computing; compilation techniques for security and safety; design of novel language constructs and their implementation; software tools, incl. debuggers, profilers, code verifiers; etc.
- April 07-09 **International Symposium on Trustworthy Global Computing** (TGC'2005). Topics include: language-based security, reliability and business integrity, language concepts and abstraction mechanisms, type checkers, software principles to support debugging and verification, etc. Deadline for submissions: January 14, 2005 (papers)

☺ April 04-08 **International Parallel and Distributed Processing Symposium** (IPDPS'2005), Denver, Colorado, USA. Topics include: Applications of parallel and distributed computing; Parallel and distributed software, including parallel programming languages and compilers, operating systems, middleware, libraries, programming environments and tools; etc.

- ☺ April 04 10th **International Workshop on High-Level Parallel Programming Models and Supportive Environments** (HIPS'2005). Topics include: Concepts and languages for parallel and Grid programming (Component programming models, Refactoring of existing applications into components, Language and platform interoperability, Concurrent object-oriented programming, Extensions to traditional programming models, ...); Supportive techniques and runtime

environments (Compiler techniques, ...); Tools for high-level parallel programming; etc.

April 04 — 3rd **International Workshop on Parallel and Distributed Systems: Testing and Debugging** (PADTAD'2005). Topics include: optimizing and verifying compilation, and related fields such as translation validation, certifying and credible compilation, but also programming language design and programming language semantics.

☺ April 05-08 — 2nd Jahrestagung Fachbereich "Sicherheit - Schutz und Zuverlässigkeit" (2nd **Annual Conference on Safety and Security**), Regensburg, Germany. Event includes: special session "Informationssicherheit im Automobil", workshop "Privacy Respecting Incident Management", etc. Contributions welcome in English or in German. Topics include (in German): Vertrauenswürdige Softwarekomponenten; Zuverlässigkeit und Fehlertoleranz in Hardware- und Softwaresystemen; Formale Techniken, Modellierung, Spezifikation und Verifikation; Betriebssicherheit unter extremen Bedingungen; Standards und Normung; Sicherheitsevaluation und -zertifizierung; Kosten von Sicherheit; etc.

April 06-08 — **Software & Systems Quality Conferences** (SQS'2005), Düsseldorf, Germany. Event includes: ICSTEST International Conference on Software Testing, SQM, a congress focussing on Software Quality Management, and CSVHC Conference on Software Validation for Health Care.

April 10-13 — **The Conference for Software Practice Advancement** (SPA'2005), Wyboston, Bedfordshire, England. Topics include: Languages; Distributed, component-based development; Pervasive or embedded systems; Patterns and pattern languages; Comparative experience (what we have learned or can learn from other disciplines); Lessons learned/experience reports; etc.

April 11-13 — 12th **Annual European Concurrent Engineering Conference** (ECEC'2005), Toulouse, France. Topics include: engineering of embedded systems, specification languages, distributed computing environments, practical solutions, pitfalls and success stories, case studies, pilot projects and experiments, etc. Deadline for submissions: January 20, 2005 (final)

April 11-13 — 9th **International Conference on Empirical Assessment in Software Engineering** (EASE'2005), Keele University, UK. Topics include: Evaluation of products, components and services; Process and tool evaluation; Quality assessment; Software experiments, case studies and observational studies; etc.

April 27-29 — 5th **International SPICE Conference on Software Process Improvement and Capability dEtermination** (SPICE'2005), Klagenfurt, Austria.

May 02-06 — **International Conference on Practical Software Quality and Testing** (PSQT'2005 West), Las Vegas, Nevada, USA. Deadline for submissions: January 14, 2005 (papers, presentations)

☺ May 15-21 — 27th **International Conference on Software Engineering** (ICSE'2005), St Louis, Missouri, USA. Topics include: Software architectures and design; Software components and reuse; Software security; Software safety and reliability; Reverse engineering and software maintenance; Software economics; Empirical software engineering and metrics; Distribution and parallelism; Software tools and development environments; Programming languages; Object-oriented techniques; Embedded and real-time software; etc.

May 14-15 — 8th **International SIGSOFT Symposium on Component-Based Software Engineering** (CBSE'2005). Topics include: Static analysis and execution monitoring of system properties; Components for real-time, secure, safety critical and/or embedded systems; etc. Deadline for submissions: January 7, 2005

May 15-16 — 13th IEEE **International Workshop on Program Comprehension** (IWPC'2005). Topics include: Comprehension during large scale maintenance, reengineering, and evolution of existing systems; Reverse engineering for the purpose of program comprehension; etc. Deadline for submissions: January 21, 2005 (technical papers), February 4, 2005 (working sessions, tool demonstrations)

☺May 17	**Workshop on Architecting Dependable Systems** (WADS'2005). Topics include: all topics related to software architectures for dependable systems. Deadline for submissions: January 21, 2005
May 17	3rd **Workshop on Software Quality** (WoSQ'2005). Topics include: Software Product Evaluation and Certification; Tradeoffs in Quality during software development; Software Quality Education; Methods and Tools for Quality Assurance; Software Quality at different stages of the development lifecycle; Building quality into software products; Testing, Inspections, Walkthroughs and Reviews; etc. Deadline for submissions: February 21, 2005 (extended abstracts, position papers)
May 22-25	5th **International Conference on Computational Science** (ICCS'2005), Atlanta, USA. Theme: "Advancing Science through Computation". Topics include: Parallel and Distributed Computing, etc.
May 23-25	9th **Brazilian Symposium on Programming Languages** (SBLP'2005), Recife, PE, Brazil. Topics include: programming language design and implementation, formal semantics of programming languages, theoretical foundations of programming languages and teaching programming languages, etc. Deadline for submissions: February 28, 2005
May 27-June 01	3rd **International Software Development Conference** (SWDC'2005), Reykjavik, Iceland. Topics include: Project success and failure analysis; Software project risk management; Software process improvement; etc. Deadline for submissions: February 21, 2005
☺ May 30-June 02	**DAta Systems In Aerospace** (DASIA'2005), Edinburgh, Scotland, UK.
June 06-09	5th **International Conference on Application of Concurrency to System Design** (ACSD'2005), St Malo, France. Topics include: Correct-by-construction design methods and integration of verification techniques with the design process; etc. Deadline for submissions: February 4, 2005 (tool demonstrations)
June 06-10	25th **International Conference on Distributed Computing Systems** (ICDCS'2005), Columbus, Ohio, USA. Sponsored by The IEEE Computer Society Technical Committee on Distributed Processing. Topics include: Fault Tolerance & Dependability, Middleware, Real-time & Embedded Systems, Security, Formal Verification, etc.
June 13-17	17th **Conference on Advanced Information Systems Engineering** (CAiSE'2005), Porto, Portugal. Topics include: Model and Software Reusability, Distributed and Open Architectures, Languages for IS, etc. Deadline for submissions: February 26, 2005 (posters), February 28, 2005 (Doctoral Consortium Papers)
☺ June 14-17	34th **International Conference on Parallel Processing** (ICPP'2005), Oslo, Norway. Topics include: Compilers and Languages, Programming Methodologies, Tools, Parallel Embedded Systems, etc. Deadline for submissions: January 3, 2005
June 15-17	7th IFIP **International Conference on Formal Methods for Open Object-based Distributed Systems** (FMOODS'2005), Athens, Greece. In conjunction with DAIS'2005 (Distributed Applications and Interoperable Systems). Topics include: Formal techniques for specification, design or analysis; Verification, testing and validation; Component-based design; Type systems for programming languages; Formal models for security; Applications and experience, carefully described; etc. Deadline for submissions: January 14, 2005 (abstracts), January 21, 2005 (full papers)
June 15-17	5th IFIP WG 6.1 **International Conference on Distributed Applications and Interoperable Systems** (DAIS'2005), Athens, Greece. Deadline for submissions: February 1, 2005 (full papers), February 15, 2005 (work-in-progress papers).
June 16-20	10th IEEE **International Conference on the Engineering of Complex Computer Systems** (ICECCS'2005), Shanghai, China. Topics include: Tools, environments, and languages for complex systems; Formal methods for developing complex systems; Software and system development processes for complex systems; Software review, inspection, and testing; Formal proof and model checking; Human factors and collaborative aspects; Interoperability and

standardization; Systems and software safety and security; Industrial automation, embedded and/or real time systems; etc.

June 18-23 — 6th **International Conference on eXtreme Programming and Agile Processes in Software Engineering** (XP'2005), Sheffield, UK. Topics include: Case studies, experiments and practioner's reports; Scalability issues; Refactoring and continuous integration; Use of SW development tools and environments; etc. Deadline for submissions: January 16, 2005 (papers), March 1, 2005 (tutorials, workshops, panels and activities, PhD Symposium, posters)

♦ June 20-24 — 10th **International Conference on Reliable Software Technologies - Ada-Europe'2005**, York, UK. Sponsored by Ada-Europe, in cooperation with ACM SIGAda (approval pending). Deadline for submissions: January 10, 2005 (industrial presentations)

June 27-29 — 10th **Annual Conference on Innovation and Technology in Computer Science Education** (ITiCSE'2005), Monte de Caparica, Portugal. Deadline for submissions: February 14, 2005 (tips & techniques summaries, posters, demos), April 18, 2005 (working group membership application)

June 27-30 — 2005 **International Multiconference in Computer Science and Computer Engineering** (IMCSE'2005), Las Vegas, Nevada, USA. Deadline for submissions: February 16, 2005 (papers)

☺ June 27-30 — **International Conference on Programming Languages and Compilers** (PLC'2005). Deadline for submissions: February 16, 2005 (papers)

June 27-30 — **International Conference on Software Engineering Research and Practice** (SERP'2005). Topics include: Formal methods in software engineering, Software engineering and high assurance systems, Software maintenance, Component-based software engineering, Quality-based software engineering, Real-time software engineering, Critical systems, Verification and validation, Software testing, Quality management, Object-oriented software engineering, Case studies, etc. Deadline for submissions: February 26, 2005

July 06-10 — 17th **International Conference on Computer-Aided Verification** (CAV'2005), Edinburgh, Scotland, 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. Deadline for paper submissions: January 21, 2005

July 11-15 — 32nd **International Colloquium on Automata, Languages and Programming** (ICALP'2005), Lisbon, Portugal. Topics include: Parallel and Distributed Computing; Principles of Programming Languages; Formal Methods; Program Analysis and Transformation; Specifications, Verifications and Secure Programming; etc. Affiliated Workshops on July 9-10 and 16-17, 2005. Deadline for submissions: February 13, 2005 (extended abstracts)

July 11-15 — 1st **International Conference on Open Source Systems** (OSS'2005), Genova, Italy. Topics include: Introduction of OSS in companies and Public Administrations, Empirical analysis of OSS, Case studies and experiments, etc. Deadline for submissions: January 15, 2005 (extended abstracts), January 31, 2005 (tutorials, workshops, panels, demos), March 1, 2005 (abstracts for Open Educational Symposium, research plans for PhD symposium)

July 17-20 — 24rd Annual ACM SIGACT-SIGOPS **Symposium on Principles of Distributed Computing** (PODC'2005), Las Vegas, Nevada, USA. Topics include: all areas of distributed systems. Deadline for submissions: February 7, 2005

July 18-22 — 13th **International Symposium of Formal Methods Europe** (FM'2005), Newcastle upon Tyne, UK. Topics include: introducing formal methods in industrial practice (technical, organizational, social, psychological aspects); reports on practical use and case studies (reporting positive or negative experiences); tool support and software engineering; environments for formal methods; etc. Deadline for submissions: January 24, 2005 (papers), March 07, 2005 (workshops, tutorials), May 09, 2005 (tools exhibition, demonstrations)

July 25-28 — 29th Annual **International Computer Software and Applications Conference** (COMPSAC'2005), Edinburgh, Scotland, UK. Theme: "High Assurance Software Systems" Topics include: Dependable service provision, Trustworthy software, Software safety, Software

	fault tolerance, High performance software, Component-based software, Design patterns, Software certification, Software standards, Software engineering education, Embedded systems, Middleware systems, Automotive telematics, etc. Deadline for submissions: February 15, 2005 (regular and workshop papers), March 21, 2005 (fast abstract submissions)
☺ July 25-29	19th **European Conference on Object-Oriented Programming** (ECOOP'2005), Glasgow, Scotland, UK. Topics include: Concurrent, real-time and parallel systems; Design patterns; Distributed systems; Frameworks and software architectures; Language design and implementation; Programming environments; Adaptability; Formal methods; Software evolution; etc. Deadline for submissions: March 18, 2005 (PhD workshop submissions), May 6, 2005 (demos, posters, exhibitions)
August 29-September 02	13th IEEE **International Requirements Engineering Conference** (RE'2005), Paris, France. Deadline for submissions: February 7, 2005 (abstracts), February 14, 2005 (papers), March 11, 2005 (workshops, tutorials, panels), April 28, 2005 (doctoral symposium, posters, research demonstrations)
☺ August 30-Sept. 02	11th **International Conference on Parallel and Distributed Computing** (Euro-Par'2005), Lisboa, Portugal. Topics include: Support Tools and Environments; Scheduling and Load Balancing; Compilers for High Performance; Distributed Systems and Algorithms; Parallel Programming: Models, Methods, and Languages; etc. Deadline for submissions: January 31, 2005 (full papers)
August 30 – Sept. 03	31st **EUROMICRO Conference on Software Engineering and Advanced Applications** (EUROMICRO'2005), Lisboa, Portugal. Topics include: Component-Based Software Engineering, Software Process and Product Improvement, Component Models for Dependable Systems, Value-based Software Engineering, etc. Deadline for submissions: March 1, 2005 (papers)
☺ September 13-16	**International Conference on Parallel Computing** 2005 (ParCo2005), Malaga, Spain. Topics include: applications; software engineering methodologies, methods and tools for developing and maintaining parallel software, incl. parallel programming models and paradigms, development environments, languages, compiling and run-time tools; etc. Deadline for submissions: January 31, 2005 (abstracts, mini-symposia proposals), July 31, 2005 (draft full papers)
September 19-22	11th **International Software Metrics Symposium** (Metrics'2005), Como, Italy. Topics include: Effort and cost estimation; Defect rate and reliability prediction; Quality Assurance; Empirical studies of global software development projects, open source software projects, agile development projects; etc. Deadline for submissions: February 23, 2005 (abstracts), March 15, 2005 (technical papers), April 1, 2005 (workshops, dissertation forum), May 30, 2005 (industry track)
September 19-23	9th International IEEE **Enterprise Distributed Object Computing Conference** (EDOC'2005), Enschede, The Netherlands. Deadline for submissions: March 4, 2005 (abstracts), March 18, 2005 (papers, workshops)
September 25-30	21st IEEE **International Conference on Software Maintenance** (ICSM'2005), Budapest, Hungary. Topics include: issues related to maintaining, modifying, enhancing, and testing operational systems, and designing, building, testing, and evolving maintainable systems. Deadline for submissions: March 21, 2005 (research papers), April 30, 2005 (industrial applications, panels, tool demonstrations, dissertation forum, tutorials)
October 02-07	8th **International Conference on Model Driven Engineering Languages and Systems** (MoDELS'2005), Montego Bay, Jamaica. Formerly the UML series of conferences. Topics include: Model-driven development methodologies, approaches, and languages; Empirical studies of modeling and model-driven development; Tool support for any aspect of model-driven development or model use; Semantics of modeling languages; etc. Deadline for submissions: March 21, 2005 (experience and scientific abstracts), April 4, 2005 (experience and scientific full papers), May 6, 2005 (workshops), June 6, 2005 (tutorials), TBA (doctoral symposium, tools and exhibits, posters and demos)
♦ November 13-17	2005 **ACM SIGAda Annual International Conference** (SIGAda'2005), Atlanta, Georgia, USA.
December 10	Birthday of Lady Ada Lovelace, born in 1815. Happy Programmers' Day!

Real-Time Java™ for Ada Programmers

Benjamin M Brosgol

AdaCore, 79 Tobey Road, Belmont, MA 02478, USA; email: brosgol@adacore.com

Abstract

This full-day tutorial was an in-depth introduction to the Real-Time Specification for Java ("RTSJ"), a class library (and JVM constraints) designed to add real-time behaviour and predictability to the Java platform. The RTSJ comprises several kinds of features: a scheduling framework and base scheduler, memory management not subject to Garbage Collection, synchronization / priority inversion control, asynchronous event handling, asynchronous transfer of control, time-related classes, and low-level features. Several implementations are in progress, and work is underway on designing a subset that is appropriate for safety-critical applications.

Keywords: real-time, Java, priority inversion, concurrency, garbage collection, asynchrony.

1 Background and Introduction

During the late 1990s Java's growing popularity as an alternative to C++ led to several workshops, organized by the National Institute for Standards and Technology (NIST) in the US, to assess the Java platform's viability for real-time applications. These culminated in a January 1999 meeting in San Diego hosted by Aonix and attended by representatives of Sun Microsystems, hardware vendors, RTOS suppliers, researchers, and users. Two obvious major technical problems were noted: the unpredictability caused by automatic garbage collection, and the lack of deterministic scheduling for threads. It was felt that these were solvable issues. A set of requirements was formulated by NIST [1], and almost immediately two parallel efforts got underway to design a compliant specification: one under the auspices of Sun Microsystems' Java Community Process (JCP), and the other by a group known as the J-Consortium (who had some issues with the JCP licensing terms).

The Sun-sanctioned effort was a Java Specification Request, JSR-001, that came to be known as the *Real-Time Specification for Java* ("RTSJ"). This has gone through several iterations: a pre-release in June 2000 led by Greg Bollella (then at IBM); an initial JCP-approved version in November 2001 let by Peter Haggar (IBM); and a JCP-approved maintenance release in August 2004 led by Peter Dibble (Timesys). As required by JCP rules, the specification is accompanied by a Reference Implementation and Technology Compatibility Kit, both from Timesys. The RTSJ has been the subject of an initial reference book by Bollella *et. al.* [2], and textbooks by Dibble [3] and Wellings [4]. Additional information can be found at the website www.rtj.org.

The J-Consortium published an alternative approach for real-time Java in Fall 2000, the "Core Extensions" document (Nilsen [5]). Despite a number of innovative aspects (such as features that help predict Worst Case Execution Time) the Core Extensions have not yet been implemented.

The tutorial focused on the RTSJ rather than the Core Extensions, since the RTSJ has much higher likelihood of being implemented and used in practice.

2 Java and Real-Time Programming

In order to support real-time development, a programming language needs to meet requirements in several areas:

- Appropriate reliability-oriented facilities, such as strong typing and encapsulation.
- A concurrency mechanism with an adequate range of priorities, well-defined scheduling semantics, safe/efficient mutual exclusion that allows management of priority inversion, and support for common idioms such as periodic activities.
- Predictability of time and space utilization, particularly in the area of memory management. Predictability also applies more generally to program execution: language semantics should be deterministic and should avoid implementation dependence.
- Asynchronous event handling and also timeouts / thread abort ("asynchronous transfer of control").
- Adequate functionality for clocks / timers and low-level support.

Performance is also a goal, although "real time" is not the same as "real fast". Indeed, there is sometimes a trade-off between predictability and performance or throughput. An interesting contrast between the RTSJ and the Core Extensions is that the latter strove explicitly to attain run-time efficiency comparable to traditional real-time approaches.

Although Java meets many of the general reliability-oriented criteria and has deterministic semantics for its sequential features, it has some obvious deficiencies. In the concurrency area, the priority range (10 values) is too narrow, and these values do not necessarily map to distinct priorities on the underlying native operating system. The scheduling semantics are (intentionally) loose; indeed, the

Java Language Specification [6] states that there is no "guarantee that the highest priority thread will always be running, and thread priorities cannot be used to reliably implement mutual exclusion". Priority may or may not be used in the selection of which thread to run when a lock is released, or which thread in a wait set to make ready when a notify occurs. Unbounded priority inversion may occur.

Another major deficiency, which some may feel is a fatal flaw, is Java's need for garbage collection. Despite the advances in "real-time GC", this topic remains a research area and is not in mainstream use. Thus any attempt to adapt Java to meet real-time requirements must address the issue of predictable memory management.

Asynchrony support in Java is weak. There is no special mechanism for asynchronous event handling; the programmer needs to realize such handlers as regular threads, with the resulting loose semantics for scheduling. Asynchronous transfer of control was simply not well thought out in the original Java specification. The main methods for realizing this behaviour – stop() and destroy() – have been deprecated because of their dangers, and in any event destroy() has never been implemented.

The other functional requirements noted above – support for time and timers, and access to low-level hardware features – are also insufficient or absent.

All of these issues presented challenges in the design of the RTSJ.

3 RTSJ Overview

The contents of this section are based on Brosgol & Wellings [7].

The RTSJ provides a flexible scheduling framework based on the Schedulable interface and the Thread subclass RealtimeThread that implements this interface. The latter class overrides various methods with versions that add real-time functionality, and supplies new methods for operations such as periodic scheduling. The Schedulable interface is introduced because certain schedulable entities (in particular, handlers for asynchronous events) might not be implemented as threads.

The RTSJ mandates a default POSIX-compliant preemptive priority-based scheduler that supports at least 28 distinct priority levels, and that enforces Priority Inheritance to manage priority inversions. The implementation can provide other schedulers (e.g., Earliest Deadline First) and priority inversion control policies (e.g., Priority Ceiling Emulation).

To deal with Garbage Collection issues, the RTSJ provides various *memory areas* that are not subject to Garbage Collection: "immortal memory", which persists for the duration of the application; and "scoped memory", which is a generalization of the run-time stack. Restrictions on assignment prevent dangling references. The RTSJ also provides a NoHeapRealtimeThread class; instances never reference the heap, may preempt the Garbage Collector at any time (even when the heap is in an inconsistent state), and thus do not incur GC latency except in specialized circumstances described below.

Java's asynchrony issues are addressed through two main features. First, the RTSJ allows the definition of asynchronous events and asynchronous event handlers – these are basically a high-level mechanism for handling hardware interrupts or software "signals". Secondly, the RTSJ extends the effect of Thread.interrupt to apply not only to blocked threads, but also to real-time threads and asynchronous event handlers whether blocked or not.

The RTSJ supports absolute and relative high-resolution time, as well as one-shot and periodic timers. It also provides several classes for low-level programming. "Peek and poke" facilities for integral and floating-point data are available for "raw memory", and "physical memory" may be defined with particular characteristics (such as flash memory) and used for general object allocation.

4 Scheduling

The RTSJ combines a priority-oriented scheduling algorithm (the *base scheduler*) with an extensible scheduling framework, support for on-line admission control and feasibility analysis, and a unified treatment of real-time threads and asynchronous event handers. A real-time thread is an instance of the RTSJ's RealtimeThread class. Asynchronous event handlers will be described below.

The base scheduler adds deterministic semantics (FIFO within priority) to the rules for choosing which ready thread to run, which wait set member to awaken after a notify(), and which "stalled" thread to unblock when a lock is released.

The extensible scheduling framework allows implementers to provide additional schedulers such as Earliest Deadline First or Round Robin.

Real-time threads and asynchronous event handlers are termed *schedulable objects*. Constructors allow the establishment of various sorts of "parameters", including scheduling parameters (priority), release parameters (cost, deadline, cost overrun handler, and deadline miss handler), and memory parameters (control over GC pacing). Support for cost overrun detection is optional, given the implementation difficulty on some platforms.

The RealtimeThread class is an extension of java.lang.Thread that implements the Schedulable interface, itself an extension of Runnable. Besides the run() method, there are a number of methods related to feasibility analysis.

Here is an example of a periodic real-time thread:

```
public class MyPeriodic extends RealtimeThread{

  public MyPeriodic( int priority,
                     int periodMillis,
                     int costMillis,
                     int deadlineMillis ){
    super( new PriorityParameters( priority ),
                  // SchedulingParameters
           new PeriodicParameters(
```

```
            // ReleaseParameters
            null, // 1st release is at start
            new RelativeTime( periodMillis, 0 ),
            new RelativeTime( costMillis, 0 ),
            new RelativeTime( deadlineMillis, 0 ),
            null, // No cost overrun handler )
            null) // No deadline miss handler
    );
  }

  public void run(){
    while (true) {
      … // Work done during each release
      RealtimeThread.waitForNextPeriod();
        // Block till next release
    }
  }

  MyPeriodic mp = new MyPeriodic( 30, 100, 20, 100 );
  // priority == 30, period ==100, cost ==20,
  // deadline == 100

  mp.start();  // Triggers initial release for mp
```

The arguments to the constructor for MyPeriodic establish the priority, period, cost and deadline. These are in turn passed to one of the superclass constructors via corresponding parameters objects. The behaviour of the periodic real-time thread is captured in the run() method, which illustrates the RTSJ style for realizing periodicity: invoking the waitForNextPeriod method. The implementation will arrange the next release automatically.

5 Memory Management

The RTSJ defines a general concept of a *memory area*, which is a region used for object allocation. The Java *heap* is one such memory area, represented by a singleton class. Another memory area is *immortal* memory, also represented by a singleton class; objects in this area are never reclaimed or relocated.

Supplementing the heap and immortal areas are *scoped* memory areas that are constructed under program control. As suggested by the terminology, a scoped area has a limited lifetime: it is reset (and all contained objects finalized) when it is no longer referenced by the program. Usage restrictions allow this to be implemented with a reference counting scheme – one count per area, not one per allocated object. Consistent with the Java language principles, there is no explicit free operation. Assignments are restricted to prevent dangling references: a reference to an object in a scoped area is not allowed to be assigned to a field in an object that resides in the heap, in immortal memory, or in an outer scope. This rule is enforced by a run-time check but could be optimized out by a clever compiler.

A memory area can be used for a single object allocation, but more typically it is "entered" and then used for all object allocations during the "closure" of a method invocation (i.e., by the method itself or other methods and constructors directly or indirectly invoked). A memory area can also be passed to a constructor for a schedulable object, in which case it is used during the closure of the schedulable object's run() method.

The user can construct real-time threads that never access the heap and that thus can pre-empt the Garbage Collector even when the GC is not at a "safe" interruption point (i.e., when the heap is in an inconsistent state). Such real-time threads, instances of the NoHeapRealtimeThread class, may only reference immortal and scoped memory areas.

The rules for scoped memory are somewhat subtle, and the workaround to the assignment restrictions typically involves heavy use of immortal memory.

The following example shows one use of scoped memory. The constructor for m creates a scoped area of size 100K bytes. At each iteration of the loop, the area is used for allocating objects that are created during execution of r.run(). At the end of each iteration each object in the area is finalized, and the area is emptied.

```
void foo(){
  LTMemory m = new LTMemory( 100000, 100000 );
  Runnable r = new Runnable(){
                        public void run(){...}
                      };
  while ( someCondition() ){
    m.enter(r);
    }
}
```

6 Synchronization

The RTSJ provides a general model for synchronization control (to manage priority inversions), and specific semantics that apply to the base scheduler.

Every object has a *monitor control policy*, assigned either implicitly or explicitly. The default (implicit) policy is Priority Inheritance ("PI"), but this can be changed at system startup. (More specifically, the RTSJ defines an abstract class MonitorControlPolicy and a singleton subclass PriorityInheritance.) If an object governed by priority inheritance is locked by a thread t1, and another thread t2 attempts to lock the object, then t1's active priority will be increased to that of t2. This is applied recursively if t1 is stalled waiting for an object that is currently locked by some other thread t3: the priority of t3 will be raised to that of t2.

The RTSJ also defines a MonitorControlPolicy subclass PriorityCeilingEmulation ("PCE") corresponding to the "highest lockers" / priority ceiling emulation policy. An object governed by an instance of this class has a ceiling that needs to be set (by the programmer) at least as high as the highest active priority of any thread that could lock the object. When a thread attempts to lock an object governed by such a policy a check is made to ensure that the thread's priority does not exceed the object's ceiling. If this condition is met then either the thread's active priority is boosted to the ceiling and the thread acquires the lock (if the object is unlocked) or the thread is queued (if the object is locked). If the ceiling check fails, an exception is thrown.

The combination of Priority Inheritance and PCE results in some subtle interactions. For example, while a thread t1 is holding locks on a PCE-governed object obj1 and a PI-governed object obj2, a high priority thread t2 may attempt to lock obj2. This would ordinarily result in t1's inheriting t2's priority. However, if t2's priority exceeds obj1's

ceiling, then the priority inheritance would cause t1 to have a ceiling violation. Since throwing an exception in t1 would be asynchronous, the rules instead require the exception to be thrown in t2.

The PriorityCeilingEmulation policy is optional, since the RTSJ authors felt that it was not as widely supported as priority inheritance in practice.

Unlike an Ada task executing a protected operation, a thread is allowed to block when in synchronized code, and thus the "no-lock" optimization for Ada's protected objects does not apply in the RTSJ. An implementation can provide a lock-free version of PriorityCeilingEmulation as an optimization, but there are a number of subtleties that will make this optimization difficult.

When a thread suffers a loss of inherited priority it goes to the ready queue for its new active priority; the RTSJ recommends that it be placed at the head, but this is not a requirement.

An object's monitor control policy can be changed dynamically; it may be governed by priority inheritance at one point, and priority ceiling emulation at other times. It can also be governed by PCE policies with different ceilings at different times.

```
class Position{
  private double x, y;

  Position(double x, double y, int ceiling){
    this.x=x; this.y=y;
    MonitorControl.setMonitorControl(
     this,
     PriorityCeilingEmulation.instance(ceiling));
  }

  synchronized void setXY(double x, double y){
    this.x=x; this.y=y;
  }
  synchronized double[] getXY() {
    return new double[2]{x, y};
  }
}

class Sensor extends RealtimeThread{
  Sensor(Position p, int priority){...}
  ...
}

class Reporter extends RealtimeThread{
  Reporter(Position p, int priority){...}
  ...
}

class Test{
  public static void main(String[] args){
    Position p = new Position(0.0, 0.0, 16);
    Sensor   s = new Sensor(p, 15);
    Reporter r = new Reporter(p, 10);
    s.start(); r.start();
    ...
  }
}
```

In this example, each Position instance is governed by PCE and is given a ceiling a construction time. The main method creates Sensor and Reporter real-time threads at priority 15 and 10 respectively, and a Position object (shared by the two real-time threads) at ceiling 16.

Communication between a no-heap realtime thread and a heap-using thread raises an interesting issue. If they attempt to synchronize on an object then the no-heap thread may incur GC-induced latency. (This can arise if the heap-using thread holds a lock on an object that the no-heap realtime thread attempts to synchronize on, since the heap-using thread may cause the GC to run.)

In order to help applications avoid this problem, the RTSJ supplies two "wait-free queue" classes. The WaitFreeWriteQueue is typically used by one writer (a no-heap realtime thread) and any number of readers (heap-using threads). The write() operation is non-blocking and non-synchronized; read() is a blocking, synchronized operation. The WaitFreeReadQueue is analogous. Communication between a no-heap realtime thread and a heap-using thread can use these queues and thus avoid the need to synchronize on the same object.

7 Asynchrony

The RTSJ offers two types of asynchrony features: asynchronous event handling, and asynchronous transfer of control.

7.1 Asynchronous Events and their Handlers

A real-time program often has to cope with events that are triggered asynchronously, either from hardware interrupts or from software threads. The RTSJ uses the standard Java "listener" pattern to model this situation, but extended for applicability to real-time systems.

One part of the model is the AsyncEvent class, which has methods that allow event handlers to be registered, and also a method – fire() – that reflects the occurrence of the event.

The other part of the model is the AsyncEventHandler class. This class has constructors similar to RealtimeThread: thus asynchronous event handlers ("AEH"s) have scheduling parameters, release parameters, etc. The class has a method – handleAsyncEvent() – that will need to be overridden on subclassing to supply the necessary logic for event handling. AsyncEventHandler also has a run() method, but this contains logic supplied by the implementation and is not overridden.

In a simple scenario, when an event e is fired, all AEHs registered to handle e are released and are then scheduled based on their parameters. When an AEH is executed, its run() method is invoked, which in turn invokes the handleAsyncEvent() method.

In a more complicated situation, an event may be "bursty": when fired, it may already have an AEH running, having been scheduled in response to a previous occurrence. In this situation there is not another scheduling of the handler. Instead, the implementation needs to keep track of the number of pending event firings via a "fire count"; the handler will invoke handleAsyncEvent() repeatedly as long as the fire count is non-zero. Methods are available that allow the handler to reset the fire count, since in some situations the iterated invocation of handleAsyncEvent() might not be desirable.

An AEH needs a thread of control to execute its run() and handleAsyncEvent() methods. The RTSJ allows the user to

specify that an AEH is bound to a dedicated thread: this is accomplished by the user subclassing BoundAsyncEventHandler. Otherwise an AEH may share a thread with other AEHs.

Here is an example of asynchronous event handling:

```
class Handler extends AsyncEventHandler{
  public void handleAsyncEvent(){
    System.out.println("Event handled");
  }
class Example{
  AsyncEvent e = new AsyncEvent();
  Handler h = new Handler();
  e.addHandler(h);
  e.fire();
}
```

When event e is fired, handler h is released. Its scheduling and release parameters (here the defaults are implied) determine when it runs, at which point it simply displays a string.

7.2 Asynchronous Transfer of Control

Asynchronous Transfer of Control ("ATC") is a transfer of control within a thread, triggered not by the thread itself but rather from some external source such as another thread or an interrupt handler. It is a controversial feature. On the one hand it is difficult to define, complicated to implement (especially so if efficiency is important) and hard to use correctly; the possibility of ATC makes program testing and analysis complex. Nevertheless, there are several common situations in practice for which ATC works better (i.e. has lower latency) than polling. These include timing out on a computation, and terminating a thread (e.g. as part of a mode change).

The ATC design in the RTSJ was based on several design principles, including the following:

- No syntactic extension. Any functionality had to be realized by method calls versus new statements.
- Non-interruptibility as the default. Code that is susceptible to asynchronous interruption must explicitly indicate its permission.
- Deferral in critical sections. Code that must execute to completion (specifically, synchronized code) is not susceptible to ATC.

The ATC approach in the RTSJ combines low-level "building blocks" with some higher level mechanisms. At its foundation are several main elements:

- The complementary concepts of *asynchronously-interruptible* ("AI") and *ATC deferred* code
- A class AsynchronouslyInterruptedException ("AIE"), a subclass of InterruptedException
- A generalization of Thread.interrupt() in regular Java when applied to a real-time thread.
- A controlled form of asynchronous exception throwing

The only code that is asynchronously interruptible is that contained textually within a method or constructor that includes AIE on its throws clause, but that is not within ATC-deferred code. Synchronized statements and methods, and also methods and constructors that lack a throws AIE clause, are ATC-deferred. Further, an AI method or constructor will be ATC deferred if invoked from a regular (i.e., non real-time) thread.

An ATC request always involves, either explicitly or implicitly, a target real-time thread t and an AIE instance aie. For example, the method call t.interrupt() posts an ATC request to t, but the AIE instance (the system-wide "generic" AIE) is implicit. Because of the special rules for AIE exception propagation, t.interrupt() would typically be used as a way to terminate t. It has an effect not simply when t is blocked (which are regular Thread semantics) but also when t is executing AI code. Here's an example of how to express a real-time thread's logic so that it may be terminated asynchronously by another thread:

```
class AbortableVictim extends RealtimeThread{
  private void body() throws AIE{
    ... // abortable here
  }
  public void run(){
    ... // not abortable here
    try{
      body(); // abortable here
    }
    catch( AIE e ){
      ... // response to abort
    }
    finally{
      ... // unconditional pre-shutdown actions
    }
  }
}
```

If another thread has a reference to an AbortableVictim v, then the method call v.interrupt() will cause an ATC in v if v is executing the body() method, since body() has a "throws AIE" clause.

Posting an ATC request to a real-time thread t basically involves throwing an AIE instance asynchronously at t, but subject to several restrictions and special rules:

- The throwing of the exception is deferred until t is executing AI code. This avoids the problem of ATC from synchronized code, which could leave an object inconsistent.
- Handling an exception thrown as a result of ATC does not automatically prevent the exception from propagating. This rule prevents a real-time thread from defeating a termination ATC by handling the exception.
- An ATC request posted to a real-time thread has no effect if performed too early (before the target thread has initiated an interruptible method invocation associated with the request) or too late (after such an invocation has completed. This rule prevents an asynchronous exception from being thrown in code not prepared to handle it.

The last rule is captured in the Interruptible interface and the doInterruptible and fire methods of AIE. The basic style is as follows:

- The target thread invokes aie.doInterruptible(i) where i is an instance of Interruptible. The effect is to invoke i.run() synchronously
- The relevant methods to implement in i are run(), which should have a "throws AIE" clause, and interruptAction.
- The run() method is the interruptible logic, and interruptAction is to be invoked when run receives an ATC
- Another thread needs to invoke aie.fire() to post the ATC request to the target thread

The RTSJ supplies an AIE subclass, Timed, which automates the firing of the AIE and serves as a basis for the RTSJ idiom for expressing a timeout.

The ATC rules are complicated and sometimes counterintuitive. For example, although it might seem that finally clauses should be ATC-deferred (analogous to Finalize being abort-deferred in Ada), this would cause problems, since finally clauses are not manifest in the bytecodes of the classfile. One of the constraints on the RTSJ was that the spec should not force compilers to be changed; requiring special handling for finally clauses would violate this condition. Thus an ATC is possible from a finally clause in an AI method, which complicates the style for arranging non-interruptible cleanup.

8 Time and Timers

The RTSJ supplies an abstract HighResolutionTime class, with non-abstract subclasses for AbsoluteTime and RelativeTime. These classes support time measured in milliseconds and nanoseconds, relative to some clock (by default a monotonically non-decreasing real-time clock). Various methods allow converting between absolute and relative time, and perfoming relevant arithmetic operations (e.g., adding millis and nanos to an absolute or relative time value). The RTSJ also supplies a RationalTime class, as a subclass of RelativeTime. This class was intended to allow expressing frequency requirements (e.g., 17 times per second) but has been deprecated because of several anomalies that it caused.

The RTSJ defines an abstract class Timer as a subclass of AsyncEvent,. It also provides OneShotTimer and PeriodicTimer as subclasses of Timer. The AsyncEventHandler mechanism may be used to realize the event handling logic.

9 Low-Level Features

Two basic mechanisms are provided for dealing with low-level issues: "raw" memory, and "physical" memory.

The RTSJ supplies two classes for raw memory: RawMemoryAccess and RawMemoryFloatAccess. The former contains methods that allow the setting and retrieval of integral values (byte, char, int, long) from specified addresses. The latter contains all of these methods plus ones to set and retrieve floating point values (float, double). (The reason for the two classes is that some implementations of the RTSJ might be on platforms that lack floating-point arithmetic.) Neither allows the setting or retrieval of references, since that would obviously defeat Java's strong typing and interfere with Garbage Collection.

Physical memory is part of the memory area mechanism and allows the user to specify immortal or scoped areas that have distinguished characteristics (e.g., flash memory or memory-mapped I/O). It can be used for object allocations in the same way as other memory areas.

10 Real-Time Java in Practice

This section highlights some current implementations and other RTSJ-related developments.

- Reference Implementation

The Reference Implementation ("RI") is part of the JSR "product" required by Sun's Java Community Process. The current RI had previously been marketed by Timesys as their commercial jTime offering. URL: www.timesys.com

- jRate (Java Real-Time Extension)

This effort at Washington University at St. Louis (US) is an extension of the GNU GCJ front-end and run-time library, to support the RTSJ.
URL: www.cs.wustl.edu/~corsaro/jRate

- OVM (OpenVM)

This is an open-source framework for language run-time systems, with emphasis on an RTSJ-compliant JVM. Participants are Purdue, State Univ. of New York in Oswego, and Univ. of Maryland (US); and DL Tech (Australia). It is sponsored by DARPA (US). URL: www.ovmj.org

- Jamaica VM

This is a commercial implementation of the RTSJ by aicas, using F. Siebert's Garbage Collection algorithm [8]. URL: www.aicas.com

- HIDOORS Project (High-Integrity Distributed Object-Oriented Realtime Systems.)

Partially funded by the European Commision, this consortium comprises organizations in Germany (FZI, R.O.S.E. Informatik, aicas, EADS), Sweden (Linköpings Univ.), France (Aonix), and Portugal (Skysoft). Their goal as stated on their website is to provide "the full functionality of the modern programming language Java for the development of distributed, realtime and safety critical systems and to provide a powerful environment of tools that support modeling, analysis, and proof of correctness of systems developed in Java". URL: www.hidoors.org

- HIJA Project (High-Integrity Java)

This consortium comprises The Open Group, aicas, Aonix, Bellstream, Fiat Research Centre (CRF), FZI, Thales-Avionics, Telecom Italia, Trialog, Universität Karlsruhe, Universidad Politécnica de Madrid (DIT-UPM), and University of York. According to their website, "the main technical objective of HIJA is to demonstrate that Java

technology can form an appropriate Architecturally Neutral, high-integrity Real-Time System."
URL: www.hija.info

- The Open Group

Under the auspices of The Open Group's Real-Time and Embedded Systems Forum, work got underway in mid-2003 to investigate a safety-critical Java profile that would be based on the RTSJ and developed under the Java Community Process. This effort is in progress, although as of late 2004 an official JSR has not yet been initiated.
URL: www.opengroup.org

11 Conclusions

Although "real-time Java" may sound like a contradiction, the RTSJ has attempted to address the deficiencies of regular Java in a technical credible manner. The combination of a priority-based default scheduler and user-assignable monitor control policies provide deterministic scheduling semantics and avoid unbounded priority inversions. Scoped and immortal memory areas do not suffer garbage collection, and no-heap realtime threads need not incur GC latency. A general scheduling framework provides extensibility and also supports on-line feasibility analysis. The unifying concept of a schedulable object allows both real-time threads and asynchronous event handlers to be analyzed consistently. And a facility for asynchronous transfer of control solves regular Java's ATC anomalies by borrowing the Ada concept of an abort-deferred operation; the resulting mechanism deals with common cases such as timeouts and thread termination. All this is achieved within the Java framework (no new syntax) and within the Java design philosophy (no explicit memory freeing).

On the other hand, the RTSJ is not yet a proven technology. It is forward-thinking and ambitious, but some features (such as the memory area-related run-time assignment checks) will require compiler optimizations in order to avoid performance degradation. There are a number of "rough edges" – inconsistencies or other minor errors that are almost inevitable when a spec is finalized before it has been implemented. Moreover, parts of the spec are rather complex – most notably the scoped memory area semantics, and also the ATC facility. This will complicate the implementation and present a learning barrier to potential users. One of the biggest advantages to Java is that developers can largely ignore memory management issues, but that does not hold true for the RTSJ. If careful attention is not paid to how memory areas are used, the program could incur a storage leakage or an assignment error.

These issues notwithstanding, the RTSJ is receiving attention from the real-time community, and several projects are either evaluating it or committing to its usage. It will most likely succeed in areas where there is already a commitment to Java for other reasons; for example, an enterprise project that has real-time components / requirements.

The potential for the RTSJ in the traditional hard real-time domain seems more questionable. That community's dominant language (and culture) is currently C, with some Ada and also other languages. They generally remain unconvinced of the purported benefits of OOP, and the dynamic flexibility inherent in Java is treated with suspicion in systems for which static analyzability is important.

Regardless of the extent of its eventual usage, the RTSJ has been a formidable technical accomplishment, illustrating some interesting technology cross fertilization. A number of ideas from Ada influenced the RTSJ (perhaps not too surprising, since the RTSJ design group included several members of the Ada real-time community). And in the other direction, some aspects of the RTSJ may influence future directions for the Ada language. Experience with the RTSJ will likely determine the extent of that effect.

Acronyms Used in This Article

AEH	Asynchronous Event Handler
AI	Asynchronously Interruptible
AIE	AsynchronouslyInterruptedException
ATC	Asynchronous Transfer of Control
GC	Garbage Collector
JCP	Java Community Process
JSR	Java Specification Request
JVM	Java Virtual Machine
PCE	PriorityCeiling Emulation
PI	Priority Inheritance
RI	Reference Implementation
RTOS	Real-Time Operating System
RTSJ	Real-Time Specification for Java

References

[1] www.nist.gov/rt-java

[2] G. Bollella, J.Gosling, B. Brosgol, P. Dibble, S. Furr, D. Hardin, M. Turnbull (2000); *The Real-Time Specification for Java*; Addison-Wesley.

[3] P. Dibble (2002), *Introduction to the Real-Time Java Platform*, Prentice-Hall.

[4] A. Wellings (2004), *Concurrent and Real-Time Programming in Java*, John Wiley & Sons.

[5] K. Nilsen (2000); *Real-Time Core Extensions for the Java Platform*; J-Consortium.

[6] J. Gosling, B. Joy, G. Steele, G. Bracha (2000); *The Java Language Specification (2nd ed.)*; Addison Wesley.

[7] B. Brosgol and A. Wellings (2003); "A Comparison of the Asynchronous Transfer of Control Facilities in Ada and the Real-Time Specification for Java", *Proc. Ada Europe 2003*; Springer.

[8] F. Siebert (2002): *Hard Realtime Garbage Collection in Modern Object Oriented Programming Languages*; aicas Books.

Rationale for Ada 2005: Introduction

John Barnes

John Barnes Informatics, 11 Albert Road, Caversham, Reading RG4 7AN, UK; Tel: +44 118 947 4125; email: jgpb@jbinfo.demon.co.uk

Abstract

This is the first of a number of papers describing the rationale for Ada 2005. In due course it is anticipated that the papers will be combined (after appropriate reformatting and editing) into a single volume for formal publication.

This first paper covers the background to the development of Ada 2005 and gives a brief overview of the main changes from Ada 95. Later papers will then look at the changes in more detail.

Keywords: rationale, Ada 2005.

1 Revision process

Readers will recall that the development of Ada 95 from Ada 83 was an extensive process funded by the USDoD. Formal requirements were established after comprehensive surveys of user needs and competitive proposals were then submitted resulting in the selection of Intermetrics as the developer under the devoted leadership of Tucker Taft. The whole technical development process was then comprehensively monitored by a distinct body of Distinguished Reviewers. Of course, the process was also monitored by the ISO committee concerned and the new language finally became an ISO standard in 1995.

The development of Ada 2005 from Ada 95 has been (and continues to be) on a more modest scale. The work has almost entirely been by voluntary effort with support from within the industry itself through bodies such as the Ada Resource Association and Ada-Europe.

The development is being performed under the guidance of ISO/IEC JTC1/SC22 WG9 (hereinafter just called WG9) chaired adroitly by James Moore whose deep knowledge leads us safely through the minefield of ISO procedures. This committee has included national representatives of many nations including Belgium, Canada, France, Germany, Italy, Japan, Sweden, Switzerland, the UK and the USA. WG9 developed guidelines [1] for a revision to Ada 95 which were then used by the Ada Rapporteur Group (the ARG) in drafting the revised standard.

The ARG is a team of experts nominated by the national bodies represented on WG9 and the two liaison organizations, ACM SIGAda and Ada-Europe. The ARG was originally led with Teutonic precision by Erhard Plödereder and is currently led with Transalpine Gallic flair by Pascal Leroy. The editor, who at the end of the day actually writes the words of the standard, is the indefatigable Randy (fingers) Brukardt.

Suggestions for the revised standard have come from a number of sources such as individuals on the ARG, national bodies on WG9, users via email discussions on Ada-Comment and so on.

At the time of writing (November 2004), the revision process is not quite finished. However, the details of all individual changes are now clear. Nevertheless, integration of the changes needs to be done and maybe a few tweaks will be necessary before the final standard emerges in late 2005 or early 2006.

2 Scope of revision

The changes from Ada 83 to Ada 95 were large. They included several major new items such as

- polymorphism through tagged types, class-wide types and dispatching,
- the hierarchical library system including public and private child packages,
- protected objects for better real-time control,
- more comprehensive predefined library, especially for character and string handling,
- specialized annexes such as those for system programming, real-time, and numerics.

By contrast the changes from Ada 95 to Ada 2005 are relatively modest. Ada 95 was almost a new language which happened to be compatible with Ada 83. However, a new language always brings surprises and despite very careful design things do not always turn out quite as expected when used in earnest.

Indeed, a number of errors in the Ada 95 standard were corrected in the Corrigendum issued in 2001 [2] and then incorporated into the Consolidated Ada Reference Manual [3]. But it was still essentially the same language and further improvement needed to be done.

Technically, Ada 2005 is defined as an Amendment to rather than a Revision of the Ada 95 standard and this captures the flavour of the changes not being very extensive.

In a sense we can think of Ada 2005 as rounding out the rough edges in Ada 95 rather than making major leaps forward. This is perhaps not quite true of the Real-Time Systems annex which includes much new material of an optional nature. Nevertheless I am sure that the changes will bring big benefits to users at hopefully not too much cost to implementors.

The scope of the Amendment was guided by a document issued by WG9 to the ARG in September 2002 [1]. The key paragraph is:

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

Note that by saying "identified problems" it implicitly rejects a major redesign such as occurred with Ada 95. The phrase in parentheses draws attention to the areas where Ada has a major market presence. Ada has carved an important niche in the safety-critical areas which almost inevitably are of a real-time and/or embedded nature. But Ada is also in successful use in very large systems where the inherent reliability and composition features are extremely valuable. So changes should aim to help in those areas. And the final sentence is really an exhortation to steer a middle course between too much change and not enough.

The document then identifies two specific worthwhile changes, namely, inclusion of the Ravenscar profile [4] (for predictable real-time) and a solution to the problem of mutually dependent types across two packages (see Section 3.3 below).

The ARG is then requested to pay particular attention to

A Improvements that will maintain or improve Ada's advantages, especially in those user domains where safety and criticality are prime concerns. Within this area it cites as high priority, improvements in the real-time features and improvements in the high integrity features. Of lesser priority are features that increase static error checking. Improvements in interfacing to other languages are also mentioned.

B Improvements that will remedy shortcomings in Ada. It cites in particular improvements in OO features, specifically, adding a Java-like interface feature and improved interfacing to other OO languages.

So the ARG is asked to improve both OO and real-time with a strong emphasis on real-time and high integrity features. It is interesting that WG9 rejected the thought that "design by contract" features should be added to the above general categories.

The ARG is also asked to consider the following factors in selecting features for inclusion:

- Implementability. Can the feature be implemented at reasonable cost?
- Need. Do users actually need it? [A good one!]
- Language stability. Would it appear disturbing to current users?
- Competition and popularity. Does it help to improve the perception of Ada and make it more competitive?
- Interoperability. Does it ease problems of interfacing with other languages and systems? [That's the third mention of interfacing.]
- Language consistency. Is it syntactically and semantically consistent with the language's current structure and design philosophy?

An important further statement is that "In order to produce a technically superior result, it is permitted to compromise backwards compatibility when the impact on users is judged to be acceptable." In other words don't be paranoid about compatibility.

Finally, there is a warning about secondary standards. Its essence is don't use secondary standards if you can get the material into the RM itself. And please put the stuff on vectors and matrices from ISO/IEC 13813 [5] into the language itself. The reason for this exhortation is that secondary standards have proved themselves to be almost invisible and hence virtually useless.

The guidelines conclude with the target schedule. This included WG9 approval of the scope of the amendment in June 2004 (this was achieved) and final ISO/IEC JCT1 ballot in late 2005 (fingers crossed).

3 Overview of changes

It would be tedious to give a section by section review of the changes as seen by the Reference Manual language lawyer. Instead, the changes will be presented by areas as seen by the user. There can be considered to be six areas:

1 Improvements to the OO model. These include a more traditional notation for invoking an operation of an object without needing to know precisely where the operation is declared (the Obj.Op(...) style), Java-like multiple inheritance using the concept of interfaces, the introduction of null procedures as a category of operation rather like an abstract operation, and the ability to do type extension at a more nested level than that of the parent type. There are also explicit features for overcoming nasty bugs that arise from confusion between overloading and overriding.

2 More flexible access types. Ada 95 access types have a hair-shirt flavour compared with other languages because of the general need for explicit conversions with named access types. This is alleviated by permitting anonymous access types in more contexts. It is also possible to indicate whether an access type is an access to a constant and whether a null value is permitted. Anonymous access-to-subprogram types are also introduced thus permitting so-called downward closures.

3 Enhanced structure and visibility control. The most important change here is the introduction of limited with clauses which allow types in two packages to refer to each other (the mutual dependence problem referred to in the WG9 guidelines). This is done by extending the concept of incomplete types (and introducing tagged incomplete types). There are also private with clauses just providing access from a private part. And there are significant changes to limited types to make them more useful; these include initialization using limited aggregates and composition using a new form of return statement.

4 Tasking and real-time improvements. Almost all of the changes are in the Real-Time Systems annex. They include the introduction of the Ravenscar profile (as explicitly mentioned in the WG9 guidelines) and a number of new scheduling and dispatching policies. There are also new predefined packages for controlling execution time clocks and execution time budgets and for the notification of task termination and similar matters. A change related to the OO model is the introduction of protected and task interfaces thereby drawing the OO and tasking aspects of the language closer together.

5 Improvements to exceptions, generics etc. There are some minor improvements in the exception area, namely, neater ways of testing for null occurrence and raising an exception with a message. Two small but vital numeric changes are a Mod attribute to solve problems of mixing signed and unsigned integers and a fix to the fixed-fixed multiplication problem (which has kept some users locked into Ada 83). There are also a number of new pragmas: Unsuppress to complement the Suppress pragma, Assert which was already offered by most vendors, No_Return which indicates that a procedure never returns normally; and Unchecked_Union to ease interfacing to unchecked unions in C. There is also the ability to have more control of partial parameters of generic formal packages to improve package composition.

6 Extensions to the standard library. New packages include a comprehensive Container library, mechanisms for directory operations and access to environment variables, further operations on times and dates, the vectors and matrices material from ISO/IEC 13813 (as directed in the WG9 guidelines) plus commonly required simple linear algebra algorithms. There are also wide-wide character types and operations for 32-bit characters, the ability to use more characters in identifiers, and improvements and extensions to the existing string packages.

Of course, the areas mentioned above interact greatly and much of 2 and 3 could be classed as improvements to the OO model. There are also a number of changes not mentioned which will mostly be of interest to experts in various areas. These cover topics such as streams, object factory functions, subtle aspects of the overload resolution rules, and the categorization of packages with pragmas Pure and Preelaborate.

The reader might feel that the changes are quite extensive but each has an important role to play in making Ada more useful. Indeed many other changes were rejected as really unnecessary. These include old chestnuts such as **in out** and **out** parameters for functions (ugh), extensible enumeration types (a slippery slope), defaults for all generic parameters (would lead one astray), and user-defined operator symbols (a nightmare).

Before looking at the six areas in a little more detail it is perhaps worth saying a few words about compatibility with Ada 95. The guidelines gave the ARG freedom to be sensible in this area. Of course, the worst incompatibilities are those where a valid program in Ada 95 continues to be valid in Ada 2005 but does something different. It is believed that serious incompatibilities of this nature will never arise. There are however, a very few minor and benign such incompatibilities concerning the raising of exceptions such as that with access parameters discussed in Section 3.2.

However, incompatibilities whereby a valid Ada 95 program fails to compile in Ada 2005 are tolerable provided they are infrequent. A few such incompatibilities are possible. The most obvious cause is the introduction of three more reserved words: **interface**, **overriding**, and **synchronized**. Thus if an existing Ada 95 program uses any of these as an identifier then it will need modification. The introduction of a new category of unreserved keywords was considered for these so that incompatibilities would not arise. However, it was felt that this was ugly, confusing, and prone to introducing nasty errors. In any event the identifiers Overriding and Synchronized are likely to be rare and although Interface is clearly a likely identifier nevertheless to have it both as an identifier and as a keyword in the same program would be nasty. Note also that the pragma Interface which many compilers still support from Ada 83 (although not mentioned by Ada 95 at all) is being put into Annex J for obsolescent features.

3.1 The OO model

The Ada 95 OO model has been criticized as not following the true spirit of the OO paradigm in that the notation for applying subprograms to objects is still dominated by the subprogram and not by the object concerned.

It is claimed that real OO people always give the object first and then the method (subprogram). Thus given

```
package P is
  type T is tagged ... ;
  procedure Op(X: T; ... );
  ...
end P;
```

then assuming that some variable Y is declared of type T, in Ada 95 we have to write

```
P.Op(Y, ... );
```

in order to apply the procedure Op to the object Y whereas a real OO person would expect to write something like

```
Y.Op( ... );
```

where the object Y comes first and only any auxiliary parameters are given in the parentheses.

A real irritation with the Ada 95 style is that the package P containing the declaration of Op has to be mentioned as well. (This assumes that use clauses are not being employed as is usually the case.) However, given an object, from its type we can find its primitive operations and it is illogical to require the mention of the package P. Moreover, in some cases involving a complicated type hierarchy, it is not always obvious to the programmer just which package contains the relevant operation.

The notation giving the object first is now permitted in Ada 2005. The essential rules are that a subprogram call of the form P.Op(Y, ...); can be replaced by Y.Op(...); provided that

- T is a tagged type,
- Op is a primitive (dispatching) or class wide operation of T,
- Y is the first parameter of Op.

The new dotted notation has other advantages in unifying the notation for calling a function and reading a component of a tagged type. Thus consider the following geometrical example which is based on that in a (hopefully familiar) textbook [6]

```
package Geometry is
  type Object is abstract tagged
    record
      X_Coord: Float;
      Y_Coord: Float;
    end record;

  function Area(O: Object) return Float is abstract;
  function MI(O: Object) return Float is abstract;
end;
```

The type Object has two components and two primitive operations Area and MI (Area is the area of an object and MI is its moment of inertia but the fine details of Newtonian mechanics need not concern us). The key point is that with the new notation we can access the coordinates and the area in a unified way. For example, suppose we derive a concrete type Circle thus

```
package Geometry.Circle is
  type Circle is new Object with
    record
      Radius: Float;
    end record;

  function Area(C: Circle) return Float;
  function MI(C: Circle) return Float;
end;
```

where we have provided concrete operations for Area and MI. Then in Ada 2005 we can access both the coordinates and area in the same way

```
X:= A_Circle.X_Coord;
A:= A_Circle.Area;          -- call of function Area
```

Note that since Area just has one parameter (A_Circle) there are no parentheses required in the call. This uniformity is well illustrated by the body of MI which can be written as

```
function MI(C: Circle) is
begin
  return 0.5 * C.Area * C.Radius**2;
end MI;
```

whereas in Ada 95 we had to write

```
    return 0.5 * Area(C) * C.Radius**2;
```

which is perhaps a bit untidy.

A related advantage concerns dereferencing. If we have an access type such as

```
type Pointer is access all Object'Class;
...
This_One: Pointer := A_Circle'Access;
```

and suppose we wish to print out the coordinates and area then in Ada 2005 we can uniformly write

```
Put(This_One.X_Coord); ...
Put(This_One.Y_Coord); ...
Put(This_One.Area); ...                    -- Ada 2005
```

whereas in Ada 95 we have to write

```
Put(This_One.X_Coord); ...
Put(This_One.Y_Coord); ...
Put(Area(This_One.all)); ... -- Ada 95
```

In Ada 2005 the dereferencing is all implicit whereas in Ada 95 some dereferencing has to be explicit which is ugly.

The reader might feel that this is all syntactic sugar for the novice and of no help to real macho programmers. So we shall turn to the topic of multiple inheritance. In Ada 95, multiple inheritance is hard. It can sometimes be done using generics and/or access discriminants (not my favourite topic) but it is hard work and often not possible at all. So it is a great pleasure to be able to say that Ada 2005 introduces real multiple inheritance in the style of Java.

The problem with multiple inheritance in the most general case is clashes between the parents. Assuming just two parents, what happens if both parents have the same component (possibly inherited from a common ancestor)? Do we get two copies? And what happens if both parents have the same operation but with different implementations? These and related problems are overcome by placing firm restrictions on the possible properties of parents. This is done by introducing the notion of an interface.

An interface can be thought of as an abstract type with no components – but it can of course have abstract operations. It has also proved useful to introduce the idea of a null procedure as an operation of a tagged type; we don't have to provide an actual body for such a null procedure (and indeed cannot) but it behaves as if it has a body consisting of just a null statement. So we might have

```
package P1 is
  type Int1 is interface;
  procedure Op1(X: Int1) is abstract;
  procedure N(X: Int1) is null;
end P1;
```

Note carefully that **interface** is a new reserved word. We could now derive a concrete type from the interface Int1 by

```
  type DT is new Int1 with record ... end record;
  procedure Op1(NX: DT);
```

We can provide some components for DT as shown (although this is optional). We must provide a concrete procedure for Op1 (we wouldn't if we had declared DT itself as abstract). But we do not have to provide an overriding of N since it behaves as if it has a concrete null body anyway (but we could override N if we wanted to).

We can in fact derive a type from several interfaces plus possibly one conventional tagged type. In other words we can derive a tagged type from several other types (the ancestor types) but only one of these can be a normal tagged type (it has to be written first). So assuming that Int2 is another interface type and that T1 is a normal tagged type then all of the following are permitted

```
type DT1 is new T1 and Int1 with null record;

type DT2 is new Int1 and Int2 with
  record ... end record;

type DT3 is new T1 and Int1 and Int2 ...
```

It is also possible to compose interfaces to create further interfaces thus

```
type Int3 is interface and Int1;
...
type Int4 is interface and Int1 and Int2 and Int3;
```

Note carefully that **new** is not used in this construction. Such composed interfaces have all the operations of all their parents and further operations can be added in the usual way but of course these must be abstract or null.

There are a number of simple rules to resolve what happens if two parent interfaces have the same operation. Thus a null procedure overrides an abstract one but otherwise repeated operations have to have the same profile.

Some more extensive examples of the use of interfaces will be given in a later paper.

Incidentally, the newly introduced null procedures are not just for interfaces. We can give a null procedure as a specification whatever its profile and no body is then required or allowed. But they are clearly of most value with tagged types and inheritance. Note in particular that the package Ada.Finalization in Ada 2005 is

```
package Ada.Finalization is
  pragma Preelaborate(Finalization);
  pragma Remote_Types(Finalization);

  type Controlled is abstract tagged private;
  procedure Initialize(Object: in out Controlled) is null;
  procedure Adjust(Object: in out Controlled) is null;
  procedure Finalize(Object: in out Controlled) is null;

  -- similarly for Limited_Controlled
  ...
end Ada.Finalization;
```

The procedures Initialize, Adjust, and Finalize are now explicitly given as null procedures. This is only a cosmetic change since the Ada 95 RM states that the default implementations have no effect. However, this neatly clarifies the situation and removes ad hoc semantic rules.

Another important change is the ability to do type extension at a level more nested than that of the parent type. This means that controlled types can now be declared at any level whereas in Ada 95, since the package Ada.Finalization is at the library level, controlled types could only be declared at the library level. There are similar advantages in generics since currently many generics can only be instantiated at the library level.

The final change in the OO area to be described here is the ability to (optionally) state explicitly whether a new operation overrides an existing one or not.

At the moment, in Ada 95, small careless errors in subprogram profiles can result in unfortunate consequences whose cause is often difficult to determine. This is very much against the design goal of Ada to encourage the writing of correct programs and to detect errors at compilation time whenever possible. Consider

```
with Ada.Finalization;  use Ada.Finalization;
package Root is
  type T is new Controlled with ... ;
  procedure Op(Obj: in out T; Data: in Integer);
  procedure Finalise(Obj: in out T);
end Root;
```

Here we have a controlled type plus an operation Op of that type. Moreover, we intended to override the automatically inherited null procedure Finalize of Controlled but, being foolish, we have spelt it Finalise. So our new procedure does not override Finalize at all but merely provides another operation. Assuming that we wrote Finalise to do something useful then we will find that nothing happens when an object of the type T is automatically finalized at the end of a block because the inherited null procedure is called rather than our own code. This sort of error can be very difficult to track down.

In Ada 2005 we can protect against such errors since it is possible to mark overridden operations as such thus

```
  overriding
  procedure Finalize(Obj: in out T);
```

And now if we spell Finalize incorrectly then the compiler will detect the error. Note that **overriding** is another new reserved word. However, partly for reasons of compatibility, the use of overriding indicators is optional; there are also deeper reasons concerning private types and generics which will be discussed in a later paper.

Similar problems can arise if we get the profile wrong. Suppose we derive a new type from T and attempt to override Op thus

```
package Root.Leaf is
  type NT is new T with null record;
  procedure Op(Obj: in out NT; Data: in String);
end Root.Leaf;
```

In this case we have given the identifier Op correctly but the profile is different because the parameter Data has inadvertently been declared as of type String rather than Integer. So this new version of Op will simply be an overloading rather than an overriding. Again we can guard against this sort of error by writing

```
  overriding
  procedure Op(Obj: in out NT; Data: in Integer);
```

On the other hand maybe we truly did want to provide a new operation. In this case we can write **not overriding** and

the compiler will then ensure that the new operation is indeed not an overriding of an existing one thus

```
not overriding
procedure Op(Obj: in out NT; Data: in String);
```

The use of these overriding indicators prevents errors during maintenance. Thus if later we add a further parameter to Op for the root type T then the use of the indicators will ensure that we modify all the derived types appropriately.

3.2 Access types

It has been said that playing with pointers is like playing with fire – properly used all is well but carelessness can lead to disaster. In order to avoid disasters, Ada 95 takes a stern view regarding the naming of access types and their conversion. However, experience has shown that the Ada 95 view is perhaps unnecessarily stern and leads to tedious programming.

We will first consider the question of giving names to access types. In Ada 95 all access types are named except for access parameters and access discriminants. Thus we might have

```
type Animal is tagged
  record Legs: Integer; ... end record;

type Acc_Animal is access Animal;   -- named

procedure P(Beast: access Animal; ... );          -- anonymous
```

Moreover, there is a complete lack of symmetry between named access types and access parameters. In the case of named access types, they all have a null value (and this is the default on declaration if no initial value be given). But in the case of access parameters, a null value is not permitted as an actual parameter. Furthermore, named access types can be restricted to be access to constant types such as

```
type Rigid_Animal is access constant Animal;
```

which means that we cannot change the value of the Animal referred to. But in the case of access parameters, we cannot say

```
procedure P(Beast: access constant Animal);  -- not 95
```

In Ada 2005 almost all these various restrictions are swept away in the interests of flexibility and uniformity.

First of all we can explicitly specify whether an access type (strictly subtype) has a null value. We can write

```
type Acc_Animal is not null access all Animal'Class;
```

This means that we are guaranteed that an object of type Acc_Animal cannot refer to a null animal. And so on declaration such an object should be initialized as in the following sequence

```
type Pig is new Animal with ... ;
Empress_Of_Blandings: aliased Pig := ... ;

My_Animal: Acc_Animal := -- must initialize
        Empress_Of_Blandings'Access;
```

(The Empress of Blandings is a famous pig in the novels concerning Lord Emsworth by the late P G Wodehouse.) If we forget to initialize My_Animal then Constraint_Error is raised; technically the underlying type still has a null value but Acc_Animal does not. We can also write **not null access constant** and **not null access all** of course.

The advantage of using a null exclusion is that when we come to do a dereference

```
Number_of_Legs: Integer := My_Animal.Legs;
```

then no check is required to ensure that we do not dereference a null pointer. This makes the code faster.

Exactly the same freedom also applies to access parameters. Thus we can write all of the following in Ada 2005

```
procedure P(Beast: access Animal);
procedure P(Beast: access constant Animal);
procedure P(Beast: access all Animal);

procedure P(Beast: not null access Animal);
procedure P(Beast: not null access constant Animal);
procedure P(Beast: not null access all Animal);
```

A little quirk is that **all** doesn't do anything in this context since access parameters always were general (that is, they could refer to declared objects as well as to allocated ones).

Note what is in practice a minor incompatibility, the first of the above now permits a null value as actual parameter in Ada 2005 whereas it was forbidden in Ada 95. This is actually a variation at runtime which is normally considered abhorrent. But in this case it just means that any check that will still raise Constraint_Error will be in a different place – and in any event the program was presumably incorrect.

Another change in Ada 2005 is that we can use anonymous access types other than just as parameters (and discriminants). We can in fact also use anonymous access types in

- the declaration of stand-alone objects and components of arrays and records,
- a renaming declaration,
- a function return type.

Thus we can extend our farmyard example

```
type Horse is new Animal with ... ;

type Acc_Horse is access all Horse;
type Acc_Pig is access all Pig;

Napoleon, Snowball: Acc_Pig := ... ;

Boxer, Clover: Acc_Horse := ... ;
```

and now we can declare an array of animals

```
Animal_Farm: constant array (Positive range <>) of
        access Animal'Class :=
          (Napoleon, Snowball, Boxer, Clover);
```

(With acknowledgments to George Orwell.) Note that the components of the array are of an anonymous access type. We can also have record components of an anonymous type

```
type Ark is
  record
    Stallion, Mare: access Horse;
    Boar, Sow: access Pig;
```

```
    Cockerel, Hen: access Chicken;
    Ram, Ewe: access Sheep;
    ...
  end record;

Noahs_Ark: Ark := (Boxer, Clover, ... );
```

This is not a very good example since I am sure that Noah took care to take actual animals into the Ark and not merely their addresses.

A more useful example is given by the classic linked list. In Ada 95 (and Ada 83) we have

```
type Cell;
type Cell_Ptr is access Cell;

type Cell is
  record
    Next: Cell_Ptr;
    Value: Integer;
  end record;
```

In Ada 2005, we do not have to declare the type Cell_Ptr in order to declare the type Cell and so we do not need to use the incomplete declaration to break the circularity. We can simply write

```
type Cell is
  record
    Next: access Cell;
    Value: Integer;
  end record;
```

Here we have an example of the use of the type name Cell within its own declaration. In some cases this is interpreted as referring to the current instance of the type (for example, in a task body) but the rule has been changed to permit its usage as here.

We can also use an anonymous access type for a single variable such as

```
List: access Cell := ... ;
```

An example of the use of an anonymous access type for a function result might be in another animal function such as

```
function Mate_Of(A: access Animal'Class)
            return access Animal'Class;
```

We could then perhaps write

```
if Mate_Of(Noahs_Ark.Ram) /= Noahs_Ark.Ewe then
  ... -- better get Noah to sort things out
end if;
```

Anonymous access types can also be used in a renaming declaration. This and other detailed points on matters such as accessibility will be discussed in a later paper.

The final important change in access types concerns access to subprogram types. Access to subprogram types were introduced into Ada 95 largely for the implementation of callback. But important applications of such types in other languages (going back to Pascal and even Algol 60) are for mathematical applications such as integration where a function to be manipulated is passed as a parameter. The Ada 83 and Ada 95 approach has always been to say "use generics". But this can be clumsy and so a direct alternative is now provided.

Recall that in Ada 95 we can write

```
type Integrand is access function(X: Float) return Float;

function Integrate(Fn: Integrand; Lo, Hi: Float)
            return Float;
```

The idea is that the function Integrate finds the value of the integral of the function passed as parameter Fn between the limits Lo and Hi. This works fine in Ada 95 for simple cases such as where the function is declared at library level. Thus to evaluate

$$\int_0^1 \sqrt{x}\, dx$$

we can write

```
Result := Integrate(Sqrt'Access, 0.0, 1.0);
```

where the function Sqrt is from the library package Ada.Numerics.Elementary_Functions.

However, if the function to be integrated is more elaborate then we run into difficulty in Ada 95 if we attempt to use access to subprogram types. Consider the following example which aims to compute the integral of the expression xy over the square region $0 \le x, y \le 1$.

```
with Integrate;
procedure Main is
  function G(X: Float) return Float is
    function F(Y: Float) return Float is
    begin
      return X*Y;
    end F;
  begin
    return Integrate(F'Access, 0.0, 1.0);          -- illegal in 95
  end G;

  Result: Float;
begin
  Result:= Integrate(G'Access, 0.0, 1.0);          -- illegal in 95
  ...
end Main;
```

But this is illegal in Ada 95 because of the accessibility rules necessary with named access types in order to prevent dangling references. Thus we need to prevent the possibility of storing a pointer to a local subprogram in a global structure. This means that both F'Access and G'Access are illegal in the above.

Note that although we could make the outer function G global so that G'Access would be allowed nevertheless the function F has to be nested inside G in order to gain access to the parameter X of G. It is typical of functions being integrated that they have to have information passed globally – the number of parameters of course is fixed by the profile used by the function Integrate.

The solution in Ada 2005 is to introduce anonymous access to subprogram types by analogy with anonymous access to object types. Thus the function Integrate becomes

```
function Integrate(Fn: access function(X: Float) return Float;
      Lo, Hi: Float) return Float;
```

Note that the parameter Fn has an anonymous type defined by the profile so that we get a nesting of profiles. This may seem a bit convoluted but is much the same as in Pascal.

The nested example above is now valid and no accessibility problems arise. (The reader will recall that accessibility

problems with anonymous access to object types are prevented by a runtime check; in the case of anonymous access to subprogram types the corresponding problems are prevented by decreeing that the accessibility level is infinite.)

Anonymous access to subprogram types are also useful in many other applications such as iterators as will be illustrated later.

Note that we can also prefix all access to subprogram types both named and anonymous by **not null** in the same way as for access to object types.

3.3 Structure, visibility and limited types

Structure is vital for controlling visibility and thus abstraction. There were huge changes in Ada 95. The whole of the hierarchical child unit mechanism was introduced with both public and private children. It was hoped that this would provide sufficient flexibility for the future.

But one problem has remained. Suppose we have two types where each wishes to refer to the other. Both need to come first! Basically we solve the difficulty by using incomplete types. We might have a drawing package concerning points and lines in a symmetric way. Each line contains a list or array of the points on it and similarly each point contains a list or array of the lines through it. We can imagine that they are both derived from some root type containing printing information such as colour. In Ada 95 we might write

```
type Object is abstract tagged
  record
    Its_Colour: Colour;
    ...
  end record;

type Point;
type Line;
type Acc_Point is access all Point;
type Acc_Line is access all Line;

subtype Index is Integer range 0 .. Max;
type Acc_Line_Array is array (1 .. Max) of Acc_Line;
type Acc_Point_Array is array (1 .. Max) of Acc_Point;

type Point is new Object with
  record
    No_Of_Lines: Index;
    LL: Acc_Line_Array;
    ...
  end record;

type Line is new Object with
  record
    No_Of_Points: Index;
    PP: Acc_Point_Array;
    ...
  end record;
```

This is very crude since it assumes a maximum number Max of points on a line and vice versa and declares the arrays accordingly. The reader can flesh it out more flexibly. Well this is all very well but if the individual types get elaborate and each has a series of operations, we might want to declare them in distinct packages (perhaps child packages of that containing the root type). In Ada 95 we cannot do this because both the incomplete declaration and its completion have to be in the same package.

The net outcome is that we end up with giant cumbersome packages.

What we need therefore is some way of logically enabling the incomplete view and the completion to be in different packages. The elderly might remember that in the 1980 version of Ada the situation was even worse – the completion had to be in the same list of declarations as the incomplete declaration. Ada 83 relaxed this (the so-called Taft Amendment) and permits the private part and body to be treated as one list – the same rule applies in Ada 95. We now go one step further.

Ada 2005 solves the problem by introducing a variation on the with clause – the limited with clause. The idea is that a library package (and subprogram) can refer to another library package that has not yet been declared and can refer to the types in that package but only as if they were incomplete types. Thus we might have a root package Geometry containing the declarations of Object, Max, Index and so on and then

```
limited with Geometry.Lines;
package Geometry.Points is

  type Acc_Line_Array is array (1 .. Max) of
                access Lines.Line;

  type Point is new Object with
    record
      No_Of_Lines: Index;
      LL: Acc_Line_Array;
      ...
    end record;

  ...
end Geometry.Points;
```

The package Geometry.Lines is declared in a similar way. Note especially that we are using the anonymous access type facility discussed in Section 3.2 and so we do not even have to declare named access types such as Acc_Line in order to declare Acc_Line_Array.

By writing **limited with** Geometry.Lines; we get access to all the types visible in the specification of Geometry.Lines but as if they were declared as incomplete. In other words we get an incomplete view of the types. We can then do all the things we can normally do with incomplete types such as use them to declare access types. (Of course the implementation checks later that Geometry.Lines does actually have a type Line.)

Not only is the absence of the need for a named type Acc_Line a handy shorthand, it also prevents the proliferation of named access types. If we did want to use a named type Acc_Line in both packages then we would have to declare a distinct type in each package. This is because from the point of view of the package Points, the Acc_Line in Lines would only be an incomplete type (remember each package only has a limited view of the other) and thus would be essentially unusable. The net result would be many named access types and wretched type conversions all over the place.

There are also some related changes to the notation for incomplete types. We can now write

```
type T is tagged;
```

and we are then guaranteed that the full declaration will reveal T to be a tagged type. The advantage is that we also know that, being tagged, objects of the type T will be passed by reference. Consequently we can use the type T for parameters before seeing its full declaration. In the example of points and lines above, since Line is visibly tagged in the package Geometry.Lines we will thus get an incomplete tagged view of Lines.

The introduction of tagged incomplete types clarifies the ability to write

```
type T_Ptr is access all T'Class;
```

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

Further examples of the use of limited with clauses will be given in a later paper.

Another enhancement in this area is the introduction of private with clauses which overcome a problem with private child packages.

Private child packages were introduced to enable the details of the implementation of part of a system to be decomposed and yet not be visible to the external world. However, it is often convenient to have public packages that use these details but do not make them visible to the user. In Ada 95 a parent or sibling body can have a with clause for a private child. But the specifications cannot. These rules are designed to ensure that information does not leak out via the visible part of a specification. But there is no logical reason why the private part of a package should not have access to a private child. Ada 2005 overcomes this by introducing private with clauses. We can write

```
private package App.Secret_Details is
   type Inner is ...
   ... -- various operations on Inner etc
end App.Secret_Details;

private with App.Secret_Details;
package App.User_View is

   type Outer is private;
   ... -- various operations on Outer visible to the user

   -- type Inner is not visible here
private
   -- type Inner is visible here

   type Outer is
      record
         X: Secret_Details.Inner;
         ...
      end record;
   ...
end App.User_View;
```

thus the private part of the public child has access to the type Inner but it is still hidden from the external user.

Note that the public child and private child might have mutually declared types as well in which case they might also wish to use the limited with facility. In this case the public child would have a limited private with clause for the private child written thus

```
limited private with App.Secret_Details;
package App.User_View is ...
```

In the case of a parent package, its specification cannot have a with clause for a child (logically the specification cannot know about the child because the parent must be declared, that is put into the program library, first). Similarly a parent cannot have a private with clause for a private child. But it can have a limited with clause for any child (thereby breaking the circularity) and in particular it can have a limited private with clause for a private child. So we might also have

```
limited private with App.Secret_Details;
package App is ...
```

The final topic in this section is limited types. The reader will recall that the general idea of a limited type is to restrict the operations that the user can perform on a type to just those provided by the developer of the type and in particular to prevent the user from doing assignment and thus making copies of an object of the type.

However, limited types have never quite come up to expectation both in Ada 83 and Ada 95. Ada 95 brought significant improvements by disentangling the concept of a limited type from a private type but problems have remained.

The key problem is that Ada 95 does not allow the initialization of limited types because of the view that initialization requires assignment and thus copying. A consequence is that we cannot declare constants of a limited type either. Ada 2005 overcomes this problem by allowing initialization by aggregates.

As a simple example, consider

```
type T is limited
   record
      A: Integer;
      B: Boolean;
      C: Float;
   end record;
```

in which the type as a whole is limited but the components are not. If we declare an object of type T in Ada 95 then we have to initialize the components (by assigning to them) individually thus

```
   X: T;
begin
   X.A := 10;  X.B := True;  X.C := 45.7;
```

Not only is this annoying but it is prone to errors as well. If we add a further component D to the record type T then we might forget to initialize it. One of the advantages of aggregates is that we have to supply all the components (allowing automatic so-called full coverage analysis, a key benefit of Ada).

Ada 2005 allows the initialization with aggregates thus

```
X: T := (A => 10,  B => True,  C => 45.7);
```

Technically, Ada 2005 just recognizes properly that initialization is not assignment. Thus we should think of the individual components as being initialized individually *in situ* – an actual aggregated value is not created and then assigned. (Much the same happens when initializing controlled types with an aggregate.)

Sometimes a limited type has components where an initial value cannot be given. This happens with task and protected types. For example

```
protected type Semaphore is ... ;

type PT is
  record
    Guard: Semaphore;
    Count: Integer;
    Finished: Boolean := False;
  end record;
```

Remember that a protected type is inherently limited. This means that the type PT is limited because a type with a limited component is itself limited. It is good practice to explicitly put **limited** on the type PT in such cases but it has been omitted here for illustration. Now we cannot give an explicit initial value for a Semaphore but we would still like to use an aggregate to get the coverage check. In such cases we can use the box symbol <> to mean use the default value for the type (if any). So we can write

```
X: PT := (Guard => <>, Count => 0, Finished => <>);
```

Note that the ability to use <> in an aggregate for a default value is not restricted to the initialization of limited types. It is a new feature applicable to aggregates in general. But, in order to avoid confusion, it is only permitted with named notation.

Limited aggregates are also allowed in other similar contexts where copying is not involved including as actual parameters of mode **in**.

There are also problems with returning results of a limited type from a function. This is overcome in Ada 2005 by the introduction of an extended form of return statement. This will be described in detail in a later paper.

3.4 Tasking and real-time facilities

Unless mentioned otherwise all the changes in this section concern the Real-Time Systems annex.

First, the well-established Ravenscar profile is included in Ada 2005 as directed by WG9. A profile is a mode of operation and is specified by the pragma Profile which defines the particular profile to be used. Thus to ensure that a program conforms to the Ravenscar profile we write

```
pragma Profile(Ravenscar);
```

The purpose of Ravenscar is to restrict the use of many of the tasking facilities so that the effect of the program is predictable. This is very important for real-time safety-critical systems. In the case of Ravenscar the pragma is equivalent to the joint effect of the following pragmas

```
pragma Task_Dispatching_Policy(FIFO_Within_Priorities);
pragma Locking_Policy(Ceiling_Locking);
pragma Detect_Blocking;
```

plus a **pragma** Restrictions with a host of arguments such as No_Abort_Statements, No_Asynchronous_Control, and No_Dynamic_Priorities.

The pragma Detect_Blocking plus many of the Restrictions identifiers are new to Ada 2005. Further details will be given in a later paper.

Ada 95 allows the priority of a task to be changed but does not permit the ceiling priority of a protected object to be changed. This is rectified in Ada 2005 by the introduction of an attribute Priority for protected objects and the ability to change it by a simple assignment such as

```
My_PO'Priority := P;
```

inside a protected operation of the object My_PO. The change takes effect at the end of the protected operation.

The monitoring and control of execution time naturally are important for real-time programs. Ada 2005 includes packages for three different aspects of this

Ada.Execution_Time – this is the root package and enables the monitoring of execution time of individual tasks.

Ada.Execution_Time.Timers – this provides facilities for defining and enabling timers and for establishing a handler which is called by the run time system when the execution time of the task reaches a given value.

Ada.Execution_Time.Group_Budgets – this enables several tasks to share a budget and provides means whereby action can be taken when the budget expires.

The execution time of a task or CPU time as it is commonly called is the time spent by the system executing the task and services on its behalf. CPU times are represented by the private type CPU_Time. The CPU time of a particular task is obtained by calling the following function Clock in the package Ada.Execution_Time

```
function Clock(T: Task_ID := Current_Task)
                  return CPU_Time;
```

A value of type CPU_Time can be converted to a Seconds_Count plus residual Time_Span by a procedure Split similar to that in the package Ada.Real_Time. Incidentally we are guaranteed that the granularity of CPU times is no greater than one millisecond and that the range is at least 50 years.

In order to find out when a task reaches a particular CPU time we use the facilities of the child package Ada.Execution_Time.Timers. This includes a discriminated type Timer and a type Handler thus

```
type Timer(T: access Task_ID) is limited private;
type Handler is access
            protected procedure (TM: in out Timer);
```

We can then set the timer to expire at some absolute time by

```
Set_Handler(My_Timer, Time_Limit, My_Handler'Access);
```

and then when the CPU time of the task reaches Time_Limit (of type CPU_Time), the protected procedure My_Handler is executed. Note how the timer object incorporates the information regarding the task concerned using an access

discriminant and that this is passed to the handler via its parameter. Another version of Set_Handler enables the timer to be triggered after a given interval (of type Time_Span).

In order to program various aperiodic servers it is necessary for tasks to share a CPU budget. This can be done using the child package Ada.Execution_Time.Group_Budgets. In this case we have

```
type Group Budget is limited private;
type Handler is access
   protected procedure (GB: in out Group_Budget);
```

The type Group_Budget both identifies the group of tasks it belongs to and the size of the budget. Various subprograms enable tasks to be added to and removed from a group budget. Other procedures enable the budget to be set and replenished.

A procedure Set_Handler associates a particular handler with a budget.

```
Set_Handler(GB => My_Group_Budget,
            Handler => My_Handler'Access);
```

When the group budget expires the associated protected procedure is executed.

A somewhat related topic is that of low level timing events. The facilities are provided by the package Ada.Real_Time.Timing_Events. In this case we have

```
type Timing_Event is limited private;
type Timing_Event_Handler is access
     protected procedure (Event: in out Timing_Event);
```

The idea here is that a protected procedure can be nominated to be executed at some time in the future. Thus to ring a pinger when our egg is boiled after four minutes we might have a protected procedure

```
protected body Egg is
  procedure Is_Done (Event: in out Timing_Event) is
  begin
    Ring_The_Pinger;
  end Is_Done;
end Egg;
```

and then

```
Egg_Done: Timing_Event;
Four_Min: Time_Span := Minutes(4);
...
Put_Egg_In_Water;
Set_Handler(Event => Egg_Done, In_Time => Four_Min,
            Handler => Egg.Is_Done'Access);
-- now read newspaper whilst waiting for egg
```

This facility is of course very low level and does not involve Ada tasks at all. Note that we can set the event to occur at some absolute time as well as at a relative time as above. Incidentally, the function Minutes is a new function added to the parent package Ada.Real_Time. Otherwise we would have had to write something revolting such as 4*60*Milliseconds(1000). A similar function Seconds has also been added.

There is a minor flaw in the above example. If we are interrupted by the telephone between putting the egg in the water and setting the handler then our egg will be overdone. We will see how to cure this in a later paper.

Readers will recall the old problem of how tasks can have a silent death. If something in a task goes wrong in Ada 95 and an exception is raised which is not handled by the task, then it is propagated into thin air and just vanishes. It was always deemed impossible for the exception to be handled by the enclosing unit because of the inherent asynchronous nature of the event.

This is overcome in Ada 2005 by the package Ada.Task_Termination which provides facilities for associating a protected procedure with a task. The protected procedure is invoked when the task terminates with an indication of the reason. Thus we might declare a protected object Grim_Reaper

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

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

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

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

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

There are three possible reasons for termination, it could be normal, abnormal, or caused by an unhandled exception. In the last case the parameter X gives details of the exception occurrence.

Another area of increased flexibility in Ada 2005 is that of task dispatching policies. In Ada 95, the only predefined policy is FIFO_Within_Priorities although other policies are permitted. Ada 2005 provides further pragmas, policies and packages which facilitate many different mechanisms such as non-preemption within priorities, the familiar Round Robin using timeslicing, and the more recently acclaimed Earliest Deadline First (EDF) policy. Moreover it is possible to mix different policies according to priority level within a partition.

Various facilities are provided by the package Ada.Dispatching plus two child packages

Ada.Dispatching – this is the root package and simply declares an exception Dispatching_Policy_Error.

Ada.Dispatching.Round_Robin – this enables the setting of the time quanta for time slicing within one or more priority levels.

Ada.Dispatching.EDF – this enables the setting of the deadlines for various tasks.

A policy can be selected for a whole partition by one of

```
pragma Task_Dispatching_Policy
      (Non_Preemptive_FIFO_Within_Priorities);

pragma Task_Dispatching_Policy
      (Round_Robin_Within_Priorities);

pragma Task_Dispatching_Policy
      (EDF_Across_Priorities);
```

In order to mix different policies across different priority levels we use the pragma Priority_Specific_Dispatching with various policy identifiers thus

```
pragma Priority_Specific_Dispatching
          (Round_Robin_Within_Priority, 1, 1);
pragma Priority_Specific_Dispatching
          (EDF_Across_Priorities, 2, 10);
pragma Priority_Specific_Dispatching
          (FIFO_Within_Priority, 11, 24);
```

This sets Round Robin at priority level 1, EDF at levels 2 to 10, and FIFO at levels 11 to 24.

The final topic in this section concerns the core language and not the Real-Time Systems annex. Ada 2005 introduces a means whereby object oriented and real-time features can be closely linked together through inheritance.

Recall from Section 3.1 that we can declare an interface thus

```
type Int is interface;
```

We can also declare an interface to be limited, synchronized, task, or protected thus

```
type Intlim is limited interface;
type Intsync is synchronized interface;
type Inttask is task interface;
type Intprot is protected interface;
```

A task interface or protected interface has to be implemented by a task type or protected type respectively. However, a synchronized interface can be implemented by either a task type or a protected type. These interfaces can also be composed with certain restrictions. Detailed examples will be given in a later paper.

3.5 Exceptions, numerics, generics etc

As well as the major features discussed above there are also a number of improvements in various other areas.

There are two small changes concerning exceptions. One is that we can give a message with a raise statement, thus

```
raise Some_Error with "A message";
```

This is a lot neater than having to write (as in Ada 95)

```
Ada.Exceptions.Raise_Exception(Some_Error'Identity,
                    "A message");
```

The other change concerns the detection of a null exception occurrence which might be useful in a package analysing a log of exceptions. The problem is that occurrences are of a limited private type and so we cannot compare an occurrence with Null_Occurrence to see if they are equal. In Ada 95 applying the function Exception_Identity to a null occurrence unhelpfully raises Constraint_Error. This has been changed in Ada 2005 to return Null_Id so that we can now write

```
procedure Process_Ex(X: Exception_Occurrence) is
begin
  if Exception_Identity(X) = Null_Id then
    -- process the case of a Null_Occurrence
  ...
end Process_Ex;
```

Ada 95 introduced modular types which are of course unsigned integers. However it has in certain cases proved very difficult to get unsigned integers and signed integers to work together. This is a trivial matter in fragile languages such as C but in Ada the type model has proved obstructive. The basic problem is converting a value of a signed type which happens to be negative to an unsigned type. Thus suppose we want to add a signed offset to an unsigned address value, we might have

```
type Offset_Type is range -(2**31) .. 2**31-1;
type Address_Type is mod 2**32;

Offset: Offset_Type;
Address: Address_Type;
```

We cannot just add Offset to Address because they are of different types. If we convert the Offset to the address type then we might get Constraint_Error and so on. The solution in Ada 2005 is to use a new functional attribute S'Mod which applies to any modular subtype S and converts a universal integer value to the modular type using the corresponding mathematical mod operation. So we can now write

```
Address := Address + Address_Type'Mod(Offset);
```

Another new attribute is Machine_Rounding. This enables high-performance conversions from floating point types to integer types when the exact rounding does not matter.

The third numeric change concerns fixed point types. It was common practice in some Ada 83 programs to define their own multiply and divide operations, perhaps to obtain saturation arithmetic. These programs ran afoul of the Ada 95 rules that introduced universal fixed operations resulting in ambiguities. Without going into details, this problem has been fixed in Ada 2005 so that user-defined operations can now be used.

Ada 2005 has several new pragmas. The first is

```
pragma Unsuppress(Identifier);
```

where the identifier is that of a check such as Range_Check. The general idea is to ensure that checks are performed in a declarative region irrespective of the use of a corresponding pragma Suppress. Thus we might have a type My_Int that behaves as a saturated type. Writing

```
function "*" (Left, Right: My_Int) return My_Int is
  pragma Unsuppress(Overflow_Check);
begin
  return Integer(Left) * Integer(Right);
exception
  when Constraint_Error =>
    if (Left>0 and Right>0) or (Left<0 and Right<0) then
      return My_Int'Last;
    else
      return My_Int'First;
    end if;
end "*";
```

ensures that the code always works as intended even if checks are suppressed in the program as a whole.

Incidentally the On parameter of pragma Suppress which never worked well has been banished to Annex J.

Many implementations of Ada 95 support a pragma Assert and this is now consolidated into Ada 2005. The general idea is that we can write pragmas such as

```
pragma Assert(X >50);
pragma Assert(not Buffer_Full, "buffer is full");
```

The first parameter is a Boolean expression and the second optional parameter is a string. If at the point of the pragma at execution time, the expression is False then action can be taken. The action is controlled by another pragma Assertion_Policy which can switch the assertion mechanism on and off by one of

```
pragma Assertion_Policy(Check);
pragma Assertion_Policy(Ignore);
```

If the policy is to check then the exception Assertion_Error is raised with the message if any. This exception is declared in the predefined package Ada.Assertions. There are some other facilities as well.

The pragma No_Return is also related to exceptions. It can be applied to a procedure (not to a function) and indicates that the procedure never returns normally but only by propagating an exception. Thus we might have

```
procedure Fatal_Error(Message: String);
pragma No_Return(Fatal_Error);
```

And now whenever we call Fatal_Error the compiler is assured that control is not returned and this might enable some optimization or better diagnostic messages.

Note that this pragma applies to the predefined procedure Ada.Exceptions.Raise_Exception.

Finally there is the pragma Unchecked_Union. This is useful for interfacing to programs written in C that use the concept of unions. Unions in C correspond to variant types in Ada but do not store any discriminant which is entirely in the mind of the C programmer. The pragma enables a C union to be mapped to an Ada variant record type by omitting the storage for the discriminant.

If the C program has

```
union {
   spvalue double;
   struct {
      length int;
      first *double;
      } mpvalue;
} number;
```

then this can be mapped in the Ada program by

```
type Number(Kind: Precision) is
   record
      case Kind is
         when Single_Precision =>
            SP_Value: Long_Float;
         when Multiple_Precision =>
            MP_Value_Length: Integer;
            MP_Value_First: access Long_Float;
      end case;
   end record;
pragma Unchecked_Union(Number);
```

One problem with pragmas (and attributes) is that many implementations have added implementation defined ones (as they are indeed permitted to do). However, this can impede portability from one implementation to another. To overcome this there are further Restrictions identifiers so we can write

```
pragma Restrictions(No_Implementation_Pragmas,
         No_Implementation_Attributes);
```

Observe that one of the goals of Ada 2005 has been to standardize as many of the implementation defined attributes and pragmas as possible.

Readers might care to consider the paradox that GNAT has an (implementation-defined) restrictions identifier No_Implementation_Restrictions.

Another new restrictions identifier prevents us from inadvertently using features in Annex J thus

```
pragma Restrictions(No_Obsolescent_Features);
```

Similarly we can use the restrictions identifier No_Dependence to state that a program does not depend on a given language defined package. Thus we might write

```
pragma Restrictions(No_Dependence =>
            Ada.Command_Line);
```

The final new general feature concerns formal generic package parameters. Ada 95 introduced the ability to have formal packages as parameters of generic units. This greatly reduced the need for long generic parameter lists since the formal package encapsulated them.

Sometimes it is necessary for a generic unit to have two (or more) formal packages. When this happens it is often the case that some of the actual parameters of one formal package must be identical to those of the other. In order to permit this there are two forms of generic parameters. One possibility is

```
generic
   with package P is new Q(<>);
package Gen is ...
```

and then the package Gen can be instantiated with any package that is an instantiation of Q. On the other hand we can have

```
generic
   with package R is new S(P1, P2, ... );
package Gen is ...
```

and then the package Gen can only be instantiated with a package that is an instantiation of S with the given actual parameters P1, P2 etc.

These mechanisms are often used together as in

```
generic
   with package P is new Q(<>);
   with package R is new S(P.F1);
package Gen is ...
```

This ensures that the instantiation of S has the same actual parameter (assumed only one in this example) as the parameter F1 of Q used in the instantiation of Q to create the actual package corresponding to P.

There is an example of this in one of the packages for vectors and matrices in ISO/IEC 13813 which is now incorporated into Ada 2005 (see Section 3.6). The generic package for complex arrays has two package parameters. One is the corresponding package for real arrays and the other is the package Generic_Complex_Types from the existing Numerics annex. Both of these packages have a floating type as their single formal parameter and it is important that both instantiations use the same floating type (eg both Float and not one Float and one Long_Float) otherwise a terrible mess will occur. This is assured by writing (using some abbreviations)

```
with ... ;
generic
  with package Real_Arrays is
          new Generic_Real_Arrays(<>);
  with package Complex_Types is
   new Generic_Complex_Types(Real_Arrays.Real);
package Generic_Complex_Arrays is ...
```

Well this works fine in simple cases (the reader may wonder whether this example is simple anyway) but in more elaborate situations it is a pain. The trouble is that we have to give all the parameters for the formal package or none at all in Ada 95.

Ada 2005 permits only some of the parameters to be specified, and any not specified can be indicated using the box. So we can write any of

```
with package Q is new R(P1, P2, F3 => <>);
with package Q is new R(P1, others => <>);
with package Q is new R(F1 => <>, F2 => P2, F3 => P3);
```

Note that the existing form (<>) is now deemed to be a shorthand for (**others => <>**). As with aggregates the form <> is only permitted with named notation.

Examples using this new facility will be given in a later paper.

3.6 Standard library

There are significant improvements to the standard library in Ada 2005. One of the strengths of Java is the huge library that comes with it. Ada has tended to take the esoteric view that it is a language for constructing programs from components and has in the past rather assumed that the components would spring up by magic from the user community. There has also perhaps been a reluctance to specify standard components in case that preempted the development of better ones. However, it is now recognized that standardizing useful stuff is a good thing. And moreover secondary ISO standards are not very helpful because they are almost invisible. Ada 95 added quite a lot to the predefined library and Ada 2005 adds more.

First, there are packages for manipulating arrays and vectors already mentioned in Section 3.5 when discussing formal package parameters. There are two packages, Generic_Real_Arrays and Generic_Complex_Arrays. They can be instantiated according to the underlying floating point type used. There are also nongeneric versions as usual.

These packages export types for declaring vectors and matrices and many operations for manipulating them. Thus if we have an expression in mathematical notation such as

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{z}$$

where $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are vectors and $\boldsymbol{A}$ is a square matrix, then this calculation can be simply programmed as

```
X, Y, Z: Real_Vector(1 .. N);
A: Real_Matrix(1 .. N, 1 .. N);
...
Y := A * X + Z;
```

and the appropriate operations will be invoked. The packages also include subprograms for the most useful linear algebra computations, namely, the solution of linear equations, matrix inversion and determinant evaluation, plus the determination of eigenvalues and eigenvectors for symmetric matrices (Hermitian in the complex case). Thus to determine X given Y, Z and A in the above example we can write

```
X := Solve(A, Y – Z);
```

It should not be thought that these Ada packages in any way compete with the very comprehensive BLAS (Basic Linear Algebra Subprograms). The purpose of the Ada packages is to provide simple implementations of very commonly used algorithms (perhaps for small embedded systems or for prototyping) and to provide a solid framework for developing bindings to the BLAS for more demanding situations. Incidentally, they are in the Numerics annex.

Another (but very trivial) change to the Numerics annex is that nongeneric versions of Ada.Text_IO.Complex_IO have been added in line with the standard principle of providing nongeneric versions of generic predefined packages for convenience. Their omission from Ada 95 was an oversight.

There is a new predefined package in Annex A for accessing tree-structured file systems. The scope is perhaps indicated by this fragment of its specification

```
with ...
package Ada.Directories is
   -- Directory and file operations
   function Current_Directory return String;
   procedure Set_Directory(Directory: in String);
   ...
   -- File and directory name operations
   function Full_Name(Name: in String) return String;
   function Simple_Name(Name: in String) return String;
   ...
   -- File and directory queries
   type File_Kind is
         (Directory, Ordinary_File, Special_File);
   type File_Size is range 0 .. implementation-defined;
   function Exists(Name: in String) return Boolean;
   ...
   -- Directory searching
   type  Directory_Entry_Type is limited private;
   type Filter_Type is array (File_Kind) of Boolean;
   ...
   -- Operations on directory entries
   ...
end Ada.Directories;
```

The package contains facilities which will be useful on any Unix or Windows system. However, it has to be recognized that like Ada.Command_Line it might not be supportable on every environment.

There is also a package Ada.Environment_Variables for accessing the environment variables that occur in most operating systems.

A number of additional subprograms have been added to the existing string handling packages. There are several problems with the Ada 95 packages. One is that conversion between bounded and unbounded strings and the raw type String is required rather a lot and is both ugly and inefficient. For example searching only part of a bounded or unbounded string can only be done by converting it to a String and then searching the appropriate slice.

In brief the additional subprograms are as follows

- three further versions of function Index with an additional parameter From indicating the start of the search are added to each of Strings.Fixed, Strings.Bounded and Strings.Unbounded.
- a further version of function Index_Non_Blank is similarly added to all three packages.
- a procedure Set_Bounded_String with similar behaviour to the function To_Bounded_String is added to Strings.Bounded. This avoids the overhead of using a function. A similar procedure Set_Unbounded_String is added to Strings.Unbounded.
- a function and procedure Bounded_Slice are added to Strings.Bounded. These avoid conversions from type String. A similar function and procedure Unbounded_Slice are added to Strings.Unbounded.

As well as these additions there is a new package Ada.Text_IO.Unbounded_IO for the input-output of unbounded strings. This again avoids unnecessary conversion to the type String. However, it has not been felt necessary to add a similar package for bounded strings partly because of the complexity involved since it would need to be generic.

Finally, two functions Get_Line are added to Ada.Text_IO itself. These avoid difficulties with the length of the string which occurs with the existing procedures Get_Line.

In Ada 83, program identifiers used the 7-bit ASCII set. In Ada 95 this was extended to the 8-bit Latin-1 set. In Ada 2005 this is extended yet again to the entire ISO/IEC 10646:2003 character repertoire. This means that identifiers can now use Cyrillic and Greek characters. Thus we could extend the animal example by

```
Сталин: access Pig renames Napoleon;
Πεγασυς: Horse;
```

In order to encourage us to write our mathematical programs nicely the additional constant

```
π: constant := Pi;
```

has been added to the package Ada.Numerics in Ada 2005.

In a similar way types Wide_String and Wide_Character were added to Ada 95. In Ada 2005 this process is also extended and a set of wide-wide types and packages for 32-bit characters are added. Thus we have types Wide_Wide_String and so on.

A major addition to the predefined library is the package Ada.Containers and its children plus two auxiliary child functions of Ada.Strings. These are a very important and considerable addition to the predefined capability of Ada and brings the best in standard data structure manipulation to the fingers of every Ada programmer. The scope is perhaps best illustrated by listing the units involved.

Ada.Containers – this is the root package and just declares types Hash_Type and Size_Type which are an implementation-defined modular and integer type respectively.

Ada.Strings.Hash and Ada.Strings.Unbounded.Hash – these are functions that hash a string or unbounded string into the type Hash_Type. There is no bounded version.

Ada.Containers.Vectors – this is a generic package with parameters giving the index type and element type of a vector plus "=" for the element type. This package declares types and operations for manipulating vectors. (These are vectors in the sense of flexible arrays and not the mathematical vectors used for linear algebra as in the vectors and matrices packages mentioned earlier.) As well as subprograms for adding, moving and removing elements there are also generic subprograms for searching, sorting and iterating over vectors.

Ada.Containers.Doubly_Linked_Lists – this is a generic package with parameters giving the element type and "=" for the element type. This package declares types and operations for manipulating doubly-linked lists. It has similar functionality to the vectors package. Thus, as well as subprograms for adding, moving and removing elements there are also generic subprograms for searching, sorting and iterating over lists.

Ada.Containers.Hashed_Maps – this is a generic package with parameters giving a key type and an element type plus a hash function for the key, a function to test for equality between keys and "=" for the element type. It declares types and operations for manipulating hashed maps.

Ada.Containers.Ordered_Maps – this is a similar generic package for ordered maps with parameters giving a key type and an element type and "<" for the key type and "=" for the element type.

Ada.Containers.Hashed_Sets – this is a generic package with parameters giving the element type plus a hash function for the elements and a function to test for equality between elements. It declares types and operations for manipulating hashed sets.

Ada.Containers.Ordered_Sets – this is a similar generic package for ordered sets with parameters giving the element type and "<" and "=" for the element type.

There are then another six packages with similar functionality but for indefinite types with corresponding names such as Ada.Containers.Indefinite_Vectors.

Ada.Containers.Generic_Array_Sort – this is a generic procedure for sorting arrays. The generic parameters give the index type, the element type, the array type and "<" for the element type. The array type is unconstrained.

Finally there is a very similar generic procedure Ada.Containers.Generic_Constrained_Array_Sort but for constrained array types.

It is hoped that the above list gives a flavour of the capability of the package Containers. Some examples of the use of the facilities will be given in a later paper.

Finally, there are further packages for manipulating times (that is of type Ada.Calendar.Time and not Ada.Real_Time.Time and thus more appropriate in a discussion of the predefined library than the real-time features). The package Ada.Calendar has a number of obvious omissions and in order to rectify this the following packages are added.

Ada.Calendar.Time_Zones – this declares a type Time_Offset describing in minutes the difference between two time zones and a function UTC_Time_Offset which given a time returns the difference between the time zone of Calendar at that time and Greenwich Mean Time (alias UTC or Coordinated Universal Time). It also has an exception which is raised if the time zone of Calendar is not known (maybe the clock is broken).

Ada.Calendar.Arithmetic – various types and operations for coping with leap seconds.

Ada.Calendar.Formatting – further types and operations for dealing with leap seconds and time zones and related matters.

Most of the new calendar features are clearly only for the chronological addict but the need for them does illustrate that this is a tricky area. However, a feature that all will appreciate is that the package Ada.Calendar.Formatting includes the following declarations

```
type Day_Name is (Monday, Tuesday, Wednesday,
        Thursday, Friday, Saturday, Sunday);

function Day_Of_Week(Date: Time) return Day_Name;
```

There is also a small change in the parent package Ada.Calendar itself. The subtype Year_Number is now

```
subtype Year_Number is Integer range 1901 .. 2399;
```

This reveals confidence in the future of Ada by adding another three hundred years to the range of dates.

4 Conclusions

This overview of Ada 2005 should have given the reader an appreciation of the important new features in Ada 2005. As mentioned earlier, integration of the final text to produce the Amendment remains to be done and this might lead to some very small adjustments. However, there should be absolutely no changes at the level of detail presented here. Some quite promising features failed to be included partly because the need for them was not clear and also because a conclusive design proved elusive. We might think of them as Forthcoming Attractions for any further revision!

Some esoteric topics have been omitted in this overview; they concern features such as: streams, object factory functions, the partition control system in distributed systems, partition elaboration policy for high integrity systems, a subtlety regarding overload resolution, the title of Annex H, quirks of access subtypes, rules for pragma Pure, and the classification of various units as pure or preelaborable.

Further papers will expand on the six major topics of this overview in more detail.

It is worth briefly reviewing the guidelines (see Section 2 above) to see whether Ada 2005 meets them. Certainly the Ravenscar profile has been added and the problem of mutually dependent types across packages has been solved.

The group A items were about real-time and high-integrity, static error checking and interfacing. Clearly there are major improvements in the real-time area. And high-integrity and static error checking are addressed by features such as the **overriding** prefix, various pragmas such as Unsuppress and Assert and additional Restrictions identifiers. Better interfacing is provided by the pragma Unchecked_Union and the Mod attribute.

The group B items were about improvements to the OO model, the need for a Java-like interface feature and better interfacing to other OO languages. Major improvements to the OO model are brought by the Obj.Op notation and more flexible access types. The Java-like interface feature has been added and this provides better interfacing.

The final direct instruction was to incorporate the vectors and matrices stuff and this has been done. There are also many other improvements to the predefined library as we have seen.

It seems clear from this brief check that indeed Ada 2005 does meet the objectives set for it.

Finally, I need to thank all those who have helped by reviewing earlier drafts of this paper. There are almost too many to name, but I must give special thanks to Randy Brukardt, Pascal Leroy and Tucker Taft of the ARG, to my colleagues on the UK Ada Panel (BSI/IST/5/-/9), and to James Moore of WG9. I am especially grateful to a brilliant suggestion of Randy Brukardt which must be preserved for the pleasure of future generations. He suggests that this document when complete be called the Ada Language Enhancement Guide. This means that if combined with the final Ada Reference Manual, the whole document can then be referred to as the ARM and ALEG. Thanks Randy.

References

[1] ISO/IEC JTC1/SC22/WG9 N412 (2002) Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of the Amendment.

[2] ISO/IEC 8652:1995/COR 1:2001, *Ada Reference Manual – Technical Corrigendum 1*.

[3] S. T. Taft et al (eds) (2001) *Consolidated Ada Reference Manual*, LNCS 2219, Springer-Verlag.

[4] ISO/IEC TR 24718:2004 (2004) *Guide for the use of the Ada Ravenscar Profile in high integrity systems*. This is based on University of York Technical Report YCS-2003-348 (2003).

[5] ISO/IEC 13813:1997 (1997) *Generic packages of real and complex type declarations and basic operations for Ada (including vector and matrix types)*.

[6] J. G. P. Barnes (1998) *Programming in Ada 95*, 2nd ed., Addison-Wesley.

Ada-Europe 2004 Sponsors

ACT Europe
Contact: Zépur Blot
8 Rue de Milan, F-75009 Paris, France
Tel: +33-1-49-70-67-16 Fax: +33-1-49-70-05-52
Email: sales@act-europe.fr URL: www.act-europe.fr

Aonix
Contact: Anne Chapey
66/68, Avenue Pierre Brossolette, 92247 Malakoff, France
Tel: +33-1-41-48-10-10 Fax: +33-1-41-48-10-20
Email : info@aonix.fr URL : www.aonix.fr

Artisan Software Tools Ltd
Contact: Emma Allen
Suite 701, Eagle Tower, Montpellier Drive, Cheltenham, GL50 1TA, UK
Tel: +44-1242-229300 Fax: +44-1242-229301
Email : info.uk@artisansw.com URL : www.artisansw.com

Green Hills Software Ltd
Contact: Christopher Smith
Dolphin House, St Peter Street, Winchester, Hampshire, SO23 8BW, UK
Tel: +44-1962-829820 Fax: +44-1962-890300
Email : chriss@ghs.com URL : www.ghs.com

I-Logix
Contact: Martin Stacey
1 Cornbrash Park, Bumpers Way, Chippenham, Wiltshire, SN14 6RA, UK
Tel: +44-1249-467-600 Fax: +44-1249-467-610
Email : info_euro@ilogix.com URL : www.ilogix.com

LDRA Ltd
Contact: Jim Kelly
24 Newtown Road, Newbury, Berkshire, RG14 7BN, UK
Tel: +44-1635-528-828 Fax: +44-1635-528-657
Email: info@ldra.com URL: www.ldra.com

Praxis Critical Systems Ltd
Contact: Rod Chapman
20 Manvers Street, Bath, BA1 1PX, UK
Tel: +44-1225-823763 Fax: +44-1225-469006
Email : sparkinfo@praxis-cs.co.uk URL : www.sparkada.com

Scientific Toolworks Inc
Contact: Matthew Bergeson
321 N. Mall Drive Suite I-201, St. George, UT 84790, USA
Tel: +1-435-627-2529 Fax: +1-877-512-0765
Email: sales@scitools.com URL: www.scitools.com

TNI Europe Limited
Contact: Pam Flood
Triad House, Mountbatten Court, Worrall Street, Congleton, Cheshire CW12 1DT, UK
Tel: +44-1260-29-14-49 Fax: +44-1260-29-14-49
Email: info@tni-europe.com URL: www.tni-europe.com